Understanding and Defending Against Malicious Identities in Online Social Networks

by

Qiang Cao

Department of Computer Science
Duke University

Date: _____

Approved:

_____
Xiaowei Yang, Supervisor

_____
Jeffrey S. Chase

_____
Bruce M. Maggs

_____
Michael Sirivianos

Dissertation submitted in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in the Department of Computer Science
in the Graduate School of Duke University
2014

ABSTRACT

Understanding and Defending Against Malicious Identities in Online Social Networks

by

Qiang Cao

Department of Computer Science
Duke University

Date: _____

Approved:

_____
Xiaowei Yang, Supervisor

_____
Jeffrey S. Chase

_____
Bruce M. Maggs

_____
Michael Sirivianos

An abstract of a dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Computer Science in the Graduate School of Duke University
2014

Abstract

Serving more than one billion users around the world, today's online social networks (OSNs) pervade our everyday life and change the way people connect and communicate with each other. However, the open nature of OSNs attracts a constant interest in attacking and exploiting them. In particular, they are vulnerable to various attacks launched through malicious accounts, including fake accounts and compromised real user accounts. In those attacks, malicious accounts are used to send out spam, spread malware, distort online voting, etc.

In this dissertation, we present practical systems that we have designed and built to help OSNs effectively throttle malicious accounts. The overarching contribution of this dissertation is the approaches that leverage the fundamental weaknesses of attackers to defeat them. We have explored defense schemes along two dimensions of an attacker's weaknesses: limited social relationships and strict economic constraints.

The first part of this dissertation focuses on how to leverage social relationship constraints to detect fake accounts. We present SybilRank, a novel social-graph-based detection scheme that can scale up to OSNs with billions of users. SybilRank is based on the observation that the social connections between fake accounts and real users, called *attack edges*, are limited. It formulates the detection as scalable user ranking according to the landing probability of early-terminated random walks on the social graph. SybilRank generates an informative user-ranked list with a substantial fraction of fake accounts at the bottom, and bounds the number of fake accounts that are ranked higher than legiti-

mate users to $O(\log n)$ per attack edge, where n is the total number of users. We have demonstrated the scalability of SybilRank via a prototype on Hadoop MapReduce, and its effectiveness in the real world through a live deployment at Tuenti, the largest OSN in Spain.

The second part of this dissertation focuses on how to exploit an attacker's economic constraints to uncover malicious accounts. We present SynchroTrap, a system that uncovers large groups of active malicious accounts, including both fake accounts and compromised accounts, by detecting their loosely synchronized actions. The design of SynchroTrap is based on the observation that malicious accounts usually perform loosely synchronized actions to accomplish an attack mission, due to limited budgets, specific mission goals, etc. SynchroTrap transforms the detection into a scalable clustering algorithm. It uncovers large groups of accounts that act similarly at around the same time for a sustained period of time. To handle the enormous volume of user action data in large OSNs, we designed SynchroTrap as an incremental processing system that processes small data chunks on a daily basis but aggregates the computational results over the continuous data stream. We implemented SynchroTrap on Hadoop and Giraph, and we deployed it on Facebook and Instagram. This deployment has resulted in the unveiling of millions of malicious accounts and thousands of large attack campaigns per month.

Contents

List of Tables

List of Figures

List of Abbreviations and Symbols

Abbreviations

OSN	Online social networks
CAPTCHA	Completely automated public Turing test to tell computers and humans apart
ML	Machine learning
URL	Uniform resource locator
HTTP	Hypertext transfer protocol
DHT	Distributed hash table
SMS	Short message service
API	Application programming interface

Acknowledgments

I am most thankful to my advisor, Professor Xiaowei Yang, for her guidance and support during my graduate study at Duke University. She taught me many skills, including how to find a promising topic, how to divide a technical problem into manageable pieces, how to think critically and write clearly, and more. There is no doubt that I will continue to benefit from this end-to-end research skill set. Even more, thanks to her trust and support, I had the chance to fully practice these skills on projects.

It is a great pleasure to have worked with Michael Sirivianos during these years. He taught me research skills and gave me many valuable suggestions on improving my English. I am particularly indebted to him for those long nights he spent on my paper drafts. When I felt frustrated about my research, he was always there to encourage me and provide constructive ideas.

I especially thank Professor Jeffery Chase and Professor Bruce Maggs for serving as my committee members and for their feedback on my research projects, conference talks, preliminary exam, and dissertation. In particular, extensive discussions with Bruce on the minimum ratio-cut problem enlarged my view and widened my understanding on friend-spam defense. I also would like to thank Professor Kamesh Munagala for pointing out the Kernighan-Lin algorithm.

I am grateful to all my collaborators during my PhD study. They include Tiago Pregueiro, Christopher Palow, Jieqi Yu, Spyridon Panagiotopoulos, Chen Liang, Angelos Stavrou, Yu Chen, Xin Wu, and Theophilus Benson. Working with you all has been a

great experience.

I want to express my gratitude to Telefonica Research and Facebook for their generous support during my internships. I also owe many thanks to Nikolaos Laoutaris, Pablo Rodriguez, Konstantina Papagiannaki, and Vijay Erramilli for thoughtful conversations at Telefonica Research and for making my days in Barcelona so enjoyable.

I am also grateful to the members of the Site Integrity team at Facebook. Christopher Palow, Jieqi Yu, Yuchun Tang, Matt Jones, Benjamin Yang, and Abe Land gave me a lot of hands-on experience and helped make SynchroTrap happen on the world's largest online social network. Thanks also to the Digraph team at Facebook for providing the graph processing infrastructure.

This PhD journey would have been much longer and less enjoyable without the support of my friends. I would like to thank Michael, Ang, Xin Liu, Xin Wu, Yu, Yang, Xuanran, Bingyang, and Hongze for being such great lab mates and office mates, and many other friends on the third floor of the LSRC building. I learned a lot from our discussions of projects, research, and life. Special thanks go to Marilyn Butler for her help with the logistics of my curricular practical trainings and exams.

Finally, I thank my family for their support and love. I thank Yi Hong for sharing her life with me. Although she is also pursuing a PhD in computer science, she did a lot to take care of things at home and allowed me to focus on my work. Whenever I felt disheartened, she always comes up with ideas to help me relax, such as cooking my favourite dishes. Even more, she always showed me her appreciation of any research achievement I made—even in the early stage when the achievement was too small to be visible to others. To my parents, Xuejian and Juxiang, and my brother, words cannot express how thankful I am for your constant support over these years.

1

Introduction

In recent years, online social networks (OSNs), such as Facebook, Twitter, Google+, and Instagram, have become platforms for people all over the world to connect and communicate with each other. Billions of users rely on those platforms to efficiently host and relay information every day. As of December 2013, Facebook had attracted more than one billion monthly active users [23], and Twitter had passed 200 million [28]. With their massive user bases, online social networks mirror the social networks among people in the real world, but enable rich social interactions that are free from real-world constraints, such as geo-location. Furthermore, as indicated by the large numbers of mobile active users on major social networking sites [23, 28], user access to OSNs expands even to the ubiquitous handheld mobile devices, which makes it easier than ever for people to maintain and foster social connections. Meanwhile, the OSN providers base their business model on valuable user bases, such as introducing online ads, online games, etc. The large ecosystem on top of online social networks makes providing such services highly profitable.

While users enjoy the convenience and fun that online social networks bring to the world, cyber criminals increasingly find them to be lucrative targets. First, the online social networks are built as open platforms. Through these platforms, attackers can easily

reach large numbers of users at little financial cost as compared to other channels on the Internet. Second, most of the users value their social connections on online social networks and place trust in them. For instance, people are more likely to click a spamming link shared by a careless friend than one they find on a random web page. Attackers can exploit this embedded trust in social connections to virally propagate malicious content. As a result, online social networks attract a constant interest in exploiting them as a venue for a variety of malicious activities, including social spam, malware distribution, online rating distortion, phishing attacks, etc. Such attacks are not only detrimental to the user experience on online social networks, but are also harmful to the advertising value and the marketability of the platforms. OSN providers expend a significant amount of resources to mitigate large attacks [86, 25]. In practice, however, the development of countermeasures is largely driven by the emergence of specific attacks. For example, the Facebook Immune System exports simple APIs that can flexibly incorporate the security specialists' observations on attack features into their filtering policies [86]. These efforts, which require a significant amount of human intervention, can fall short of large attacks in terms of effectiveness, scalability, and strategy resilience.

We focus on systematically disrupting the core element of sophisticated attacks: the malicious accounts under an attacker's control. As in the real world, user identity plays a central role in the service model of online social networks. A user is usually required to register a unique account before being able to use an OSN. A registered account is then used as this user's online identity by the OSN to link the user's online profile and various activities. In order to launch campaign attacks, attackers often manipulate a large set of malicious accounts that includes both *fake accounts* and *compromised accounts*:

• **Fake accounts:** The accounts that are created using fraudulent user information, and therefore do not correspond to real humans [38, 97]. They are called *Sybils* in online social networks.

• **Compromised accounts:** The real user accounts that are compromised by attackers

via malware [26], phishing [57], etc.

Because most malicious content and activities emanate from malicious accounts, if OSN providers can effectively uncover and flag the malicious accounts controlled by attackers, they can then prevent those accounts from launching future attacks.

This dissertation aims to secure large OSNs by understanding, detecting, and throttling malicious accounts in online social networks. The goal of this work is to design robust and practical defense schemes that remain effective under a broad range of circumstances and can serve as the foundation to achieve attack resilience on online social networks. At a high level, our idea is to leverage the fundamental weaknesses of attackers to effectively defeat them. Targeting the attackers' fundamental weaknesses empowers the achievement of robustness in our defense designs in a way that can undermine the incentives behind large attacks. In this dissertation, we have explored two dimensions of attackers' fundamental weaknesses: social relationship constraints and economic constraints. By exploiting each category of these constraints, we designed and implemented two defense systems: *SybilRank* and *SynchroTrap*.

1.1 Rationale of exploiting the weaknesses of attackers

Although attackers appear to be destructive when abusing an open OSN platform, they are restrained by a set of fundamental constraints that are not common to normal users. Motivated by profit, attackers usually manipulate a disproportionally large number of malicious accounts and aim at campaign missions that meet specific goals such as timing, campaign targets, monetary budgets, etc. Those preconditions for profitable attacks lead to fundamental constraints that make the malicious accounts under an attacker's control in aggregate different from real users. We have investigated attackers' constraints and their subsequent exhibition in groups of malicious accounts, in order to design effective schemes to systematically capture them.

• **Social relationship constraints:** Attackers can control an extraordinary large number of accounts, but they have limited social networking resources, i.e., being acquaintant with limited real humans. As a result, the bulk fake accounts they craft from scratch are only able to obtain limited social engagement from real users. The social edges in a readily available OSN social graph represent such a type of strong and robust social engagement. It is widely believed [100, 99, 90, 93] that establishing and retaining social edges to real users requires non-trivial human efforts. Therefore, we propose *SybilRank*, which uses the social edges connecting to real users as a resource test to detect the fake accounts created by attackers.

• **Economic constraints:** As most of the attackers are economically motivated, they are concerned with the cost and revenue of running large campaign attacks. The attack cost mainly includes the expense of the computing and operating resources used for campaigns. In particular, attackers have limited hosting infrastructure (i.e., compromised machines, IP proxies, etc.), as well as limited human labor to operate campaigns and manage a disproportionally large number of malicious accounts. Additional computing and operating resources raise the cost to attackers. More importantly, the quality of campaign accomplishment determines the revenue of an attack. Campaign missions are usually time sensitive, as overdue missions can result in substantial reduction in the profit. Besides, a significant fraction of attackers' missions are driven by the demands from the underground market [78, 78]. Those missions are often issued with clear requirements, i.e., collecting a large number of likes for a particular set of Facebook pages.

The above infrastructure, mission, and time constraints constitute an inherent economic weakness point of the attacks. Such economic constraints together affect the behavior of the active malicious accounts under an attacker's control, leading to distinguishable activity patterns exhibited on those accounts. That is, malicious accounts an attack controls tend to repeatedly use the same set of computing resources (e.g., IP addresses), perform actions targeting the same set of objects (e.g., pages, followees, etc.), and com-

plete missions before deadlines. Therefore, we propose *SynchroTrap*, which systematically uncovers large groups of active malicious accounts based on their behavior patterns as exhibited in the readily available user-action logs.

1.2 Exploiting social graphs to detect fake accounts

The social graph contains a wealth of information about the users. Because attackers have limited numbers of acquaintances in the real world, the large number of fake accounts they create can attract only limited social connections from real users, called *attack edges*, on the OSN social graph. This graph property enables us to distinguish fake accounts from real users via large-scale graph analysis.

The social-graph-based defense was first introduced in distributed systems such as peer-to-peer networks [100, 71, 67] to secure collaborative tasks, including file sharing, unwanted traffic filtering, distributed hash table (DHT) routing, etc. Subsequently, attack-resilient system designs based on the social graph property have been proposed for specific popular applications [90, 82], such as online vote collections and online marketplaces. However, the existing solutions fall short when applied to OSNs. First, the distributed defense schemes designed in decentralized settings are often not sufficiently efficient in a centralized OSN that may host billions of users. This is because distributed schemes usually assume the workload can be distributed to all participants, which otherwise is prohibitively expensive for a centralized computation entity. Second, although the design of attack-resilient applications leverages the domain knowledge to achieve application-specific goals such as limiting the numbers of votes collected from fake accounts, a design that has been optimized for one application may not be applicable to other systems. Therefore, we aim to design an effective, scalable, and immediately deployable defense scheme that helps online social network providers to uncover fake accounts.

We propose SybilRank, the first deployed social-graph-based defense system that en-

ables effective detection of fake accounts in large OSNs. SybilRank is based on the same assumption as in previous social-graph-based schemes: social relationships with real users comprise a precious resource to attackers. In a social graph where fake accounts have limited social connections to real users, an early-terminated random walk starting from a non-Sybil node has a higher degree-normalized landing probability of reaching a non-Sybil node than a Sybil node. Consequently, we formulate Sybil detection as scalable user ranking, and derive quality ranking according to the landing probability of early-terminated random walks. As a result, SybilRank generates a quality-ranked list with a substantial proportion of fake accounts at the bottom. In addition, SybilRank's user ranking can be efficiently computed using a parallel computing framework such as Hadoop MapReduce, resulting in a practical solution for large OSNs. SybilRank has been deployed in Tuenti, the largest online social network in Spain.

1.3 Exploiting economic constraints to uncover active malicious accounts

Profit-driven campaign attacks are bound by fundamental economic constraints. There have been research efforts in examining the characteristics of the hosting infrastructure used for spamming [95, 83, 96] and the role of the underground economy in web and social network abuse [61, 5, 78, 87, 88]. We take one step further to explicitly exploit the fundamental economic constraints of the attack ecosystem, in order to design a robust and generic defense scheme.

Our intuition is that the two sets of economic constraints an attacker faces, i.e., limited infrastructure resources and strict mission requirements, lead to *loosely synchronized* action patterns on groups of active malicious accounts: specifically, they repeatedly perform the same actions around the same time for a sustained time period. In contrast, the behavior of a set of legitimate users is often more diverse. Therefore, we propose SynchroTrap, a generic defense scheme that systematically uncovers groups of active malicious accounts

by detecting their loosely synchronized actions. SynchroTrap continuously monitors the activities of each user account, and signals an anomaly if it detects that a large group of users perform loosely synchronized actions within an operator-defined time period (e.g., one week). SynchroTrap introduces a tuple abstraction, with an explicit constraint field for time-stamped user actions, that can express both the infrastructure constraints and mission constraints. At its core, SynchroTrap has a parallel clustering algorithm that groups users who have performed similar actions under a tuple-similarity metric. By accurately correlating time-stamped user actions according to the constraint field, SynchroTrap attains the capability of effectively capturing the loose synchronization among active malicious accounts. This design explicitly attacks the economic weakness point of attackers and makes it difficult for them to change the attack patterns or account features to evade detection. We also designed SynchroTrap as an incremental processing system and have implemented it on Hadoop and Giraph. SynchroTrap is able to handle large OSNs and has been deployed at Facebook and Instagram.

1.4 Contributions

The first contribution of this dissertation is its lessons on how to build a practical social-graph-based defense system to aid the detection of fake accounts in large OSNs. SybilRank achieves both the scalability to handle large OSNs and the best provable guarantee of suppressing false rates. We discovered that the scalable user ranking used by SybilRank greatly benefits the current human-in-the-loop OSN countermeasures, given that existing binary detectors yield high false rates and are computationally expensive to run on large OSNs. Our implementation and deployment of SybilRank in Tuenti demonstrated that SybilRank detected a substantial fraction of fake accounts in Tuenti and significantly improved the efficiency of Tuenti's manual inspection.

The second contribution of this dissertation is its lessons on understanding and ex-

ploiting attackers' economic constraints for the purpose of robust defense. Despite the difficulty of directly tracking those constraints, we found that they are revealed by the loosely synchronized action patterns on groups of malicious accounts. This enables the detection of malicious accounts by examining the readily available user-action logs. After addressing important algorithmic and system challenges in large OSNs, we designed and implemented a practical detection system that can process the continuous and sheer volume of user activities over time. We deployed SynchroTrap at Facebook and Instagram. It uncovered a large fraction of malicious accounts that were not detectable by Facebook's existing automatic countermeasures.

The proposals in this dissertation form a defense in depth for online social networks. The social-graph-based approach SybilRank complements the activity-based approach SynchroTrap by detecting dormant fake accounts at an early stage; SynchroTrap enables the detection of compromised real-user accounts that have befriended many real users.

The rest of this dissertation is organized as follows. Chapter 2 contains a description of related work and introduces the background. In Chapter 3, we present SybilRank, an effective and scalable social-graph-based scheme to detect fake accounts. In Chapter 4, we present SynchroTrap, which uncovers large groups of malicious accounts by detecting their loosely synchronized actions. Future work is discussed in Chapter 5; Chapter 6 contains the summary of this dissertation.

2

Background and Related Work

2.1 Overview

In this chapter, we introduce the background of our work and provide an overview of related work in this field. A large body of work exists in the research community [100, 99, 93, 38, 86, 94, 46, 104], and extensive efforts have been made in practice [12, 6, 3, 10, 86] to detect attacks in OSNs. Beyond the detection of attacks, there also have been research efforts to design Sybil-resilient systems [85, 101, 71, 79, 67, 84, 82, 47, 76], and to understand the infrastructure (e.g., botnets) used for attacks on the web [50, 51, 96, 104, 54, 83]. First we briefly survey countermeasures currently used in practice and identify their limitations. We then describe the research efforts that have attempted to mitigate a variety of problems caused by malicious accounts. According to the approaches that were used, we categorize these efforts into social-graph-based Sybil defense, activity-based schemes, and Sybil-resilient systems. Last, we summarize existing work on web abuse from the perspective of the infrastructure an attacker uses.

2.2 Countermeasures in practice

OSN providers have invested significant resources in resisting attacks, and in sterilizing the social graphs and user-generated content hosted on their platforms. Those efforts have included improving user interface (UI) to better protect user credentials [6], employing CAPTCHA challenges to slow down aggressive attacks [1], using photo-based social authentication to verify user accounts when they are perceived as abnormal [13], building cross-company partnerships to collaboratively suppress Internet threats [3, 10], etc. To facilitate the extension of the defense line, OSN providers tend to add more system support for a variety of supervised machine learning-based approaches to capture malicious activities [86, 68]. For example, the Facebook Immune System provides interfaces to flexibly integrate attack features into filtering policies [86]. In addition to the efforts to improve the systems, providers also afford a large amount of human labor to gain intuition on attack features and to manually review suspicious and user-reported accounts and content. This is because automated supervised machine-learning approaches require quality input to train classifiers and inevitably result in false rates. The use of human labor is meant to improve the quality of both the input and output of these automated classifiers. For example, Tuenti has dedicated 14 full-time security employees to detecting, manually verifying, and suspending suspicious accounts [12].

2.2.1 Survey on Tuenti's defense toolchain

We conducted a survey with Tuenti [17] to understand its human-in-the-loop countermeasures. The large number of distinct reasons for the creation of fake accounts [9, 4, 7] and the compromising of real user accounts has led to numerous and diverse malicious behaviors that have been manifested as profile features and activity patterns. Existing automated feature-based (e.g., supervised learning-based) approaches [86] suffer from high false negative and positive rates, due to the large variety and unpredictability of legiti-

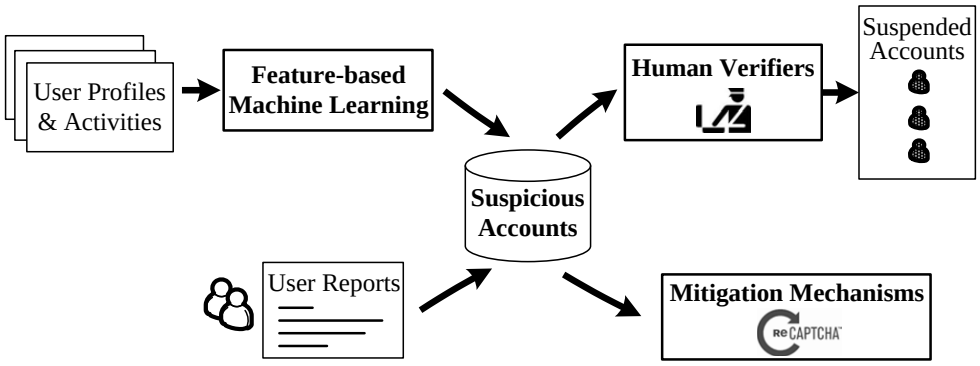


FIGURE 2.1**: Defense toolchain used in Tuenti to mitigate attacks launched by malicious accounts.**

mate and malicious OSN users' behaviors. In response to the inapplicability of automated account suspension, OSNs usually employ CAPTCHA [1] and photo-based social authentication [13] to rate-limit suspected users, or manually inspect the features of accounts reported as abusive (flagged) by other users [87, 12]. Figure 2.1 shows Tuenti's defense toolchain, which represents typical practice.

The manual inspection involves matching profile photos to the age or address provided by the user, understanding the natural language in posts, examining a user's friends, etc [12]. The inspectors also use simple tools to parse account activity and to compile IP statistics for suspicious accounts. These tasks require human intelligence and intuition, however, which renders them hard to automate and scale. Tuenti receives on average 12,000 reports regarding abusive accounts and 4,000 reports for inappropriate photos per day; an employee can review an average of 250 to 350 reports per hour. Among the reported suspicious accounts, only $\sim$5% are indeed fake [12]. On average, Tuenti's manual inspection team, which consists of 14 employees, deletes 800 to 1500 fake accounts per day.

Similar to Tuenti's practice, most of the automated OSN countermeasures that scrape malicious accounts and activities at scale involve humans in the loop. They rely on human

intelligence to improve the quality of the train data and the output. As a result, OSN providers pursue practical defense schemes that achieve high effectiveness, scalability, and robustness, but with little dependence on human intervention.

2.3 Social-graph-based Sybil defenses

Sybil (fake account) detection has been the focus of the research community [90, 91, 100, 99, 42, 93]. Existing work distinguishes Sybils from non-Sybil users using the social graph property that Sybils have limited social connections to real users.

The decentralized protocols SybilGuard and SybilLimit use random-walk traces to infer Sybils through route intersection [100, 99]. SybilGuard bounds the number of accepted Sybils per attack edge to $O(\sqrt{n}\log n)$. SybilLimit improves over SybilGuard and limits the number of accepted Sybils per attack edge to $O(\log n)$, when the number of attack edges is $o(\frac{n}{\log n})$. Applying SybilLimit to an online social network leads to a computation cost of $O(\sqrt{m}n\log n)$, where m is the number of edges. SybilInfer [42] is also based on random-walk traces. It uses the Metropolis-Hastings (MH) algorithm to sample N (N is a system parameter) non-Sybil user sets according to the random-walk traces. Counting the user occurrence in N samples, SybilInfer generates a marginal probability for each node being non-Sybil. It incurs a computation cost of $O(n(\log n)^2)$, but does not specify any upper-bound guarantee on false rates [98].

The flow-based scheme GateKeeper [91] improves over SumUp [90]. At its heart is a vote-collection scheme; thus it detects Sybils by finding users whose votes cannot be accepted. It first computes the multi-source max flow from the voters to a single trusted vote collector and then distributes *tickets*, in a breadth-first-search (BFS) manner, in order to assign capacity to the links of the graph and to bound the number of accepted Sybil votes by the attack edges. A vote is accepted only if it is on a path with non-zero flow to the collector. Although the ticket distribution is efficient, it relies on a strong assumption that

requires balanced social graphs [98]. Viswanath et al. [93] proposed community detection algorithms such as Mislove's algorithm [72] to detect Sybils. However, community detection (CD) algorithms rarely provide provable guarantees, and Mislove's CD algorithm costs $O(n^2)$.

In addition, the limited attack edges between Sybils and non-Sybil users can be modeled by a graph cut with a small conductance. However, finding cuts with the minimum conductance in a graph is known to be NP-hard [100]. An $O(\log n)$-approximation requires computation of $O(m^2 \log m)$ [62]. Although it has been proven that there exists a randomized polynomial-time algorithm to find an $O(\sqrt{\log n})$-approximation, the single step of solving the semi-definite program (SDP) costs $O(n^2)$ [31].

The above schemes either do not exhibit sufficient accuracy or they incur a prohibitive computational cost in a centralized OSN setting. Our social-graph-based proposal SybilRank differs from previous work in that it achieves equivalent or higher accuracy (both analytically and empirically), and it is computationally more efficient, i.e., it has $O(n \log n)$ cost. More importantly, SybilRank can be implemented on parallel computing framework such as Hadoop, where it can achieve the scalability that is required to handle large OSNs.

2.4 Activity-based defense schemes

Recently, there have been increasing efforts to design defenses by catching the anomalies in user activities on OSNs [34, 94, 46]. Wang et al. [94] used k-gram-based clickstream analysis to obtain normal user clusters and fake account clusters. They proposed several models to analyze the clickstreams and applied them to activity traces of 16K RenRen users. The premise is that fake accounts act similarly on OSNs and can be clustered into groups using their clickstreams. However, because k-gram-based clickstream analysis makes use of the ordering of different types of clicks, it cannot capture the constraints behind the same type of clicks, e.g., page likes. Besides, the k-gram-based decomposi-

tion of user clickstreams may lead to a high-dimensional gram-count vector for each user. Clustering high-dimensional vectors incurs both a scalability challenge in large OSNs and the accuracy issue caused by the curse of dimensionality [64]. COMPA [46] identifies compromised accounts using statistical models to detect sudden changes in user behavior, i.e., messages sent out. This approach bears similarity to existing countermeasures in practice that monitor user activity and seek abrupt changes [86], but is a comprehensive approach customized for message spam. Beutel et al. [34] used co-clustering to spot large sets of fraudulent page likes at Facebook. Their co-clustering algorithms identify the rows (users) in a matrix that share similar values (the time-stamp of a like) at multiple columns (pages). However, because this approach relies on the assumption of a single action between a user and a page, it is difficult to generalize it to other applications. That is, it does not fit time-series data because it uses greedy search to find similar rows and the search space increases exponentially with the length of the time-series data. Also, because this approach uncovers only a fixed number of clusters surrounding the preset seeds, it can miss a non-trivial number of clusters due to the local search technique it uses.

Our work on SynchroTrap is also motivated by the activity-based technique. However, it differs from existing work by explicitly targeting the economic constraints behind malicious users' activities. This explicit targeting leads to a robust defense that is difficult for attackers to circumvent, i.e., they must afford more monetary cost.

2.5 Sybil-resilient systems

Unlike Sybil detection mechanisms that aim at explicitly distinguishing Sybils from non-Sybil users, a Sybil-resilient design aims to make a particular application (e.g., a recommendation system or a DHT) resilient to Sybil attacks [85, 101, 71, 67, 84, 82]. Resnick et al. [85] proposed sum-Sybil-proof protocols for user interactions such that the total potential damage that Sybils can cause is limited to the total trust they obtain from non-Sybil

users. DSybil [101], a recommendation system based on user feedback, has an online learning algorithm that provides a provable guarantee on the number of malicious recommendations. Ostra [71] bounds the unwanted communication from malicious users to the number of attack edges they have established. SumUp [90] leverages the social graph to bound bogus votes in online voting systems. Lesniewski-Laas et al. [67] built Sybil-resilient DHTs based on social connections between peers. In their approach, routing tables and successor lists are constructed using nodes sampled by short random walks. Quercia et al. [84] proposed MobID to mitigate Sybil attacks in mobile networks based on the social relationships between mobile users. Post et al. [82] proposed Bazaar, an improved reputation system for online marketplaces. In Bazaar, the aggregate amount of fraudulent transactions that a malicious user can conduct is bounded by the amount of transactions in which he has successfully participated.

The key advantage of the above Sybil-resilient system designs is that they can use application-specific knowledge to achieve application-specific goals such as limiting the number of votes collected from Sybil users [90]. However, a Sybil-resilient design optimized for an application may not be applicable to other systems, whereas a Sybil detection mechanism can detect Sybils in other systems as long as users in those systems can be mapped to an online social network.

2.6 Infrastructure-level schemes

Extensive work has been done on measuring and detecting the hosting infrastructures, such as botnets, that are used for web abuse [50, 51, 96, 104, 54]. Most of the existing work has used network-level features of the hosting infrastructure to detect and defend against botnet-borne web spam and abuse. BotMiner [50] and BotSniffer [51] detect bots via network traffic analysis, under the assumption that the bots share the same command and control channel and respond to commands in a similar way. Those approaches require to

monitor the network traffic of clients' computers. Xie et al. proposed UDmap to identify dynamic IP addresses for email spam filtering [95]. Qian et al. [83] explored the granularity of IP clusters on which IP blacklists are used to filter spam, and proposed to cluster IP addresses based on a combination of BGP prefixes and DNS information. AutoRE [96] generates regular expressions for URLs embedded in messages, in order to characterize the spamming botnets whose traffic is bursty in time and also dispersed through the IP address space. AutoRE needs to inspect message content and is limited to URL-embedded payload. BotGraph [104] is a graph-based approach that detects botnets based on the premise that the IP addresses of bots are shared among spammers. However, this coarse-grained approach does not consider the time correlation. It may mix malicious users and legitimate users behind dynamic IP addresses, IP proxies, and shared Internet-access points. Hao et al. [54] used a machine-learning (ML) approach to infer the reputation of an email sender based on network-level spatio-temporal features. This approach can detect spammers if one obtains high-quality training data that covers a large fraction of spam.

Although the above approaches are applicable to OSNs, they can fall short of sufficiently dealing with the emerging threats that exploit unique functionalities on OSNs (e.g., user following, page like, etc.), and that do not necessarily manifest the features they seek. Our proposal SynchroTrap addresses these important limitations. Based on the understanding of the attackers' economic constraints, SynchroTrap is designed as a generic framework that can precisely detect various attacks in OSNs.

3

SybilRank: Aiding the Detection of Fake Accounts in Large-Scale Social Online Services

In this chapter, we describe how we exploit the social friendship constraint to aid the detection of fake accounts in large online social networks.

3.1 Introduction

The popularity surge of online social networking services such as Facebook, Twitter, Digg, LinkedIn, Google+, and Tuenti has been accompanied by an increased interest in attacking and manipulating them. Due to their open nature, such services are particularly vulnerable to the Sybil attack [45], in which a malicious user can create multiple fake OSN accounts.

It has been reported that 5-6% Facebook accounts are fake [15], and that 1.5 million fake or compromised Facebook accounts were for sale in February 2010 [7]. Fake (Sybil) OSN accounts can be used for various purposes [9, 4, 7]. For instance, they enable spammers to abuse an OSN's messaging system to post spam [48, 87], or waste an OSN advertising customer's resources by making him pay for online ad clicks or impressions from or to fake profiles. Fake accounts can also be used to acquire users' private

contact lists [35]. Sybils can also use the "+1" button to manipulate Google search results [11] or to pollute location crowd-sourcing results [8]. Furthermore, fake accounts can be used to access personal user information [47] and perform large-scale crawls over social graphs [76].

If an OSN provider can effectively detect Sybil nodes in its system, it can improve the experience of its users and their perception of the service by stemming annoying spam messages and invitations. It can also increase the marketability of its user base and its social graph. In addition, it can enable other online services or distributed systems to treat a user's online social network identity as an authentic digital identity, a vision foreseen by recent efforts such as Facebook Connect [2].

We therefore aim to answer the question: "can we design a social-graph-based mechanism that enables a multi-million-user OSN to pick up accounts that are very likely to be Sybils?" The answer can help an OSN stem malicious activities in its network. For example, it can enable human verifiers to focus on accounts that are likely to be fake. It can also guide the OSN in sending CAPTCHA [30] challenges to suspicious accounts, while running a lower risk of annoying legitimate users.

We present the design (§4.4), implementation (§3.6), and evaluation (§3.7,§3.8,§3.6) of SybilRank. SybilRank is a Sybil inference scheme customized for OSNs whose social relationships are bidirectional.

Our design is based on the same assumption as in previous work on social-graph-based defenses [90, 91, 100, 99, 42, 93]: that OSN Sybils have a disproportionately small number of connections to non-Sybil users. Differently, our system achieves a significantly higher detection accuracy at a substantially lower computational cost. It can also be implemented in a parallel computing framework, e.g., MapReduce [43], enabling the inference of Sybils in OSNs with hundreds of millions of users.

Social-graph-based solutions uncover Sybils from the perspective of already known non-Sybil nodes (*trust seeds*). Unlike [90] and [91], SybilRank's computational cost does

not increase with the number of trust seeds it uses (§3.4.4). This efficiency facilitates the use of multiple seeds to increase the system's robustness to seed selection errors, such as designating a Sybil as a seed. It also allows the OSN to cope with the multi-community structure of online social graphs [93] (§3.4.1).

We show that SybilRank outperforms existing approaches [99, 42, 91, 72, 59]. In our experiments, it detects Sybils with at least 20% lower false positive and negative rates (§3.7) than the second-best contender in most of the attack scenarios. We also deployed SybilRank on Tuenti, the leading OSN in Spain (§3.8). Almost 100% and 90% of the 50K and 200K accounts, respectively, that SybilRank designated as most suspicious, were indeed fake. This compares very favorably to the $\sim$5% hit rate of Tuenti's current abuse-report-based approach.

Our contributions. In summary, this work makes the following contributions:

• We re-formulate the problem of Sybil detection in OSNs. We observe that due to the high false positives of binary Sybil/non-Sybil classifiers, manual inspection needs to be part of the decision process for suspending an account. Consequently, our aim is to efficiently derive a quality ranking in which a substantial portion of Sybils ranks low. This enables the OSN provider to focus its manual inspection efforts on the end of the list, where it is more likely to encounter Sybils. Moreover, our ranking can inform the OSN on whether to challenge suspicious users with CAPTCHAs.

• The design of SybilRank, an effective Sybil-likelihood ranking scheme for OSN users based on the landing probability of short random walks. We efficiently compute this landing probability using early terminated power iteration. In addition, SybilRank copes with the multi-community structure of social graphs [93] by using multiple seeds at no additional computational cost. We thoroughly compare SybilRank with existing Sybil detection schemes.

• We implemented SybilRank in a parallel computing framework [43] using commodity

machines. Our implementation illustrates that it is practical and computationally efficient to use the social graph to detect Sybil users in large-scale OSNs.

• We have deployed SybilRank on Tuenti, an OSN with 11 million users. Our system has allowed Tuenti operations to detect, within the same time period, 18 times more fake accounts than their current process. We expect that SybilRank will also work for other OSNs such as LinkedIn, where user connections are more likely to reveal real-life relationships.

3.2 Models, Assumptions, and Goals

We now introduce our system and threat model, and our design assumptions and goals.

3.2.1 System and threat model

In online social networks (OSNs), social relationships exist between users who communicate, collaborate, or befriend each other. We consider bilateral social relationships and model an OSN as an *undirected* graph $G = (V, E)$, where each node in V corresponds to a user in the network, and each edge in E corresponds to a bilateral social relationship. In the social graph G, there are $n = |V|$ nodes, $m = |E|$ undirected edges, and a node v has a degree of $deg(v)$. We assume that the OSN provider, e.g., Tuenti, has access to the entire social graph.

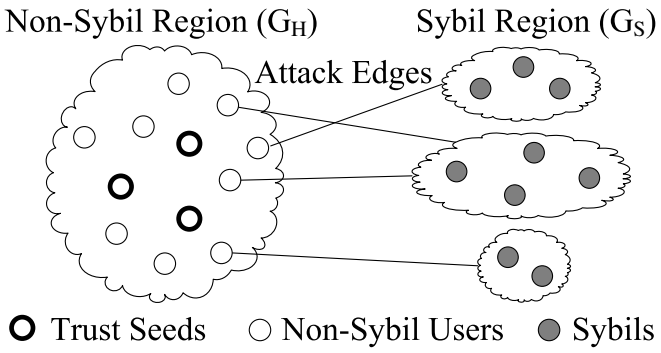


FIGURE 3.1: **Non-Sybil region, Sybil region, and attack edges in an OSN under a Sybil attack. All Sybils created by malicious users are placed into the Sybil region. The Sybil collective may not be well connected.**

In our threat model, attackers can launch Sybil attacks [45, 40] by creating many fake identities. As in previous work [100, 99, 42], we divide the node set V into two disjoint sets H and S, representing non-Sybil and Sybil users respectively, as shown in Figure 3.1. We denote the *non-Sybil region* G_H as the subgraph induced by the non-Sybil user set H, which includes all non-Sybil users and the links among them. Similarly, the *Sybil region*

G_S is the subgraph induced by S. The non-Sybil and the Sybil regions are connected by g *attack edges* between Sybils and non-Sybil users.

3.2.2 Assumptions

We make the following assumptions, which share common elements with prior social-graph-based Sybil defenses [100, 99, 42, 90, 91].

Social graph. The social graph is undirected. The non-Sybil region G_H is well connected and non-bipartite. In this case, random walks on the graph can be modeled as an irreducible and aperiodic Markov chain [33]. This Markov chain is guaranteed to converge to a stationary distribution in which the landing probability on each node after sufficient steps is proportional to its degree.

Limited attack edges. We assume that Sybils establish a limited number of attack edges due to the difficulty of soliciting and maintaining reciprocal social relationships. SybilRank is designed for large scale attacks, where fake accounts are crafted and maintained at a low cost and are therefore unable to befriend many real users. SybilRank will not detect benign pseudonyms that are created by users to distinguish among components of their social lives such as friends, family, professional associations, etc, because each of these pseudonyms retains sufficient social connections to real users. Nor does SybilRank aim at detecting accounts of real users with good connectivity that have been compromised and used for malicious purposes. To SybilRank these accounts are indistinguishable from non-Sybil accounts. Furthermore, SybilRank can be deployed over a social graph that includes only strong-relationship edges. To this end, OSNs can employ user interfaces that encourage the creation of links that reflect strong social ties or facilitate their inference. For instance, Google+ may consider only social connections between users that appear in mutually close circles.

Limited attack edges result in a sparse cut between the non-Sybil and the Sybil regions. Because the well-connected non-Sybil region is unlikely to have such a sparse cut [100],

there exists a significant difference between the mixing time of the non-Sybil region G_H and the entire graph G [33] (§A.1). A graph's *mixing time* is the maximum number of steps that a random walk needs to take so that the probability of landing at each node reaches the stationary distribution [33]. As in previous work [100, 99, 42], we begin with the assumption that the non-Sybil region G_H is fast mixing, i.e., its mixing time is $O(\log n)$, and consider the scenarios in which the non-Sybil region mixes more slowly in §3.4.1.

Attack edge distribution. To make our analysis tractable, we assume that Sybils randomly attach attack edges to non-Sybils. A more effective attack strategy against a social-graph-based Sybil defense is to establish attack edges close to the trust seeds. We refer to this attack as a *targeted attack*. We study how SybilRank performs under the targeted attack through experiments in § 3.7.5.

3.2.3 Goals

SybilRank aims to aid OSNs in identifying highly suspicious accounts by ranking users. In principle, SybilRank can be used to defend against fake accounts that are created on a large scale and at a low per-account cost for purposes, such as rating manipulation (e.g., with Facebook "likes"), spam, and crawls. Our design has the following main goals.

Effectiveness. The system should mostly rank Sybil nodes lower than non-Sybils (low false positives), while limiting the number of non-Sybils ranked below Sybils (low false negatives). It should be robust under various attack strategies. A very high portion of the nodes at the bottom of the ranked list should be fake. The portion of fakes can decrease as we go up the list.

Efficiency. The system should have a low computational cost. It should be able to handle large social networks with commodity machines so that OSN providers can deploy it on their clusters.

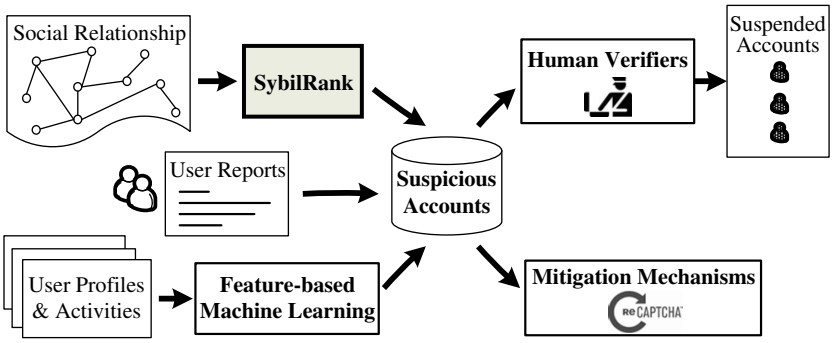


FIGURE 3.2**: SybilRank as a part of an OSN's Sybil defense toolchain.**

3.3 System overview

Motivated by the need for a method that enables human verifiers to focus on accounts that are very likely to be fake and to disable them in a timely manner, we design SybilRank as an effective and scalable social-graph-based detection scheme. SybilRank generates a quality ranked list of users with a substantial fraction of fake accounts ranked at the bottom. Our solution can be a component of the overall framework employed by OSN providers to ensure the health of their user base (Figure 3.2). The ranked list enables OSNs to focus their manual inspection efforts or to regulate the frequency with which they send CAPTCHAs to suspected users. It addresses the shortcoming of inefficient use of human labor in current human-in-the-loop countermeasures. Unlike reporting-driven manual inspection, SybilRank is a proactive method that can uncover fakes even before they interact with real users. Thus, it complements supervised-learning- and user-report-based methods.

SybilRank relies on the observation that an *early-terminated* random walk [98] starting from a non-Sybil node in a social network has a higher degree-normalized (divided by the degree) landing probability to land at a non-Sybil node than at a Sybil node. Intuitively, this observation holds because the limited number of attack edges forms a narrow passage

from the non-Sybil region to the Sybil region in a social network. When a random walk is sufficiently long, it has a uniform degree-normalized probability of landing at any node, a property referred to as the convergence of a random walk [33]. However, a shortened random walk originating from a non-Sybil node tends to stay within the non-Sybil region of the network, as the walk is unlikely to traverse one of the relatively few attack edges.

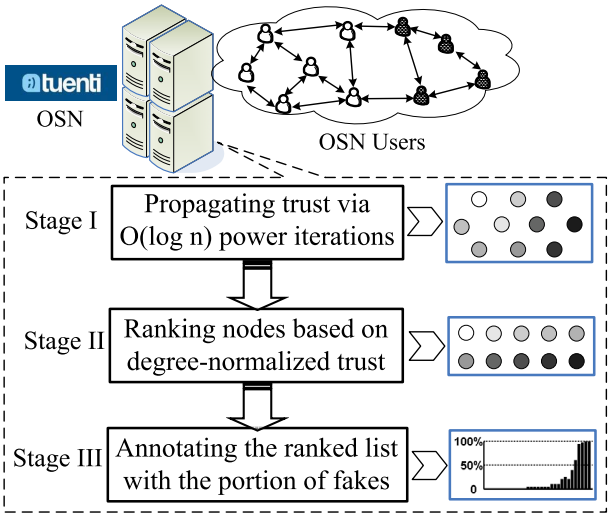


FIGURE 3.3: **SybilRank detects Sybils in three stages. Black users are Sybils. Darker nodes obtain less trust after trust propagation.**

Our key insight is to rank nodes in a social graph according to the degree-normalized probability of a short random walk that starts from a non-Sybil node to land on them. We screen out low-ranked nodes as potential fake users. A further novelty of our approach is that we use power iteration [65], a standard technique to efficiently calculate the landing probability of random walks in large graphs. This is in contrast to prior work that used a large number of random walk traces [100, 99, 42, 74, 98] obtained at a high computational cost. For ease of exposition, we refer to the probability of the random walk to land on a node as the node's *trust* and the initial probability of starting from the trust seeds as the initial trust vector.

As shown in Figure 3.3, SybilRank unveils users that are suspected to be Sybils after three stages. In Stage I, through $w = O(\log n)$ power iterations (§3.4.1), trust flows from known non-Sybil nodes (trust seeds) and spreads over the entire network with a bias towards the non-Sybil region. In Stage II, SybilRank ranks nodes based on their degree-normalized trust. In the final stage, SybilRank assigns portions of fake nodes in the intervals of the ranked list.

We next describe each stage in detail. We discuss in §3.5 the security guarantees provided by SybilRank.

3.4 System Design

We now describe the design details of each SybilRank component.

3.4.1 Propagating trust

Power iteration involves successive matrix multiplications where each element of the matrix represents the random walk transition probability from one node to a neighboring node. Each iteration computes the landing probability distribution over all nodes as the random walk proceeds per step.

Early-terminated random walks

We terminate the power iterations after $O(\log n)$ steps. The number of iterations needed to reach the stationary distribution is equal to the graph's mixing time. In an undirected graph, if a random walk's transition probability to a neighbor node is uniformly distributed, the landing probability on each node remains proportional to its degree after reaching the stationary distribution. SybilRank exploits the mixing time difference between the non-Sybil region G_H and the entire graph G in order to distinguish Sybils from non-Sybils. The intuition is that if we seed all trust in the non-Sybil region, then trust could flow into the Sybil region only via the limited number of attack edges. If we terminate the power iteration early before it converges globally, non-Sybil users will obtain higher trust than that in the stationary distribution, whereas the reverse holds for Sybils.

Trust propagation via power iteration. We define $T^{(i)}(v)$ as the trust value on node v after i iterations. Initially, the total trust, denoted as T_G ($T_G > 0$), is evenly distributed on K ($K > 0$) trust seeds $\tilde{v}_1, \tilde{v}_2, \ldots, \tilde{v}_K$:

$$T^{(0)}(v) = \begin{cases} \frac{T_G}{K} & \text{if node } v \text{ is one of the } K \text{ trust seeds} \\ 0 & \text{else} \end{cases}$$

Seeding trust on multiple nodes makes SybilRank robust to seed selection errors, as

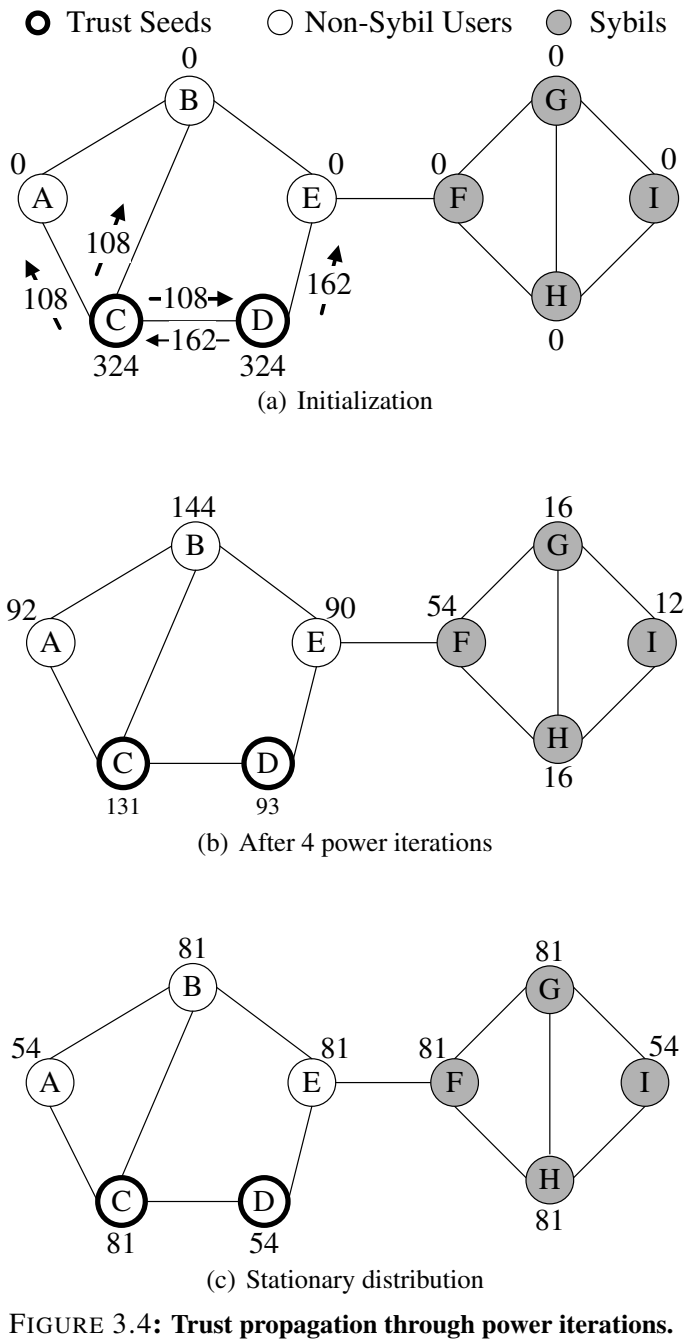


FIGURE 3.4**: Trust propagation through power iterations.**

incorrectly designating a node that is Sybil or close to Sybils as a seed causes only a small fraction of the total trust to be initialized in the Sybil region.

During each power iteration, a node first evenly distributes its trust to its neighbors. It then collects trust distributed by its neighbors and updates its own trust accordingly. This

process is shown below. Note that the total amount of trust T_G remains unchanged.

$$T^{(i)}(v) = \sum_{(u,v)\in E} \frac{T^{(i-1)}(u)}{deg(u)}$$

Early termination. With a sufficient number of power iterations, the trust vector converges to the stationary distribution: $\lim_{i\to\infty} T^{(i)}(v) = \frac{deg(v)}{2m} \times T_G$ [33]. However, SybilRank terminates the power iteration after $w = O(\log n)$ steps, thus before convergence. This number of iterations is sufficient to reach an approximately uniform distribution of degree-normalized trust over the fast-mixing non-Sybil region, but limits the trust that escapes to the Sybil region.

Alternative use of power iteration. We note that power iteration is also used by the trust inference mechanisms PageRank, EigenTrust, and TrustRank [81, 59, 52, 41], where it is executed until convergence to the stationary distribution of the complete graph [65]. At each step and with a constant probability, PageRank's random walks jump to random users, and EigenTrust/TrustRank's walks jump to trust seeds. EigenTrust/TrustRank is personalized PageRank with a customized reset distribution over a particular set of seeds.

SybilRank's random walks do not jump to random users, because this would allow Sybils to receive trust in each iteration. Besides, SybilRank's random walks do not jump to the trust seeds, because this would assign high trust to Sybils that are close to the trust seeds (§3.7.3), making the seed selection intractable.

Our design significantly simplifies the trust-seed selection process. After power iterations equal to the mixing time, each non-Sybil node obtains almost identical degree-normalized trust that is largely independent of the distance between it and the trust seeds. Due to this property, we can select any node in the non-Sybil region to be a trust seed. By contrast, previous work in trust inference [59] and Sybil defenses such as SumUp [90] tend to assign high trust or give acceptance preference to nodes that are close to the trust seeds. This implies that in these schemes, the seeds should be "far" away from the Sybils,

which makes the robust selection of trust seeds an arduous task.

In addition, we do not seek to improve the Sybil resilience of power-iteration-based trust inference mechanisms by enforcing early termination. This is because with their implicit directed connections to random nodes (PageRank) and trust seeds (EigenTrust and TrustRank), the mixing time of their modified graphs decreases significantly. A formal analysis on the convergence of random walks on these graphs is documented in [65]. Empirical reports show that EigenTrust takes only 10 iterations to converge on a 1000-node graph [59]. (similar results in PageRank [81].) Therefore, early termination after $O(\log n)$ steps cannot improve those schemes.

Example. Figure 3.4 illustrates the process of our trust propagation. In this example, we initially seed $T_G = 648$ on the non-Sybil nodes C and D. We do not normalize T_G to 1 for ease of exposition. After four power iterations, all of the non-Sybil nodes $\{A, B, C, D, E\}$ obtain higher degree-normalized trust than any of the Sybils $\{F, G, H, I\}$. However, as shown in Figure 3.4(c), if we let the power iteration converge (i.e., if we allow more than 50 iterations), the Sybils have similar trust as the non-Sybil nodes, and each node's trust is only proportional to its degree.

Coping with the multi-community structure

Existing social-graph-based defenses are sensitive to the multi-community structure in the non-Sybil region [93]. Due to the limited connectivity between communities, they may misclassify non-Sybils that do not belong to the same communities of the trust seeds as Sybils.

Distributing seeds among communities. SybilRank is intended to reach a uniform distribution of degree-normalized trust within the non-Sybil region, which is independent of the location of the non-Sybil seeds. Thus, any set of non-Sybil users is eligible for the trust seeds. Seeding trust on Sybils would degrade SybilRank's effectiveness as the initial trust on the Sybil seeds is not bounded by the limited attack edges. OSN providers can

easily identify non-Sybil users for trust seeds by manually inspecting a few users. SybilRank works with an arbitrary number of non-Sybil seeds regardless of their locations. In contrast, the seed selection is complicated for previous trust inference schemes (§3.4.1) and Sybil defenses such as SumUp [90] because they must guarantee that the seed(s) are "far" from Sybils.

We leverage SybilRank's support for multiple seeds to improve its performance in OSNs with multi-community structures. The key idea is to distribute seeds among the communities. Thus, at initialization we distribute trust in such a manner that the non-Sybil users are not starved of trust even if the inter-community connectivity is weak. We validate this design choice with simulations in §3.7.4, where SybilRank maintains a high detection accuracy in synthetic graphs with high-level community structure.

Inspecting random users for trust seeds to cover most of the communities in the non-Sybil region would require a prohibitive manual inspection workload, as indicated by the Coupon Collector's Problem [73]. Instead, we apply an efficient community detection, but not a Sybil-resilient algorithm, to obtain an estimation of the community structure [36] and then seed trust on non-Sybil users in each major community.

Estimating the multi-community structure. We use the Louvain method [36], which can efficiently detect communities. This method iteratively groups closely connected communities to improve the partition modularity. In each iteration, every node represents a community, and well-connected neighbor nodes are combined into the same community. The graph is reconstructed at the end of each iteration by converting the resulting communities to nodes and adding links that are weighted by the inter-community connectivity. Each iteration has a computational cost that is linear to the number of edges in the corresponding graph. Typically, only a small number of iterations are required. For instance, the original Tuenti graph has 11 million nodes with 1.4 billion links. The first iteration outputs a graph with only 120K nodes and 1.5 million edges. These iterations can be fur-

ther reduced by simply tweaking this multi-level aggregation mechanism to stop as soon as a manageable set of communities for seed candidate inspection has been obtained. This method can also be parallelized on commodity machines [103].

As illustrated in Figure 3.5, in each identified community we inspect a small set of random nodes ($\sim$100 in total for the 11-million-node Tuenti social network) and only seed trust at the nodes that have passed the verification. This method achieves adequate seed coverage over the non-Sybil region and avoids initiating trust in the Sybil region.

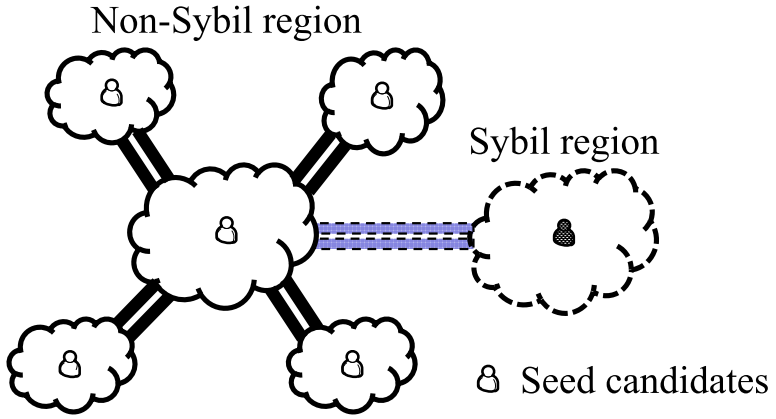


FIGURE 3.5**: Distributing seeds into multiple communities. The Sybil seed candidates (black) cannot pass the manual inspection.**

Community detection algorithms have been proposed to directly detect Sybils [93]. These algorithms seek a partition of a graph that has dense intra-community connectivity and weak inter-community connectivity. For instance, the Louvain method searches for a partition with high modularity [36]. Thus, Sybils that are not well connected to each other may be classified as non-Sybils belonging to nearby non-Sybil communities. In contrast, by placing seeds in communities SybilRank is able to uncover subsets of Sybils within Louvain-detected communities, as demonstrated by our Tuenti deployment (§3.8.3).

General CD algorithms rarely provide guarantees on errors in their classifications [98]. They need more calibration work on their algorithms or parameters to achieve high accu-

racy in Sybil detection.

Sensitivity to the mixing time

Although for the purpose of analytical tractability, we assume that the non-Sybil region is fast mixing as in previous work [99, 42, 91], SybilRank's effectiveness does not rely on the absolute value of the non-Sybil region's mixing time. Instead, it requires only that the graph G containing both Sybils and non-Sybils has a longer mixing time than the non-Sybil region G_H. Under this condition, the early-terminated power iteration yields a gap between the degree-normalized trust of non-Sybils and Sybils.

Ideally, the number of iterations that SybilRank performs is set as equal to the mixing time of the non-Sybil region. Whether or not the mixing time of the non-Sybil region is $O(\log n)$ or longer, setting the number of iterations to that value produces almost uniform degree-normalized trust among the non-Sybil users. Previous Sybil defenses [99, 42] have assumes that the mixing time of social networks is $O(\log n)$. However, measurement studies [44, 75] have shown that the mixing time of some social networks is longer than expected. Whether this increase derives from a large coefficient in front of the term $\log n$, or whether the mixing time of social networks is not $O(\log n)$, remain unclear.

In SybilRank we simply use $O(\log n)$ power iterations. If the mixing time of the non-Sybil region is larger than this value, the trust that escapes to the Sybil region is further limited. However, there is also a risk of starving the non-Sybil users that are not well-connected to seeds. This risk is mitigated by placing seeds in many communities and by dispersing multiple seeds in each community (§3.4.1), thereby ensuring that the trust is initiated somewhere close to those non-Sybil users.

3.4.2 Ranking by degree-normalized trust

After $w = O(\log n)$ power iterations, SybilRank ranks nodes by their degree-normalized trust. A node v's degree-normalized trust is defined as: $\hat{T}_v = \frac{T^{(w)}(v)}{deg(v)}$. We rank nodes by

degree-normalized trust for two reasons.

Eliminating the node degree bias. This design gives each non-Sybil node almost identical degree-normalized trust. It reduces false positives from low-degree non-Sybil nodes and false negatives from high-degree Sybils. This reduction occurs because after $O(\log n)$ power iterations, the trust distribution in the non-Sybil region approximates the stationary distribution in that region, i.e., the amount of trust on each non-Sybil node is proportional to its degree. Normalizing a node's trust by its degree eliminates the bias that a low-degree non-Sybil node may garner a lower total trust than a high-degree Sybil node does, making a non-Sybil user rank higher than a Sybil with high probability.

The removal of the degree bias simplifies the Sybil/non-Sybil classification, i.e., all non-Sybil users are supposed to belong to the same class with an almost identical degree-normalized trust. This is in contrast to prior trust inference schemes [81, 59, 52] that attempted to differentiate between Sybils and non-Sybils with highly variable trust scores.

Bounding highly-ranked Sybils. Our design ensures that the number of Sybils ranked higher than the non-Sybil nodes is upper bounded by $O(\log n)$ per attack edge. We show this result formally in §3.5, but provide a high-level description here. This result holds because after early termination, the trust distribution in the entire graph G has not reached its stationary distribution. Because we begin the trust distribution from the non-Sybil region, the Sybil region (in aggregate) receives only a fraction f ($f < 1$) of the total amount of trust it should obtain in the stationary distribution. As the total amount of trust is conserved in the entire graph, the non-Sybil region obtains an aggregate amount of trust that is c ($c > 1$) times higher than that of the stationary distribution. Because the non-Sybil region is well connected, each non-Sybil node obtains approximately identical degree-normalized trust, i.e., $c \times \frac{T_G}{2m}$, where $\frac{T_G}{2m}$ is a node's degree-normalized trust in the stationary distribution (§3.4.1).

The amount of degree-normalized trust obtained by each Sybil node depends on how

Sybils connect to each other. However, because the aggregate amount of trust of the Sybil region is bounded, on average each Sybil obtains $f \times \frac{T_G}{2m}$ degree-normalized trust, which is less than a non-Sybil node's degree-normalized trust. We are able to show that at most $O(\log n)$ Sybils per attack edge obtain higher degree-normalized trust than non-Sybil nodes (§3.5).

3.4.3 Annotating the ranked list

SybilRank relies on the limited attack edges to distinguish Sybils. It may give high rankings to the Sybils that obtain many social links to non-Sybil users and consider them as non-Sybil users. This is true for all social-graph-based defenses. Essentially, some Sybils are not ranked at the bottom, but instead are mixed with real users in the middle or at the top of the ranked list. In fact, the top and bottom intervals of the ranked list may comprise a non-negligible portion of fake and real accounts, respectively. Thus, an OSN cannot simply identify a pivot in the ranked list below which all nodes are fake and, instead, it still has to rely on manual inspection.

At the same time, OSNs have limited resources for manual inspection. To aid OSNs in adjusting their inspection focus, we annotate intervals in the ranked list with fake portions. In particular, we sample random users in each interval of a particular size and report a portion of fakes after manual inspection. With these annotations, OSNs can decide where on the ranked list to assign their limited human verifiers. If it is urgent for an OSN provider to discover many fakes with only limited resources, he can investigate the intervals that are closer to the bottom of the list, where a large number of fake accounts are located. If the goal is to also unveil Sybils "closer" to the non-Sybil region, intervals with a lower portion of fakes can be investigated. The annotations can also be used to regulate the frequency of CAPTCHAs and other challenges sent to the users who are under suspicion. Moreover, the annotations can help OSNs to decide on accounts that do not exhibit sufficient evidence about whether they are fake. In our real-world evaluation, those annotations are used to

evaluate SybilRank's performance in Tuenti (§3.8.3).

3.4.4 Computational cost

SybilRank's computational cost is $O(n \log n)$. This is because each power iteration costs $O(n)$, and is iterated $O(\log n)$ times. The cost for ranking the nodes according to their degree-normalized trust is also $O(n \log n)$. The cost for estimating communities is $O(m)$, because each iteration in Louvain method has a computational cost linear to the number of edges and the graph shrinks rapidly with only a few iterations. Because the node degree in OSNs is always limited, the community estimation cost is $O(n)$. Thus the overall computational cost is $O(n \log n)$, irrespective of the number of trust seeds.

3.5 Security Guarantees

In this section, we provide security guarantees under the assumption that the attack edges between non-Sybil nodes and Sybils are randomly established. Although our system does not depend on the absolute mixing time of the non-Sybil region (§3.4.1), the following analysis assumes that the non-Sybil region is fast-mixing.

We first bound the aggregate trust in the Sybil region after $O(\log n)$ iterations, and then find the upper bound of the number of Sybils that are ranked higher than non-Sybil nodes. This bound is independent of the formation of the Sybil region. We provide the intuition and main results in this section and proofs are presented in §A.2.

3.5.1 Dynamics of aggregate trust of Sybils

SybilRank attempts to allocate a comparatively small amount of trust to Sybils, such that they are ranked at the bottom. First, we investigate the expected aggregate trust within the Sybil region after $O(\log n)$ power iterations. Given an initial trust vector, the trust assigned to each individual node after $O(\log n)$ power iterations is determined by the row-normalized adjacency matrix, i.e., the graph topology. We assume that there are g attack edges.

The following terms are used to describe the security guarantees that SybilRank provides. For a node set X, we denote its *volume* as the sum of node degrees within X, i.e., $vol(X) = \sum_{x \in X} deg(x)$. We also denote the *expansion* of X as $C(X) = \frac{|\partial(X)|}{vol(X)}$, where the edge boundary $\partial X = \{(u, v) | u \in X, v \in V \backslash X\}$. A high value of $C(X)$ indicates that the node set X connects well to the rest of the graph. If the outgoing links are randomly distributed in the node set X, a $C(X)$ fraction of trust from X is expected to escape to the rest of the graph in each power iteration. In particular, for the node sets of non-Sybils and Sybils, $|\partial H| = |\partial S| = g$. Therefore, $C(H) = g/vol(H)$ and $C(S) = g/vol(S)$. For simplicity, we denote them as C_H and C_S, and define $T^{(i)}(X)$ as the aggregate trust in the

node set X after the i_{th} iteration.

Lemma 1. *In the $(i+1)_{th}$ iteration, $\forall i \geq 0$, the expected aggregate trust in the Sybil region increases by an amount of $(1 - C_H - C_S)^i \times C_H T_G$.*

This lemma is proved by induction. As each node exchanges trust with its direct neighbors in each iteration, the trust escapes from the non-Sybil region to the Sybil region. Lemma 1 indicates that the expected aggregate Sybil trust increases monotonically. When i is large enough, the increment of the aggregate Sybil trust in each iteration approximates 0. Meanwhile, the aggregate Sybil trust converges to the stationary distribution $\frac{vol(S)}{2m} \times T_G$.

Corollary 2. *In the i_{th} iteration, the expected increment of aggregate trust in the Sybil region is upper bounded by $(1 - C_H)^i \times C_H T_G$.*

Corollary 2 is derived directly from Lemma 1.

3.5.2 Aggregate trust in the Sybil region

We denote t_{G_H} and t_G as the mixing time of G_H and G respectively. As shown in §A.1, $t_G \gg t_{G_H}$. Suppose that in the fast-mixing non-Sybil region $t_{G_H} = w_0 \log n$ (w_0 is a positive integer). We quantify t_G as $t_G \simeq r \log n$, where r is a large integer.

Theorem 3. *Suppose the non-Sybil region G_H is fast-mixing, and the mixing time of the entire graph G is $r \log n$, where r is a large integer. After $w = w_0 \log n$ ($r = k w_0$) power iterations, where w_0 and k are positive integers, Sybils acquire a disproportionally small amount of aggregate trust. This amount is only a small fraction f of that in the stationary distribution, i.e., $T^{(w)}(S) = f \frac{vol(S)}{2m} T_G$, such that*

$$f = \frac{1}{1+(1-\alpha)^{w_0 \log n}+(1-\alpha)^{2w_0 \log n}+\cdots+(1-\alpha)^{(k-1)w_0 \log n}}$$

and $\alpha = C_H + C_S$.

Theorem 3 is derived from a direct comparison of aggregate Sybil trust between the stationary distribution and the trust distribution after w iterations. It shows a skewed trust

distribution over the entire graph and bounds the aggregate trust in the Sybil region after $O(\log n)$ iterations to a fraction f of that in the stationary distribution. Therefore, a trust gap exists between the Sybils and the non-Sybil nodes. The magnitude of their trust difference is determined by the mixing-time difference and the conductance between the non-Sybil region and the Sybil region, both of which depend on the number of attack edges that the Sybils can establish.

3.5.3 Sybil ranking analysis

Because the total trust is conserved within the graph during the power iterations, Theorem 3 further implies that the non-Sybil nodes are given aggregate trust that is c ($c > 1$) times of that in the stationary distribution, where $c = 1 + (1 - f)\frac{vol(S)}{vol(H)}$. If the non-Sybil region is fast-mixing, we postulate that the degree-normalized trust distribution after $O(\log n)$ iterations within the non-Sybil region is close to uniform, as $O(\log n)$ iterations approximate the stationary distribution of the non-Sybil region. In particular, the non-Sybil nodes have almost the same degree-normalized trust $c \times \frac{T_G}{2m}$. This hypothesis has been empirically validated by our experiments (§ 3.7.3).

However, depending on the formation of the Sybil region, w iterations may not result in identical degree-normalized trust in the Sybil region, Therefore, given the limited aggregate Sybil trust, SybilRank may accept a different number of Sybils. We draw two extreme cases of the trust distribution in the Sybil region that include derivations of a lower bound and an upper bound on the number of Sybils that are ranked higher than non-Sybil nodes.

The lower bound comes from the first extreme, where the Sybil region is assumed to be well connected and fast-mixing and each Sybil has the same degree-normalized trust, i.e., $f \times \frac{T_G}{2m}$. In this case, although each Sybil has a particular amount of trust, none of them is ranked higher than non-Sybil users because of $f < 1 < c$.

At the second extreme, we assume that the Sybil users could arbitrarily determine the trust distribution with the limited aggregate Sybil trust and thereby are able to get

the maximum number of Sybils accepted. Because an amount of degree-normalized trust $c \times \frac{T_G}{2m}$ is the minimum trust that can make a Sybil rank as high as a non-Sybil user, the optimal strategy is to assign $c \times \frac{T_G}{2m}$ degree-normalized trust to as many Sybils as possible, and leave the rest Sybils with zero trust. We use contradiction to prove that at most $\frac{f}{c} \times vol(S)$ Sybils would have degree-normalized trust higher than $c \times \frac{T_G}{2m}$. Consider the opposite scenario, where more than $\frac{f}{c} \times vol(S)$ Sybils have degree-normalized trust higher than $c \times \frac{T_G}{2m}$. Because each of these Sybils has a degree of at least 1 (otherwise they would be disconnected and unable to garner any trust), the aggregate trust of these Sybils is more than $\frac{f}{c} \times vol(S) \times c \times \frac{T_G}{2m}$, i.e., $f\frac{vol(S)}{2m}T_G$. This scenario contradicts Theorem 3.

Moreover, the attackers cannot in fact fabricate the second extreme—a statement that can also be proved by contradiction. In the second extreme, we could split Sybils into two disjoint groups: a *boosting group* consisting of Sybils that consume all aggregate Sybil trust and a *silent group* formed by Sybils without any trust. This split would imply that the boosting group and the silent group are disconnected, and that all attack edges are connected with the boosting group. Otherwise, during propagation the trust would either flow from the boosting group to the silent group or from the non-Sybil region to the silent group. In both cases, the Sybils in the silent group would obtain a non-zero amount of trust. However, such disconnection in the Sybil region actually reduces the effective size of the Sybil region to the number of Sybils in the boosting group, i.e., the silent group can simply be removed. The reduction of the Sybil region would decrease $vol(S)$, and hence increase C_S due to $C(S) = g/vol(S)$. As indicated by Lemma 1, the increment of aggregate Sybil trust in each iteration, i.e., $(1 - C_H - C_S)^i \times C_H T_G$, would decline accordingly. Therefore, the sum of the increments of aggregate Sybil trust over $O(\log n)$ iterations would be less than $T^{(w)}(S)$. This is a contradiction.

3.5.4 Upper bound

We provide further analysis on the second extreme where at most $\frac{f}{c} \times vol(S)$ Sybils ranked higher than non-Sybil nodes. Theorem 4 gives an upper bound on $\frac{f}{c} \times vol(S)$.

Theorem 4. *When an attacker randomly establishes g attack edges in a fast-mixing social network, the total number of Sybils that rank higher than non-Sybils is $O(g \log n)$.*

Theorem 4 can be proved by providing an upper bound on $\frac{f}{c} \times vol(S)$.

3.6 Implementation

OSNs (e.g., Facebook) grow rapidly and already have hundreds of millions of users. A parallelized Sybil defense is desirable for OSN providers because it can be easily deployed in data warehouses and on existing OSN systems. We now describe how we implemented SybilRank using the MapReduce [43] parallel computing framework. It can also be implemented on recent data processing platforms such as Giraph [19] and Spark [102]. Our MapReduce implementation enables an OSN provider to process social network graphs with hundreds of millions of users on a cluster of commodity machines. A graph at this scale does not fit in the memory of a typical commodity machine. Even if it does, finishing SybilRank's computation on a single machine would be very time-consuming.

We divide the entire graph into multiple partitions so that each one fits into the hardware of a commodity machine. We observe that the complexity of SybilRank is dominated by the first two stages (Figure 3.3): trust propagation via $O(\log n)$ power iterations and node ranking based on degree-normalized trust. Together they have $O(n \log n)$ complexity.

3.6.1 Trust propagation via power iterations

In trust propagation, trust splitting and trust aggregation at each iteration are inherently parallelizable. Therefore, we treat each iteration as a MapReduce job and create multiple map tasks to split trust and multiple reduce tasks to aggregate trust in parallel.

For each graph chunk that contains a subset of nodes, a map task splits trust on each node. It emits intermediate output in a form of $\langle \text{neighbor ID}, \text{trust} \rangle$. Reduce tasks aggregate the intermediate tuples targeting to the same node and eventually emit $\langle \text{node ID}, \text{trust} \rangle$ to update the trust on each node. In the last iteration, we divide the trust of each node by its degree to generate degree-normalized trust.

To perform consecutive power iterations, during the reduce phase we need to embed

the updated node trust in the graph structure, so that the map phase in next iteration can be aware of the trust reallocation. A naive implementation would shuffle the graph structure between map and reduce tasks. Instead, we bypass the shuffling of the graph structure using a design pattern, called *Schimmy* [69]. The basic idea is to draw a mapping between graph chunks and reduce tasks in the partitioner, such that a reduce task can read a chunk of graph structure directly from disk and update node trust accordingly.

3.6.2 Ranking based on degree-normalized trust

The ranking stage is also a bottleneck because the a sorting algorithm has a complexity of $O(n \log n)$ on a single machine. MapReduce is an efficient distributed sorting framework because it shuffles and sorts intermediate tuples by keys before shipping them to reduce tasks. We use a MapReduce job to rank users based on the degree-normalized trust. We switch the roles of elements in the tuples generated by trust propagation, i.e., the degree-normalized trust becomes the key and the node ID becomes the value. A reduce task then collects all tuples and yields a sorted sequence of nodes in an non-decreasing order of the degree-normalized trust. Because the sorting of a subset of tuples is completed in parallel and the merging sorted subsets of tuples incurs a linear cost, the total cost of the ranking stage can be greatly reduced.

To further reduce the intermediate results that need to be shuffled between map tasks and reducer tasks, we incorporate other optimization techniques for MapReduce jobs, including in-mapper combining and customized partitioning [69].

3.6.3 Efficiency

We evaluate the efficiency of our prototype on an Amazon EC2 cluster, which processes large-scale synthetic graphs with hundreds of millions of nodes. The cluster consists of 11 m1.large instances, one of which serves as the master and the other 10 as slaves.

We generate very large synthetic graphs based on the scale-free model, as in §3.7.1.

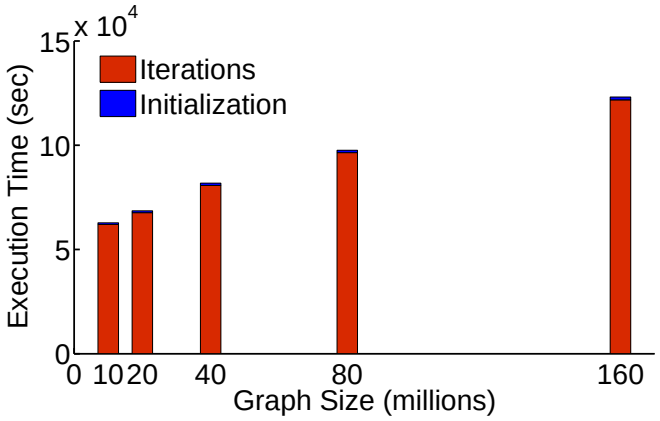


FIGURE 3.6**: Total execution time as a function of graph size in an Amazon EC2 cluster.**

The synthetic graphs are generated with exponentially increasing sizes from 10M to 160M nodes, i.e., 10M, 20M, ...,160M. SybilRank performs successfully on each graph with $\log n$ power iterations.

Figure 3.6 shows SybilRank's execution time on each graph. The total execution time includes two parts: the time it takes to seed trust and to partition the graph during initialization, and the time to execute power iterations to propagate trust and to rank users. As can be seen in Figure 3.6, the latter dominates the total execution time, which increases almost linearly with the size of social graphs. For the largest graph (160M nodes), our implementation finishes in less than 33 hours. This result suggests that a SybilRank implementation can process large-scale social graphs using a few commodity machines.

3.7 Evaluation

In this section, we present a comparative evaluation of SybilRank's ability to provide a meaningful ranking of nodes to uncover Sybils. We first compare SybilRank against other approaches in terms of its effectiveness in assigning low ranking to Sybils. We then examine the SybilRank's component that copes with the multi-community structure, and we study the resilience of our approach to attacks that target the seeds. For a fair comparison, we do not use SybilRank's component for coping with the multi-community structure except for §3.7.4.

3.7.1 Simulation setup

Compared approaches. We compare SybilRank (SR) to the state-of-the-art social-graph-based Sybil defenses: SybilLimit (SL) [99], SybilInfer (SI) [42], Mislove's community detection [72] (CD), and GateKeeper (GK) [91]. Importantly, we also compare to EigenTrust (ET) [59], which uses power iteration to assign trust.

Datasets. The non-Sybil regions of the simulated social graphs, which exclusively comprise non-Sybils, are samples from several popular social networks (Table 3.1). The Facebook graph [49] is a connected component sampled via the "forest fire" method [66]. The synthetic graph is generated using Barabasi's scale-free model [32]. The rest of the graphs [16] have been widely used to study social-graph properties and to evaluate recent Sybil defense mechanisms [93, 75].

Attack strategies. We create a $5K$-node Sybil region that connects to a non-Sybil region through a varying number of random attack edges. We choose this large number of Sybils to stress-test each scheme. To investigate the schemes' robustness to the formation of the Sybil collective, we include two representative Sybil region structures: regular random graphs and scale-free graphs [32]. We call the first attack *regular attack*: each Sybil establishes connections to d random Sybils. We refer to the second attack as a *scale-free*

attack: each Sybil preferentially connects to d Sybils upon its arrival, with the probability of connecting to a Sybil proportional to the Sybil's degree. We set $d = 4$ in both attacks.

Table 3.1: Social graphs used in our experiments. The last three graphs are 10K-node BFS samples.

Social Network	Nodes	Edges	Clustering Coefficient	Diameter
Facebook	$10,000$	$40,013$	0.2332	17
ca-AstroPh	$18,772$	$198,080$	0.3158	14
ca-HepTh	$9,877$	$25,985$	0.2734	18
Synthetic	$10,000$	$39,399$	0.0018	7
wiki-Vote	$7,115$	$100,736$	0.1250	7
soc-Epinions	$10,000$	$222,077$	0.0946	6
soc-Slashdot	$10,000$	$153,404$	0.0582	4
email-Enron	$10,000$	$105,343$	0.1159	6

Performance metrics. Our evaluation is based on a framework [93] that reduces defense schemes to a general model to produce a trust-based node ranking. The conversion for SybilLimit, SybilInfer, and CD have been documented in [93]. For GateKeeper, we rank nodes by the number of tickets that each node obtains. For EigenTrust, we rank nodes according to their trust scores.

We use three metrics to compare the node ranking: the area under the Receiver Operating Characteristic (ROC) curve [53], the false positive rates, and the false negative rates. The ROC curve exhibits the change of the true positive rate with the false positive rate as a pivot point moves along the ranked list: a node below the pivot point in the ranked list is determined to be a Sybil; if the node is actually a non-Sybil, we have a false positive. The area under the ROC curve measures the overall quality of the ranking, i.e., the probability that a random non-Sybil node is ranked higher than a random Sybil. It ranges from 0 to 1, with 0.5 indicating a random ranking. An effective Sybil detection scheme should achieve a value larger than 0.5. Given a node ranked list, sliding the pivot point regulates the trade-off between the two false rates. We set the pivot point based on a fixed value for one false rate and compute the other false rate. We set the fixed false rate equal to 20%.

In the real world, OSNs do not need a pivot point because none of the defenses so far can yield a binary Sybil/non-Sybil classifier with an acceptable false positive rate.

Trust seed selection. For a fair comparison, we strive to use the same trust seeds for all schemes in each simulation on each social network. For schemes that use a single seed, we randomly pick a node from the top-10 non-Sybil nodes that have the highest degree. For schemes supporting multiple seeds at one run, i.e., SybilRank, EigenTrust, and GateKeeper, we use 50 trust seeds. One seed is the same top-10 degree node as in the single-seed schemes, and the other seeds are randomly chosen from the non-Sybil nodes.

Other simulation settings. We perform $\log n$ power iterations for SybilRank, where n is the size of the social graph. We run EigenTrust until convergence with a reset probability 0.15, as in [59]. For each attack scenario, we average the results over 100 runs.

3.7.2 Ranking quality comparison

To compare the Sybil defense schemes, we start with a few attack edges and increase the number to a sufficiently large value such that the detection accuracy of each scheme degrades significantly. We show representative simulation results in Figure 3.7 and refer the reader to our technical report for the complete results [37]. As can be seen, when the number of attack edges is small, most of the schemes perform well and Sybils can be distinguished from non-Sybils by connectivity.

SybilRank. SybilRank outperforms all other schemes. It achieves the highest value of the area under the ROC curve and the lowest false positive and false negative rates. For the Facebook graph under the regular attack, as indicated by the value of the area under the ROC curve, even when the $5K$-node Sybil cluster obtains 1500 attack edges, a non-Sybil node has a probability of 70% of ranking higher than a random Sybil.

SybilLimit. SybilLimit outperforms most other schemes, but it performs worse than SybilRank. We believe that this is due to the different use of random walks, i.e., Sybil-

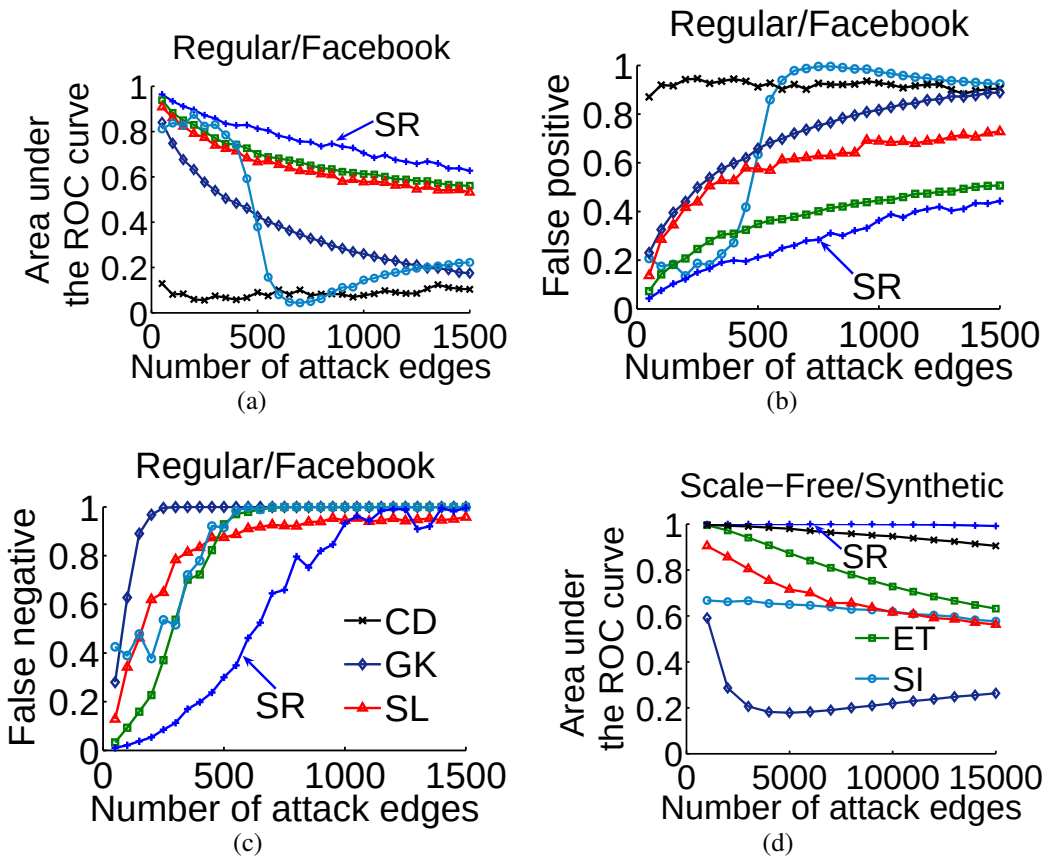


FIGURE 3.7**: We depict the area under the ROC curve, the false positive rate, and the false negative rate with respect to the number of attack edges under the regular attack and the scale-free attack for various Sybil detection schemes in the Facebook and the scale-free synthetic graphs. In the ROC curve figures (a and d), a higher curve indicates a more effective scheme. In the false rate figures (b and c), a lower curve indicates a more effective scheme. SR stands for SybilRank, CD for the community detection algorithm, GK for GateKeeper, SI for SybilInfer, ET for EigenTrust, and SL for SybilLimit.**

Limit uses random walk traces, whereas SybilRank uses the power-iteration-computed landing probability. SybilLimit's security guarantee only limits the Sybils accepted by the verifier's (trust seed's) random walks that never cross any attack edge to the Sybil region [98]. However, the accepted Sybils cannot be bounded if a verifier's random walk enters into the Sybil region. In contrast, SybilRank allows trust to escape to the Sybil region, but does not accept a Sybil unless it obtains higher degree-normalized trust than

non-Sybil users.

GateKeeper. It performs worse than both SybilRank and SybilLimit, although it bounds the accepted Sybils to $O(\log g)$ per attack edge, where g is the total number of attack edges. This is because this bound comes from a strong assumption that does not always hold in real social networks: with high probability, a breadth-first search starting from a non-Sybil user and visiting at most $n/2$ nodes covers a large fraction of non-Sybil users [98].

SybilInfer. We observe a steep fall in the area under the ROC curve for SybilInfer when the number of attack edges is close to 500 in Facebook under the regular attack. We suspect that this sharp performance degradation is because SybilInfer uses the Metropolis-Hastings (MH) algorithm to sample the non-Sybil node set [42]. However, it remains unclear when the sampling converges, although Danezis et al. provide an empirical estimation and terminate the sampling after $O(n \log n)$ steps. If the Sybils obtain many attack edges and become difficult to detect, $O(n \log n)$ steps may not suffice to reach the convergence of the MH sampling and therefore, the detection accuracy is likely to degrade.

Mislove's CD. It underperforms under the regular attack, as the area under the ROC curve area is below 0.2 in most cases (Figure 3.7). Interestingly, it becomes effective under the scale-free attack in the Synthetic graph (Figure 3.7(d)). This significant performance difference is due to the greedy search for the local community, which is sensitive to the graph topology and cannot provide a false-rate bound [98].

Although Mislove's CD algorithm was intended to be used with one seed, it can support multiple seeds at the same cost: by initializing the local community search with those seeds. In §3.7.4 we show that this extension can improve its performance to some extent.

EigenTrust. EigenTrust improves over PageRank [81], and uses the same basic mechanism as TrustRank [52]. Figure 3.7 shows that EigenTrust mostly outperforms previously proposed Sybil defenses. Nonetheless, it has at least 20% higher false positive and negative rates than SybilRank in most of the attack scenarios.

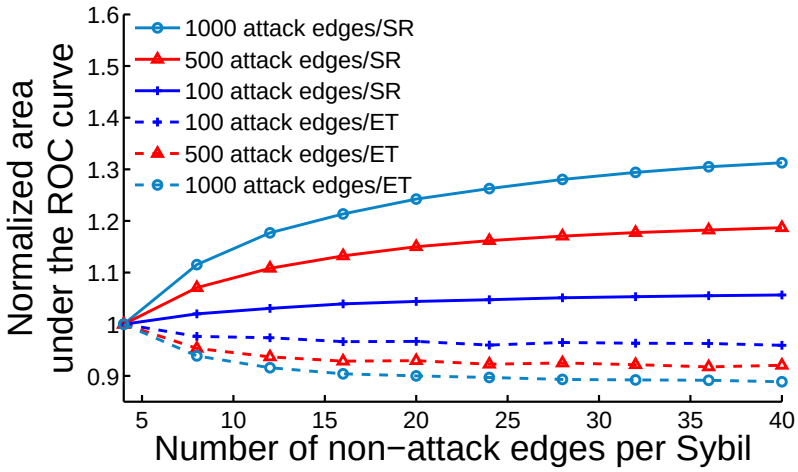


FIGURE 3.8**: Normalized area under the ROC curve as a function of the number of edges connecting to other Sybils.**

3.7.3 Comparison with EigenTrust

EigenTrust is related to SybilRank because it also uses power iteration. However, it exhibits bias towards nodes that have a high degree or are "closer" to seeds. By further investigating EigenTrust we reveal the importance of SybilRank's two main differentiating characteristics: a) removal of degree bias (§3.4.2); and b) early termination and not jumping back to trust seeds (§3.4.1).

Impact of the connectivity of the Sybil region. We examine how the connection density within the Sybil region impacts the node ranking generated by EigenTrust and SybilRank. To do so, we vary the number of edges each Sybil has with other Sybils from 4 to 40 under a regular attack on the Facebook graph. We refer to such edges as *non-attack*.

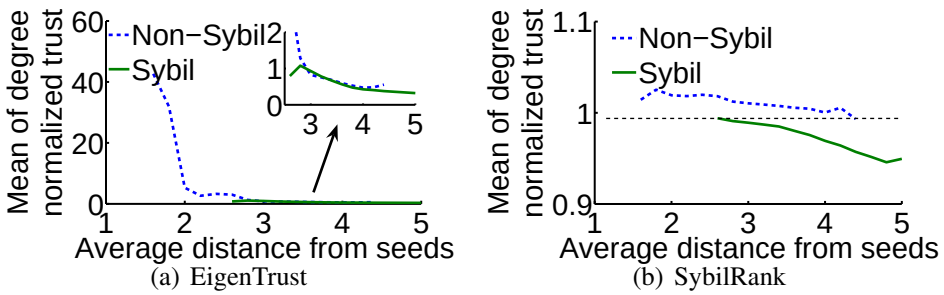
Figure 3.8 shows the *normalized area under the ROC curve*, which is computed by dividing the area under the ROC curve for each attack scenario with the baseline case where each Sybil has only 4 non-attack edges. As can be seen in Figure 3.8, with EigenTrust, the value of the area under the ROC curve decreases when the connections within the Sybil region become dense (the normalized area under the ROC curve is always less than 1).

This result is because in EigenTrust a node that has many incoming links or is pointed to by other highly-ranked nodes is typically ranked high [81, 59]. A malicious user can simply create dense connections within the Sybil region to boost the ranks of selected Sybils. Therefore, denser Sybil connections lower EigenTrust's values of the area under the ROC curve, indicating that more Sybils are ranked higher than non-Sybils.

On the contrary with SybilRank, due to the removal of degree bias, high-degree Sybils do not benefit. Instead, SybilRank's performance improves as the connections among Sybils become denser. This is because it exploits the sparse cut between the Sybil and the non-Sybil region, which the addition of Sybil connections makes even sparser (it lowers the conductance).

Impact of the distance from trust seeds. EigenTrust's trust distribution is sensitive to the locations of the seeds because its random walks jump back to them. Figure 3.9 compares the distribution of *degree-normalized* trust generated by EigenTrust and SybilRank with respect to a node's average shortest hop distance from the trust seeds. We simulate a regular attack in the synthetic graph with $10K$ attack edges, and select 5 seeds for both EigenTrust and SybilRank. The total trust is set to $2m$. As shown in Figure 3.9(a), EigenTrust tends to allocate high degree-normalized trust to nodes close to the seeds, whereas nodes that are relatively farther from the seeds receive substantially lower trust. In fact, the degree-normalized trust distribution in EigenTrust has a "heavy" tail. Among the $10K$ non-Sybil nodes, more than $9.4K$ have degree-normalized trust less than 2. As we zoom into the tail, this large set of non-Sybil users and Sybils show indistinguishable degree-normalized trust .

Figure 3.9(b) shows that unlike EigenTrust, SybilRank assigns roughly the same degree-normalized trust to each non-Sybil node while keeping a distinguishable gap between non-Sybils and Sybils. This result empirically validates our observation that the degree-normalized trust is approximately uniformly distributed in the non-Sybil region after $O(\log n)$



(a) EigenTrust (b) SybilRank

FIGURE 3.9**: Trust distribution with respect to the average distance to seeds in the synthetic graph under the regular attack with** 10000 **attack edges.**

power iterations (§3.5.3).

3.7.4 Coping with multiple communities

As discussed in §3.4.1, we distribute seeds among communities to cope with the multi-community structure in the non-Sybil region. We illustrate this with simulations on synthetic graphs. The simulations include Mislove's CD and EigenTrust, which can be initialized with multiple seeds at no additional computational cost. We do not include GateKeeper because it uses random walks to seek seeds (ticket sources), thus we do not fully control the placement of seeds among the communities.

Similar to the simulation scenarios in [93], we set up a non-Sybil region consisting of 5 scale-free synthetic communities, each of which has 2000 nodes with an average degree of 10. We designate one community as the core of the social graph. The other 4 communities do not have any connections to each other, but connect to the graph core via only 500 edges. This process builds a non-Sybil region where multiple communities connect to the core community of the graph via limited links.

We contrast the following two seed selection strategies: a) all 50 seeds are confined to the core community; and b) the seeds are distributed among all the non-Sybil communities, each of which has 10 seeds. As shown in Figure 3.10, seed selection strategy (b) improves the detection accuracy for all schemes. Because wide seed coverage in the non-

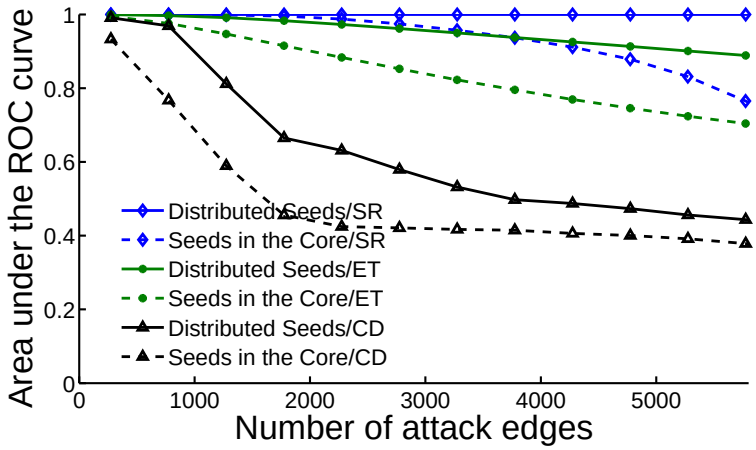


FIGURE 3.10**: Comparison of SybilRank, EigenTrust, and Mislove's CD in a multi-community graph.**

Sybil region offsets the impact of its multi-community structure, the resulting ranking is mostly determined by the sparse cut between the non-Sybil region and the Sybil region. In Figure 3.10, we see that SybilRank maintains the highest accuracy for each of the seed placement strategies due to the disadvantages of EigenTrust and Mislove's CD mentioned in §3.7.3 and §3.7.2.

3.7.5 Resilience to seed-targeting attacks

Last, we study how various schemes perform under targeted attacks. In such attacks, sophisticated attackers may obtain partial or full knowledge about the OSN and thereby discover the locations of the trust seeds. They can then establish attack edges to nodes close to those seeds.

We simulate a targeted attack in the Facebook graph. The 5K-node Sybil region forms a regular attack structure and has 200 attack edges connecting to the non-Sybil region. Instead of being completely randomly distributed, those 200 attack edges are established by randomly connecting to the k non-Sybil nodes with the shortest distances from the trust seed. For schemes that support multiple seeds, we target the attack edges to the highest-

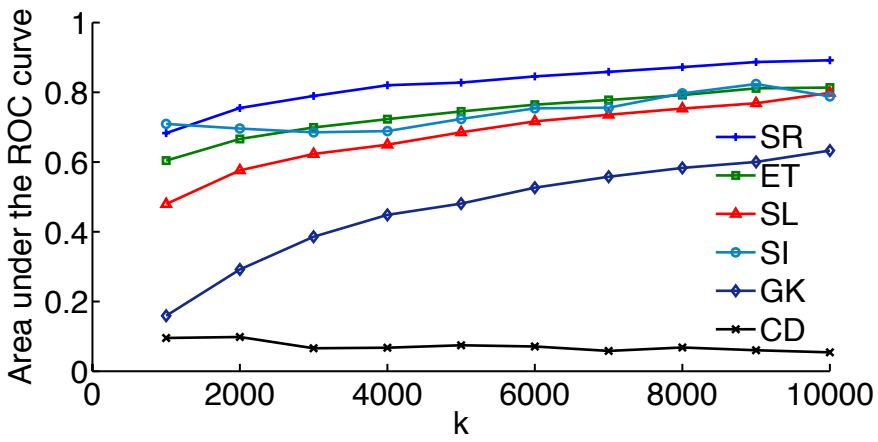


FIGURE 3.11: **Detection effectiveness under attacks targeting the k non-Sybils with the shortest distance to the seed.**

degree trust seed. We vary k from 1K to 10K. A smaller k represents a shorter average distance between Sybils and seeds.

As shown in Figure 3.11, when attack edges are attached close to the seed, all schemes' performance degrades. SybilRank keeps the most stable performance across a wide range of k values. This is because SybilRank does not assign excessive trust to nodes that are close to the seeds (§3.7.3). However, SybilRank's performance still degrades when k is small. This is because these closely targeted attacks force SybilRank to "leak" a fraction of trust in the Sybil region during early power iterations. Thus, its detection accuracy declines as some Sybils have gained higher trust than non-Sybils. In addition, Sybilinfer's accuracy improves a little when k is small. This is because it uses the MH sampling, which may be not sensitive to targeting attacks when there are not many attack edges.

3.8 Real-world Deployment

We now discuss the deployment of our system on a snapshot of Tuenti's complete social-friendship graph, which was obtained in August 2011. Due to the sheer volume of users, it would have been infeasible for us to undertake manual inspection in order to determine whether each user is a Sybil. Therefore, we are unable to evaluate SybilRank with the same metrics as in the simulations (§3.7), such as the area under the ROC curve, the false negative rate and the false positive rate. Instead, we attempt to determine the portion of fake users at varying segments of the ranked list. We do so by manually inspecting a user sample in each particular interval in the ranked list (§3.4.3). Due to the practical constraints and the limited availability of human verifiers, we do not deploy the other Sybil defenses on Tuenti.

3.8.1 Pre-processing

Pruning edges on extremely high-degree nodes. We observed that some fake Tuenti accounts are well-maintained and have extremely high degree. They may introduce many attack edges if they connect to real users.

To limit the impact of those well-maintained fake accounts, we set a degree threshold, i.e., 800, and randomly prune excessive edges on the nodes that have a degree above the threshold, so that the fake social hubs would not gain additional attack edges. We set the degree threshold to 800 because it is a reasonable upper bound on the number of friends that a real user can manage in an OSN. Moreover, removing extra edges on real users would not harm the non-Sybil region, because there are still sufficient social connections to keep the non-Sybil region well connected.

Deferring the consideration of new users. New users who join for a short time typically have only a few social connections in OSNs, due to various constraints. Therefore, the social connections on those recent users are incomplete. To this end, we defer the consid-

eration of new accounts until six months after their creation. Half a year is a reasonable time period for a normal user to fully map his real-world social relationships to an OSN. To obtain a more precise age threshold, OSNs may conduct a study on their own users and find the account age below which most of the users finish establishing a significant number of social connections. Before an account hits this age threshold, we suggest that OSNs enforce strict rate-limiting policies such as IP-based CAPTCHAs, in case it is manipulated by a malicious user.

Communities in Tuenti. The complete Tuenti social graph has 1,421,367,504 edges and 11,291,486 nodes, among which 11,216,357 nodes form a Giant Connected Component (GCC). Our analysis focuses on the GCC. With the Louvain method, we found 595 communities, among which 25 large communities contain more than 100K nodes. We inspected 4 nodes in each community and designated as SybilRank trust seeds the nodes that have passed this manual verification.

3.8.2 Validating the mixing-time assumptions

As discussed in §3.2.2, SybilRank relies on the mixing time gap between the non-Sybil region and the entire graph. Because we seed trust in each large community, the trust propagation is mainly determined by those communities. To this end, we investigate the 25 large communities identified by the Louvain method. As shown in §3.8.3, fake accounts are embedded in each community. We then measure the mixing time gap between each community and its non-Sybil part.

We measure the mixing time using its definition, i.e., the maximum necessary walk length to achieve a given variation distance [73] from the stationary distribution. Due to Tuenti's large population, it is difficult to remove all Sybils in order to measure the mixing time of the non-Sybil part in each community. Therefore, our measurement approximates the mixing time gap by contrasting the mean length of random walks from random users and from the Sybils that are captured by SybilRank in each community.

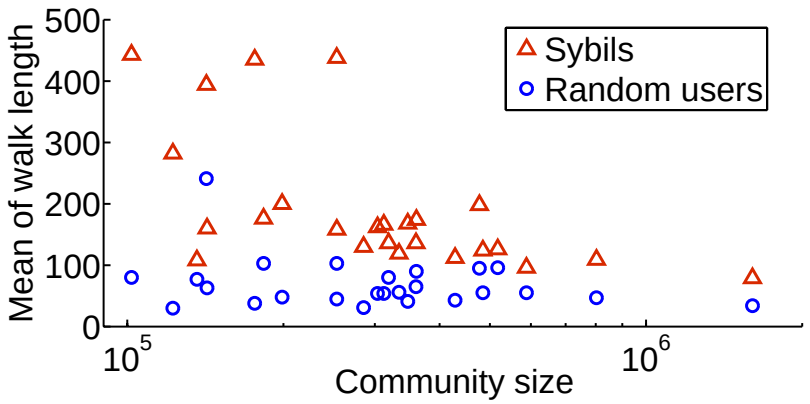


FIGURE 3.12: **Contrasting the mean length of random walks from Sybils and from random users in each community.**

We hypothesize that due to the majority of users in Tuenti being non-Sybil, if the Sybils are weakly connected to the majority, given a total variation distance, it is necessary for the random walks from Sybils to be longer than those from average users. In such a case, the Sybils' random walk length can approximate the mixing time of the entire community, and we designate the length of random walks from average users as the mixing time of the non-Sybil part. Because the average users may include hidden Sybils, using their walk length only overestimates the mixing time of the non-Sybil part.

We select 1000 random users and 100 confirmed Sybils in each community. The total variation distance is set equal to 0.01. As shown in Figure 3.12, the mean length of random walks from the random users in each community is small (mostly less than 100), while the mean Sybil walk length is much longer. This difference indicates that the Sybils connect to the majority of users with a limited number of edges, which makes their needed walk length longer. This result demonstrates that social-graph-based defenses can be effective for our real OSN graph. In addition, recall that in SybilRank we have placed multiple random seeds in each community, which facilitates the convergence of the trust propagation. The number of power iterations that SybilRank needs is even smaller than the random

walk length in Figure 3.12.

3.8.3 Detecting fakes in Tuenti

Manually inspecting the ranked list. We ran SybilRank on the complete Tuenti social graph. We inspected 2K users at the bottom of the resulting ranked list, and all of them were fake (Sybils). We further examined the ranked list by inspecting the first lowest-ranked one million users. We randomly selected 100 users out of each 50K-user interval for inspection. As shown in Figure 3.13, the 100% fake portion was maintained at the first 50K-user interval, but the fake portion gradually decreased as we went up the list. Up to the first 200K lowest-ranked users, around 90% are fake, as opposed to the $\sim$5% hit rate of Tuenti's current abuse-report-based method. These results suggest an 18-fold increase in the efficiency with which Tuenti can process suspected accounts.

We also observe that above the 200K lowest-ranked users, the portion of fakes decreases abruptly from $\sim$90% to $\sim$50%, and then $\sim$10%. This is because fake accounts that have established many social links to real users could be ranked high. This reveals that SybilRank's limitation lies in the open nature of OSNs, i.e., the ease at which some fake accounts can befriend real users.

Although we have sampled users for inspection until the lowest 1 million users on the ranked list, due to confidentiality reasons we report the exact portions of fake users until the lowest 650K users. Above the lowest 650K, we obtain even lower portions of fakes, which subsequently stabilize. One can sample above the point at which the portion of fakes stabilizes to infer the portion of fakes in the complete Tuenti network. We have also obtained a more accurate estimate of the portion of fakes in the network by uniformly sampling over the complete network. This allows us to determine how many fake accounts are not captured by SybilRank. Again, we cannot reveal this statistic, as we are bound by confidentiality.

Precision. To further quantify SybilRank's detection accuracy in Tuenti, we adopt a

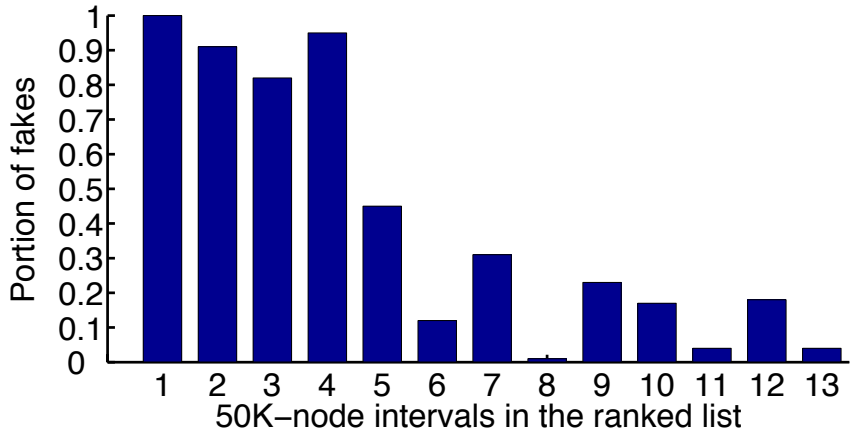


FIGURE 3.13**: Sybil distribution over the lowest 650K users on the ranked list (intervals are numbered from the bottom).**

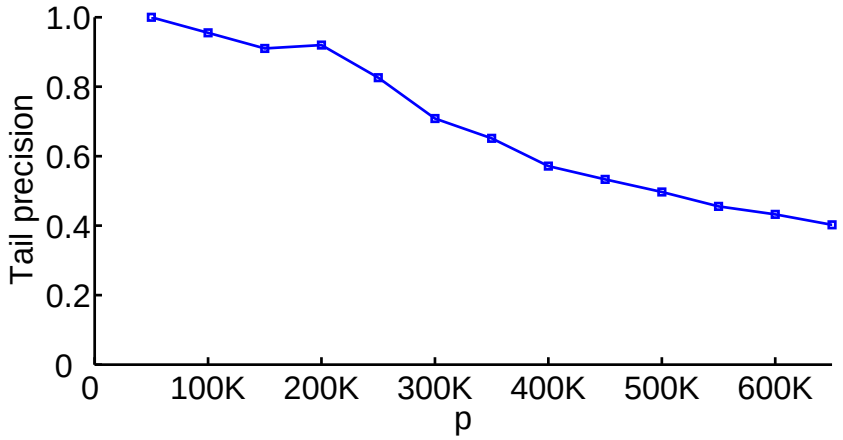


FIGURE 3.14**: Tail precision up to the lowest 650K users.**

standard metric from collaborative filtering called *precision* [55], which is the ratio of relevant items to all selected. We define the *tail precision* at position p as the portion of Sybils among the p lowest-ranked nodes. Figure 3.14 shows the tail precision through the first 650K lowest-ranked users. Among the first 200K lowest-ranked users, the tail precision remains as high as 90%.

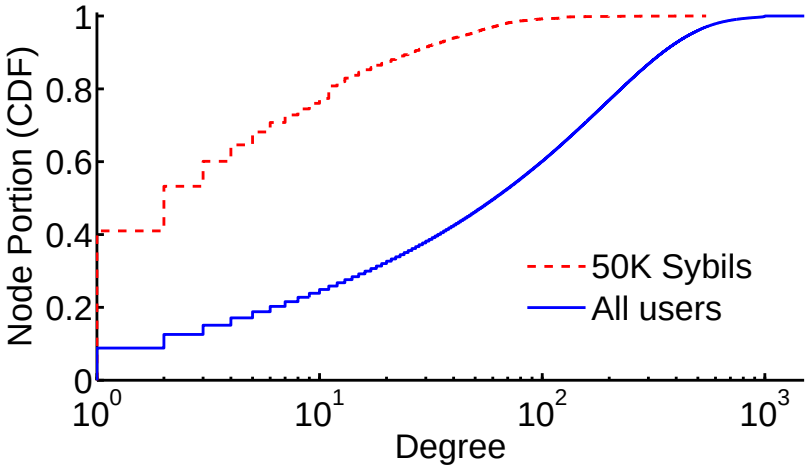


FIGURE 3.15**: Node degree of the first 50K lowest-ranked users and all users.**

Sybil degree distribution. As above, SybilRank has achieved an almost 100% detection of fakes in the first 50K low-ranked nodes. Figure 3.15 shows the degree CDF of those 50K users. On average, Sybils have fewer friends than all users. Although a large proportion of Sybils have relatively low degree, i.e., less than 10, we have captured Sybils with more than 100 friends. Due to the large variance of Sybil degrees, the node degree is not a reliable metric with which to detect fake accounts.

Formation of the Sybil collective. We study the social connections among the first 50K lowest-ranked users. As shown in Figure 3.16, the fake accounts exhibit various formations, including the three large connected components in the middle. These three components manifest a simple tree-like connection pattern, with each node exhibiting similar degree. This indicates that those accounts may be automatically crafted at scale. Furthermore, we found that fakes detected by SybilRank are rarely isolated from each other and that real-world Sybils exhibit clustering, such as star, full-mesh topology, etc. For example, the star collective in Figure 3.16 has two hubs of high degree, but few of its other

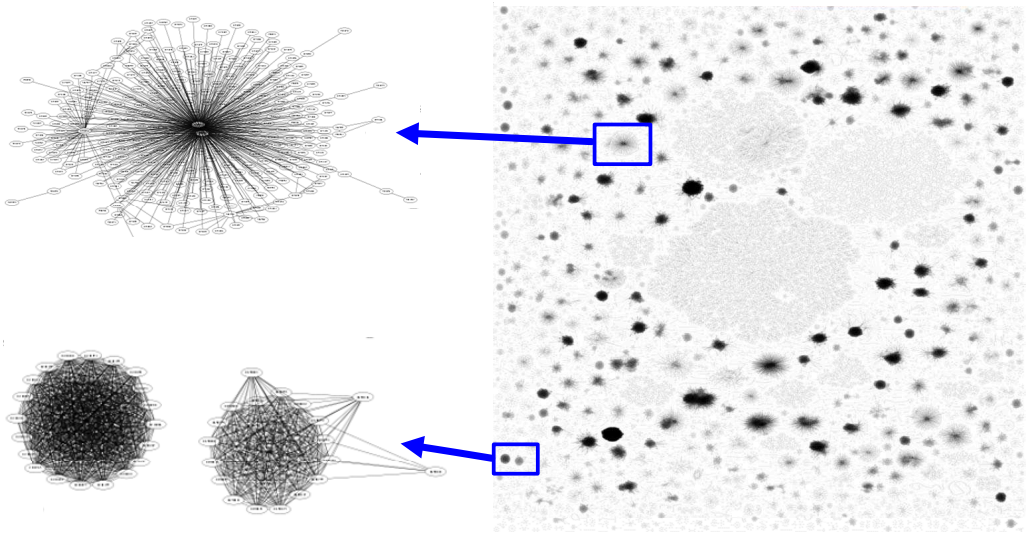


FIGURE 3.16: **Formation of the first 50K lowest-ranked users in Tuenti. The identified accounts exhibit various connection patterns.**

nodes connect to each other. Those 50K users do not form a single connected component. Presumably this is because the attackers behind those fake accounts are not centrally coordinated.

Sybils in large communities. SybilRank leverages the community structure to properly seed trust. This approach is much more accurate than merely determining if an entire community is fake. Some of the $50K$ lowest-ranked users are embedded in the 25 large communities (Table 3.2). For example, each of the top 10 largest communities has hundreds of Sybils. This indicates that SybilRank can detect fake accounts even in large communities that mostly consist of non-Sybil users.

3.8.4 Discussion

With SybilRank, Tuenti needs ∼570 man hours to review the 200K lowest-ranked users and discover ∼180K fake accounts. With its current abuse-report-based method, which has only a ∼5% hit rate, and assuming all these fakes are reported by other users, Tuenti would need ∼10,300 hours. By executing SybilRank periodically, e.g., every month, we

Table 3.2: The lowest-ranked 50K-user distribution in the 25 largest communities with >100K nodes.

Community size	Sybils	Community size	Sybils
1,604,186	416	303,755	279
802,744	588	285,503	392
588,066	738	253,488	1584
517,818	498	253,393	176
484,719	587	198,824	274
477,408	1,142	183,090	287
428,457	165	176,080	745
360,847	571	142,473	186
360,146	441	142,083	145
347,280	630	136,181	386
333,999	140	122,445	247
318,976	278	101,876	327
312,324	505		

expect that Tuenti could be able to to efficiently identify a substantial number of fake accounts over time.

Our study pin-pointed 200K suspicious accounts after a total of 20 hours of SybilRank and community estimation processing. Those accounts can be verified by one full-time (8h) employee in $\sim$70 days. We found that Sybils in Tuenti do form dense connections among themselves, which as we show in §3.7.3 further enables SybilRank to uncover Sybils.

We note that a significant portion of the detected accounts were spammers [12]. This suggests that spammers maintain many accounts that are not directly connected to real users. These accounts cannot spam real user walls or newsfeeds. We postulate that spammers create these accounts to spam via connection requests or in the hope that they will establish real connections at a later time.

SybilRank's guarantee relies on the attack edges that Sybils can obtain. If SybilRank is deployed in OSNs where the social connections are considered to be more valuable such as LinkedIn, or where good user interfaces are developed to induce strong social ties such as Google+, we expect it would detect fake accounts with higher accuracy.

3.9 Summary

Large-scale social online services devote immense attention to the experience of their user base, and the marketability of their user profiles and the social graph. Thus they face significant challenges from the existence and continuous creation of fake user accounts, which dilute the advertising value of their network and annoy legitimate users. For these reasons, we have proposed SybilRank, an effective and efficient fake account-inference scheme, which allows OSNs to rank accounts according to their perceived likelihood of being fake.

The suspension of fake accounts typically requires manual profile inspection because false positives have a detrimental impact on the user experience. Using analysis, simulations, and a live deployment on the 11-million-user Tuenti social network, we showed that SybilRank is effective in designating users that are very likely to be Sybils. In addition, it is able to do so at a substantially lower computational cost than solutions of comparable effectiveness.

Therefore, this work represents a significant step toward practical Sybil defense: it enables an OSN to focus its manual inspection efforts, as well as to correctly target existing countermeasures, such as CAPTCHAs.

4

SynchroTrap: Uncovering Malicious Accounts in Social Networks by Detecting Their Loosely Synchronized Actions

In this chapter, we describe how we exploit attackers' economic constraints to uncover groups of active malicious accounts.

4.1 Introduction

Online social networks (OSNs) such as Facebook, Google+, Twitter, and Instagram are popular targets for cyber attacks. By creating fake accounts [38, 97] or compromising existing user accounts [26, 57], attackers can use OSNs to propagate spam messages, spread virus/malware, launch social-engineering attacks, or manipulate online voting results.

In this chapter, we present SynchroTrap, a system that uncovers large groups of malicious accounts by detecting their loosely synchronized actions. We have deployed SynchroTrap at Facebook for more than six months. Our insight in designing SynchroTrap is that most attackers are economically motivated and resource-limited. That is, the number of malicious accounts an attacker controls is small compared to the total number of a so-

cial network's user accounts; and an attacker is often under a strict campaign contract (e.g., obligated to acquire thousands of page likes within a few days). As a result, the malicious accounts an attacker controls tend to act in loose synchrony, specifically by taking similar actions to accomplish a campaign mission within the contracted time. In contrast, the behavior of a group of random social-network users would be much more diverse (§ 4.2.1).

Different from the social-connectivity based approaches, SynchroTrap detects both fake and compromised accounts. Different from previous feature-based classification work, SynchroTrap does not train classifiers, and uses economically constrained behavioral features to uncover malicious accounts. It is difficult or even irrational for an attacker to change those behavioral features, as doing so would undermine his economic incentives. Different from the co-clustering approach, SynchroTrap detects all types of loosely synchronized actions, including both once-only and repeated actions.

We face two main challenges in designing SynchroTrap. One is algorithmic and the other is a system challenge. Algorithmically, the actions from malicious accounts are a weak signal. Given the scale of OSNs, they constitute only a very small fraction of the total user actions. For instance, Facebook, where we have deployed SynchroTrap, has more than 600 million [22] daily active users who perform billions of actions everyday [80]. In contrast, the number of malicious accounts involved in an attack campaign is often on the order of thousands.

How can be detect such a weak signal from an enormous amount of noisy data? We address this challenge by first partitioning user actions into multiple dimensions, and then mapping the detection problem into a scalable clustering algorithm (§ 4.4.4). In particular, we categorize user actions according to the applications they target. We then compute the similarity of pair-wise user actions on the chosen dimensions to reduce the comparison of irrelevant actions. Finally, we use a clustering algorithm to group accounts with similar actions into the same cluster, and output the clusters that exceed an operator-specified threshold as groups of malicious accounts.

The system challenge we face is how to implement SynchroTrap's detection algorithm on large data sets. The daily amount of data SynchroTrap processes is on the order of a few terabytes. To address this challenge, we design a paralleled version of the single-linkage clustering algorithm [58] and implement it on Hadoop and Giraph (§ 4.5). We also design SynchroTrap as an incremental processing system that processes small data chunks on a daily basis but aggregates the computational results over the continuous data stream. This design greatly reduces the requirements on hardware capacity (e.g., memory) and the cost of failure recovery for a single job, making SynchroTrap a practical solution in the real world.

We have deployed SynchroTrap at Facebook and Instagram for more than six months. In a detailed study of one-month data (§ 4.7.2), we observe that SynchroTrap uncovered more than two million malicious accounts and 1165 attack campaigns. We have randomly sampled a subset of malicious accounts SynchroTrap caught, and asked security specialists to inspect the accuracy of the results. Their manual inspection suggests that SynchroTrap has a false positive rate lower than 1%. During the course of its deployment, SynchroTrap catches an average of $\sim 274K$ malicious accounts weekly. We have also evaluated the performance of SynchroTrap on a 200-machine cluster at Facebook. The performance results show that SynchroTrap is able to process Facebook and Instagram's user data. It takes a few hours for SynchroTrap to process the daily data and $\sim$15 hours to process a weekly aggregation job.

In summary, we make the following main contributions:

• We propose to use loosely synchronized actions to uncover malicious accounts. This design explicitly attacks an attacker's economic constraints and makes it difficult for an attacker to change the attack patterns or account features to evade detection.

• We have designed and implemented a scalable system SynchroTrap and deployed it at Facebook and Instagram. SynchroTrap uses a parallel clustering algorithm and incremental processing to address the large data challenge and can detect different types of loosely

synchronized actions from malicious accounts among billions of user accounts.

• We present an analysis of the characteristics of malicious accounts caught by SynchroTrap. This analysis may provide insight for other feature-based malicious account detection systems.

4.2 Motivation

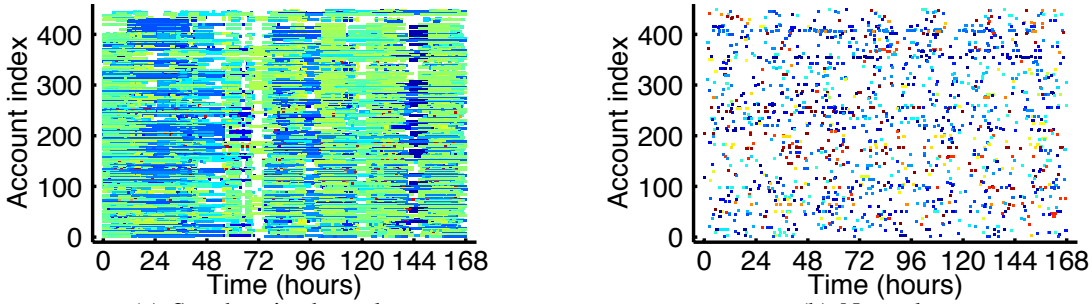
In this section, we first illustrate with real-world examples how malicious accounts loosely synchronize to launch attacks in various OSN applications, which motivates the need for a systematic defense. We then identify the main constraints on the attacker side that lead to the synchronization of account activities.

4.2.1 Real-world examples

We present two examples of loosely synchronized campaigns that we have observed in the real world. One example occurred in the Facebook photo-upload application and the other in the Instagram user-following application.

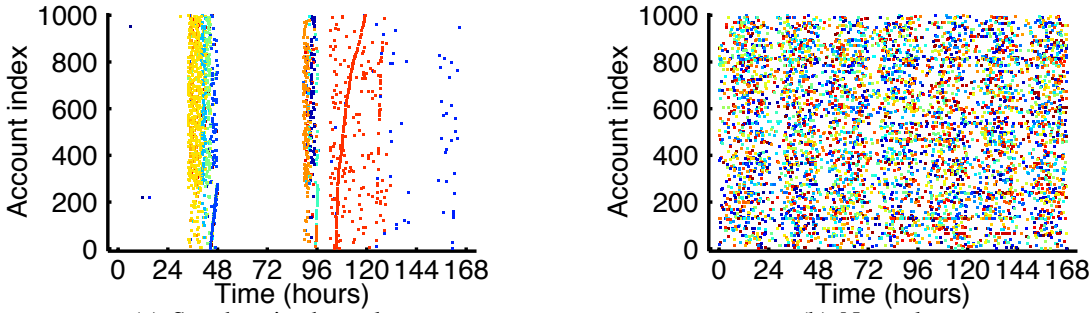
Coordinated Facebook photo uploads from the same set of IP addresses. Figure 4.1 shows a comparison of the photo-uploading activities of malicious users to normal users at Facebook. Figure 4.1(a) shows the timestamps of photo uploads of a group of 450 malicious accounts. Facebook caught those accounts because they promoted diet pills by uploading spamming photos. As we can see, those accounts use a few IP addresses for a large number of photo uploads. The vertical color stripes indicate that they bounce among IP addresses in batches during the week. Figure 4.1(b) shows the photo-uploading actions of 450 normal users, who tend to use fixed IP addresses. Moreover, the timestamp sequence of a user's photo-uploading actions is not close to other users.

Campaigns to inflate the number of followers in Instagram. Malicious users in Instagram have incentives to follow other users, or even strangers. By following users, they can inflate the followers for target users. Figure 4.2 shows a comparison of user-following activities of malicious users to normal users. The malicious accounts are comprised of 1,000 randomly sampled accounts from a campaign involving 7K accounts. We randomly sample another 1,000 normal accounts for comparison. Figure 4.2 shows the time-stamped user-following sequence for each sampled account over a week. As we can see in Fig-



(a) Synchronized attack (b) Normal

FIGURE 4.1: **An example in Facebook photo upload. The x-axis is the timestamp of an account's photo upload action, and the y-axis is an account's ID. A dot (x, y) in the figure indicates a photo upload action from an account with ID x at time y. The color of a dot encodes the IP address of the action. Photo uploads of the same color come from the same IP address.**



(a) Synchronized attack (b) Normal

FIGURE 4.2: **An example in Instagram user following. The x-axis is the timestamp of an account's following action and the y-axis is an account's ID. A dot (x, y) indicates an account x follows a targeted account at time y. The color of a dot encodes the followed account's ID. Following actions of the same color are performed to the same followed account.**

ure 4.2(a), those accounts are coordinated to follow a set of target users in batches. The entire group of accounts shows a salient on-off action pattern. During the active period, they follow the same set of users around the same time. In contrast, normal users exhibit diverse user-following behavior. As shown in Figure 4.2(b), little perceivable correlation can be found among the time-stamped user-following sequences of normal users.

These types of attacks above differ from flash crowds, which can be triggered by viral content, e.g., breaking news. This is because legitimate users in flash crowds do not exhibit

repeated similar activities and also use more dispersed IP addresses. In addition, due to the enormous diversity of behaviors among legitimate users, it is difficult to discern the malicious users in those attacks if one seeks only anomalies in each individual user's activity history.

4.2.2 Economic constraints of attackers

Independent of the OSN applications, the invariant of those attacks is that the activities of the controlled accounts are consistently correlated in the dimensions of time and the associated OSN entities. Our key insight is that attackers have to coordinate their accounts in a loosely synchronous way due to fundamental economic constraints.

Attack cost on computing and operating resources. Attackers are constrained by limited physical computing resources. Although they can purchase and compromise machines (e.g. botnets), or even rent from hosting services, the expense of purchasing, compromising, and renting computing resources imposes cost to them. Furthermore, those computing resources can be bound by limited operation time. This is because an infected machine may go offline, recover, or even be quarantined at any time [77, 105], and that machine rental is usually charged based on the consumed computing utility [18]. Another operating constraint is caused by the limited human labor that attackers use to grow and compromise accounts, and to maintain controlled accounts. In order to manage a disproportionally large number of accounts, attackers usually choose to script accounts and schedule batched activities. Therefore, these computing and operating constraints drive an attacker to perform activities from a set of machines within a constrained operational time period.

Attack revenue from missions with strict requirements. Attackers in OSNs are often deeply rooted in the black market, e.g., BlackHatWorld and Freelancer [78]. Most of their missions are driven by customer demands that contain specific requirements. Usually the objective of campaigns in OSNs is to achieve prevalence, which brings profit by affecting real users. Therefore, the mission requirements often include the *level of prevalence* that

a customer pursues and a strict *deadline* by which the mission must be accomplished. For example, many social-networking tasks in Freelancer solicit X Facebook friends/likes within Y days [78]. Similar tasks with clear requirements target other social network missions such as followers, posts, reviews, etc. Beyond that, to launch large campaigns a customer may attempt to purchase aged account credentials (e.g. Twitter, Facebook, Gmail, etc) that meet certain criteria [87, 88]. As a consequence, the strict requirements of black-market customers induce attackers to target certain aspects of the victim services and to send out action requests in advance of the mission deadlines.

In this dissertation, we call the constraints of limited computing and operating resources as *resource constraints*, and the constraints of strict requirements on attackers' missions as *mission constraints*. Because of these constraints, attackers repeatedly perform actions pertaining to certain OSN objects (e.g., IP addresses and followees) at roughly the same time over the courses of their campaigns. More importantly, those loosely synchronized campaigns can be revealed by user action logs, as OSNs usually log user actions with a number of attributes, e.g., timestamp, user agent, IP address, and other associated OSN objects. This key observation motivates our efforts to develop a systematic scheme against large groups of loosely synchronized accounts.

4.3 System Overview

SynchroTrap is a generic framework that can effectively throttle large-scale malicious accounts in OSNs. At its core, SynchroTrap has a detection module (§4.4) that uncovers groups of orchestrated malicious accounts restrained by the resource constraints and the mission constraints on the attacker side. SynchroTrap also includes a signature extraction module to extract attack characteristics and a response module to choose appropriate responses to attacks (§4.6.2).

The intuition of SynchroTrap is that the activities of malicious accounts under an attacker's control are loosely synchronized on certain associated *service entities* (e.g., IP addresses, followee IDs, etc), due to the attacker's constraints (Figure 4.3). Consequently, we define metrics to accurately measure the closeness of users in terms of the attributes and timestamps of their actions, and find large groups of malicious accounts that repeatedly perform similar actions over sustained periods of time. We further transform our detection scheme to a scalable clustering algorithm. In order to handle massive user activities at Facebook-scale OSNs, SynchroTrap employs pipelining to enable incremental data processing and partitions data with overlapping sliding windows to mitigate stragglers. With the identified user groups, an OSN provider can take action on malicious users and inappropriate content, and generate attack signatures.

4.3.1 Design goals

SynchroTrap aims to defend against large groups of malicious accounts in online social networks. Our design has the following main goals.

Generic (§4.4.2): The defense mechanism is designed to suppress malicious accounts that may be involved in a variety of large attacks. In order to be applicable to those attacks, the design of SynchroTrap should be independent of the OSN applications and service entities

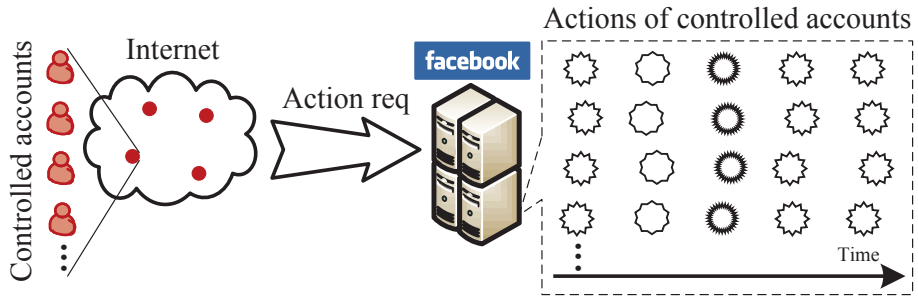


FIGURE 4.3: **Large-scale attacks in OSNs. Controlled accounts are coordinated to send out action requests to OSN servers. User actions under constraints are indicated by different shapes. From the OSN provider's view, accounts under the same control are loosely synchronized.**

that an attack targets.

Effective (§4.4.3, §4.4.4): The system should be effective in uncovering large-scale coordinated malicious accounts and should incur only a small number of false positives.

Scalable (§4.5): We design SynchroTrap for large online social networks. Due to their ever-increasing user population and user-generated content volume, the system should be efficient and able to scale up to handle services with massive user bases, such as Facebook and Instagram.

4.4 The clustering algorithm

In this section we elaborate on how to effectively and efficiently cluster users to uncover loosely synchronized user groups. We describe the centralized version of our clustering algorithm that considers the similarity on both the target service entities and the action timestamps. To reduce the impact of irrelevant actions, we categorize user actions according to OSN applications (§4.4.1). Within each application, we define a general matching metric for time-stamped user actions (§4.4.2) and quantify the similarity of a user pair using the match ratio of their actions (§4.4.3). Based on the pair-wise user similarity, we adapt the single-linkage hierarchical clustering algorithm to group users (§4.4.4). In §4.5, we parallelize this algorithm to address the large-data challenge.

4.4.1 Detecting on the basis of OSN applications

As mirrored by their large scale, OSNs provide applications with a large number of features and functions so that users may better share information and interact with each other. Malicious accounts are not necessarily coordinated across all types of actions allowed by the platforms. In order to reduce operational cost, attackers can focus on their missions and target partial dimensions of the user actions, e.g., uploading spam pictures, promoting rogue apps, etc. To better disguise themselves, attackers can even mimic legitimate users on other dimensions of user actions which are not immediately related to their missions. Accurate detection of coordinated malicious accounts poses challenges, because holistic comparison of user activities may not be able to reveal malicious user groups that target only some OSN functions. This problem is reminiscent of the "curse of dimensionality" in clustering high-dimensional data [64].

To reduce the impact of irrelevant user actions, we categorize user actions into subsets according to the applications they target, which we call *application contexts*. We then detect malicious accounts within each application context. For example, we segre-

gate contexts of photo upload and page like to detect spam photos and fraudulent page likes, respectively. Such categorization of user actions eliminates the comparison noise introduced by irrelevant actions from high-dimensional user activity.

In addition, categorizing user actions enables OSN providers to better understand user behavior of both legitimate and malicious users, because it reduces the types of actions they have to focus on within each application. Thus, OSN providers can monitor, review, and detect malicious users on an application basis, and adapt detection settings (§4.7) and responses (§4.6.2) according to the application an attacker targets. Next, we describe our user clustering method for an application.

4.4.2 Matching time-stamped user actions

In SynchroTrap, we abstract time-stamped user actions as tuples, each of which has an explicit constraint field that can express both resource and mission constraints. We require exact match on the constraint field to capture an attacker's constraints. From the point of view of OSN providers, each user action has a number of attributes. Table 4.1 summarizes the main attributes and their meanings. An *ActionType* may include an application-layer network protocol (e.g., HTTP) to indicate a fine-grained application context. An *AssocID* could be an ID of many associated service entities (e.g., users, photos, pages, etc). A user action can also have other attributes, for example the browser cookie and the HTTP user agent. In order to capture the attackers' constraints, we represent a time-stamped user action with a tuple abstraction: $\langle U, T, C \rangle$, where U, T, and C denote user ID, action timestamp, and constraint object, respectively. A *constraint object* can be a concatenation of certain service entities embedded in user action attributes, such as a combination of AssocIDs, the source IP address, etc. It is application-specific.

Our tuple abstraction of user actions is expressive. It enables SynchroTrap to quickly adapt to a specific attack in an application, if the constraint field is properly designated. For example, one can choose the followee identifier (a type of *AssocID*) as the constraint

Table 4.1: Attributes of a user action

Attribute	Meaning
UID	User ID
Timestamp	Timestamp of the user action
ActionType	User action type, e.g. login, posting, messaging, etc
AssocID	Associated entity ID this user action interacts with
IP address	IP address of the user client

field to defeat abusive user following on Instagram.

Based on the tuple abstraction, we define action *match*, denoted by "≈". Two actions of different users match if they pertain to the same constraint object and their timestamps fall into the same time window of a pre-defined length T_{sim} (e.g., 1 hour).

$\langle U_i, T_i, C_i \rangle \approx \langle U_j, T_j, C_j \rangle \quad$ if $C_i = C_j$ and $|T_i - T_j| \leq T_{sim}$

4.4.3 User-activity similarity

We measure similarity of users by comparing their action sets. Specifically, we quantify the similarity of two users by computing the ratio of their matched actions during a past period of time T_p (e.g., a week). We use the Jaccard similarity [56], a standard metric to measure the similarity of two sets. Jaccard similarity ranges from 0 to 1. A value close to 1 indicates high similarity.

Per-constraint similarity. We first introduce the per-constraint user similarity to measure the ratio of matched actions on a single constraint object (e.g., a single IP address). Let A_i be the set of actions a user U_i has performed, i.e. $A_i = \{\langle U, T, C \rangle | U{=}U_i\}$. As we enforce exact match on the constraint field of user actions, we further break down A_i into subsets according to the constraint object of each user action, i.e., where $A_i^k = \{\langle U, T, C \rangle | U{=}U_i, C{=}C_k\}$. We derive user similarity on each constraint object by Jaccard similarity. When we compute the Jaccard similarity for A_i^k and A_j^k, we apply the action matching operator "≈" (§ 4.4.2) to obtain the intersection and the union.

$$\text{Sim}(U_i, U_j, C_k) = \frac{|A_i^k \cap A_j^k|}{|A_i^k \cup A_j^k|}$$

Overall similarity. In certain OSN applications, the association of a user to a constraint object does not appear more than once. For example, in Facebook app installation, a user can install an app only once. In these cases, the Jaccard similarity of a user pair on a single constraint object (i.e., an app ID) can only be either 0 or 1. To better characterize the similarity among users, we use the overall Jaccard similarity of user actions across all constraint objects.

$$\text{Sim}(U_i, U_j) = \frac{|A_i \cap A_j|}{|A_i \cup A_j|} = \frac{\sum_k |A_i^k \cap A_j^k|}{\sum_k |A_i^k \cup A_j^k|}$$

4.4.4 User clustering

We choose to use the single-linkage hierarchical clustering [58] as a building block for SynchroTrap, due to its scalability and effectiveness. There are other off-the-shelf clustering schemes (e.g., k-means). We do not use those schemes because they either rely on a special distance metric (e.g., Euclidean distance in k-means), or are not scalable. We refer readers to [58] for a complete review of the clustering techniques. In addition, we do not seek to use graph partitioning algorithms for clustering users at Facebook, because even the widely-used graph partitioning tools like METIS [63] take many hours to process a graph with only 41 million nodes [92]. Instead, our objective is to transform our attack detection scheme to a clustering algorithm that can scale up to large OSNs.

Single-linkage hierarchical clustering. Single-linkage hierarchical clustering is an agglomerative approach that begins with each user as a different cluster, and iteratively merges clusters with high similarity and produces larger clusters. Such an algorithm generates a cluster-merging *dendrogram* that illustrates the merging hierarchy of user clusters

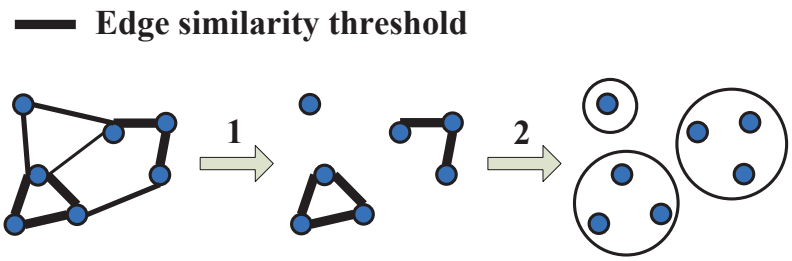


FIGURE 4.4: **Adapting the single-linkage clustering algorithm to the connected components algorithm in two steps. User pairs connected by thicker edges have higher similarity.**

and the degree of similarity on each level. By breaking the dendrogram at a desired level, one obtains a set of clusters in which intra-cluster user similarity exceeds a certain threshold. A detailed description of the algorithm is documented in [58]. Because this algorithm relies on a sequential process to construct the entire dendrogram in a bottom-up fashion, a straightforward implementation is hard to scale.

Adapting the algorithm. The key property of single-linkage hierarchical clustering is that the similarity of two clusters is determined by the maximum similarity among all pairs of users drawn from different clusters. This cluster-similarity measure translates a group of close clusters in each iteration into a larger connected component in a *user similarity graph*, where nodes are users and an undirected edge exists between a pair of users if their similarity is above a certain threshold.

Using this property we adapt the single-linkage hierarchical clustering algorithm to obtain a parallel version. Our idea is that if we set the similarity threshold first and filter out user pairs below that, the desired user clusters are exactly the connected components in the pruned user similarity graph. Therefore, we can employ an efficient graph algorithm [60] to search for connected components. Figure 4.4 illustrates our two-step adaptation of the single-linkage clustering algorithm. We choose to adapt to the connected components algorithm because it is highly scalable on massive graphs due to its inherent parallism [60].

User-pair filtering function. We use a filtering function to screen out user pairs with

Algorithm 1 userMatch (U_i, U_j)

```
1: // K is the total number of constraint entity instances
2: // Len_thre is the threshold on the number of user actions
3: // Sim_pc, Sim_overall are thresholds on per-constraint and overall similarity
4: // Obj_thre is the threshold on the number of constraint objects
5: pcSim ← false;
6: overallSim ← false;
7: objCounter ← 0;
8: while k ≤ K do
9:    if |A_i^k| ≥ Len_thre and |A_j^k| ≥ Len_thre and
          Sim(U_i, U_j, C_k) ≥ Sim_pc then
10:      objCounter ← objCounter + 1;
11:   end if
12: end while
13: if objCounter ≥ Obj_thre then
14:   pcSim ← true;
15: end if
16: if Sim(U_i, U_j) ≥ Sim_overall then
17:   overallSim ← true;
18: end if
19: return pcSim or overallSim
```

action synchronization above a certain degree. We introduce two criteria to decide on a user pair according to the types of similarity we have defined (§ 4.4.3).

• *F1:* There exists at least one constraint object, for each of which users have a per-constraint similarity above a certain threshold.

• *F2:* Their overall similarity is above a certain threshold.

The first filtering criterion uncovers malicious user pairs that manifest loosely synchronized behaviors on each of a set of constraint objects (e.g., IP addresses). In some cases, malicious accounts may even spread their actions over a number of constraint objects. In criterion *F2*, we examine the overall similarity to deal with the applications where a user can carry out actions only once per constraint object. Algorithm 1 shows the pseudocode of our filtering function.

4.5 Addressing the large-data challenge

The major challenge in making SynchroTrap a practical system is to process the large volume of user activities in OSNs. For example, more than 600 million daily active users [22] rely on Facebook to store and share information. Large and complex batch computations at Facebook-scale services can be prohibitive due to the requirements on hardware capacity (e.g., memory) and the cost of failure recovery. In this section, we describe how we enable incremental processing in SynchroTrap and derive a scalable implementation based on Hadoop [20] and Giraph [19].

4.5.1 Incremental processing

Instead of performing large batch processing on bulk data sets, our main idea is to split computation on a bulk data set into a series of smaller computations and to pipeline data chunks between them. We implement SynchroTrap as a pipeline of Hadoop jobs and store the intermediate results in HDFS. This greatly reduces the size of a single job and thus its hardware consumption, making SynchroTrap a more manageable solution in practice. In addition, by splitting large intermediate data into small logical chunks, our processing pipeline reduces the recovery cost in the face of failures and simplifies system management.

Pipelining. Figure 4.5 shows the data flow of the processing pipeline in SynchroTrap. At Facebook, we slice the computation of user comparison and designate daily jobs to generate similar user pairs based on the user-activity log. Because SynchroTrap seeks consistently loosely synchronized activities over a sustained period of time, we aggregate daily similarity measures and perform user clustering periodically (e.g., weekly). Our pipelining system enables the aggregate jobs to reuse the results of daily jobs. As shown in Figure 4.5, a new aggregation job does not incur re-execution of daily jobs.

Aggregating user similarity. We develop an aggregatable data interface between the

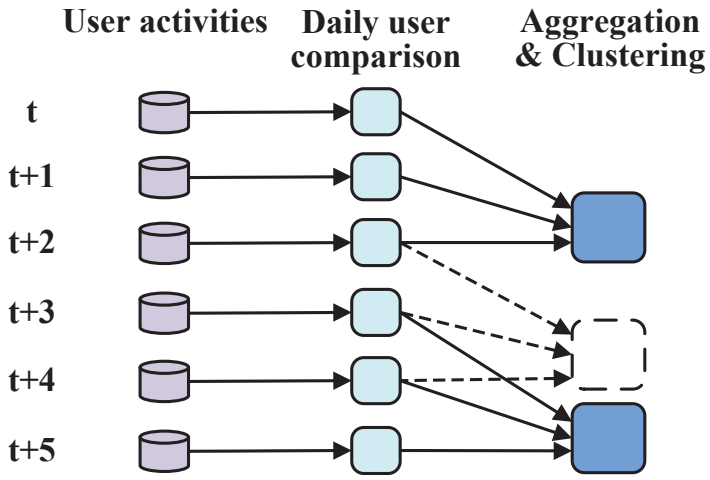


FIGURE 4.5: **SynchroTrap's processing pipeline at Facebook. A new aggregation job (dashed) does not incur re-execution of daily jobs. Arrows indicate the data flow.**

daily jobs and the aggregation jobs. We design a data structure for aggregatable user similarity by decomposing the period-long user similarity in §4.4.3 over days. Let $A^k_{i,t}$ denote the actions on constraint object C_k that user U_i issues on day t, i.e. $A^k_{i,t} = \{\langle U, T, C\rangle | U{=}U_i, C{=}C_k, T \text{ is within day } t\}$. For a user pair (U_i, U_j) and a constraint object C_k, we generate and store the number of daily action matches $|A^k_{i,t} \cap A^k_{j,t}|$, and the daily total actions each of them has carried out, i.e. $|A^k_{i,t}|$ and $|A^k_{j,t}|$. By aggregating the daily results as below, one derives the user similarity $Sim(U_i, U_j, C_k)$ over a course of time.

$$\begin{aligned}\text{Sim}(U_i, U_j, C_k) &= \frac{|A^k_i \cap A^k_j|}{|A^k_i \cup A^k_j|} = \frac{|A^k_i \cap A^k_j|}{|A^k_i| + |A^k_j| - |A^k_i \cap A^k_j|} \\ &= \frac{\sum_t |A^k_{i,t} \cap A^k_{j,t}|}{\sum_t |A^k_{i,t}| + \sum_t |A^k_{j,t}| - \sum_t |A^k_{i,t} \cap A^k_{j,t}|}\end{aligned}$$

The last equality holds because the user action match across days is rare, especially the action-match window is usually set to minutes or a few of hours.

4.5.2 *Daily user comparison*

We allocate daily jobs to count the daily user action matches on each constraint object. The actions issued from a user are packed in different sets, i.e. $A^k_{i,t}$, according to the constraint objects. To measure the daily action matches of users U_i and U_j on a constraint object C_k, we extract their timestamp sets from their actions $A^k_{i,t}$ and $A^k_{j,t}$, and count the pairs of timestamps drawn from the two sets that fall into the same time window of size T_{sim}. The timestamp matches can be computed in a single pass after sorting each set.

Partitioning by constraints. Because a user-action match requires an exact match on the constraint field, the comparison of user actions on different constraint objects can be performed in parallel. With the Map-Reduce paradigm, we leverage this data parallelism by routing the user-action sets tied to the same constraint object to the same worker for comparison.

Mitigating stragglers with overlapping sliding windows. A straightforward implementation of the daily user comparison yields jobs whose overall completion time could be prolonged for days by straggler reducers. This straggler issue originates from the highly skewed distribution of user actions over the constraint objects. Even on a daily basis, there exist hot objects that are associated with an extremely large number of actions/users. Figure 4.6 shows the distribution of the number of login users over IP addresses on one day at Facebook. As we can see, while most of the IP addresses are not used by many users (less than 100), the user population distribution is heavy-tailed. These few popular IP addresses that are used by more than $100K$ daily active users lead to straggler reducers, which could run for days when SynchroTrap is applied to IP addresses at Facebook login.

We mitigate this issue by further partitioning users with their actions on large constraint objects. When the number of users in a partition is reduced by a factor of $\frac{1}{p}$ ($p > 1$), the execution time can be reduced by a factor of $\frac{1}{p^2}$, because we perform a quadratic number of user-user similarity computations. The challenge is how to effectively partition data,

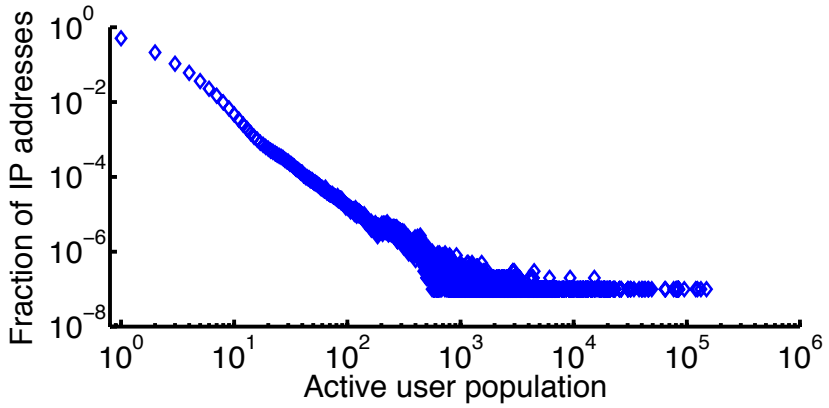


FIGURE 4.6**: Distribution of user population over IP addresses in login. We depict the fraction of IP addresses with respect to the number of active users per IP address.**

while retaining the capability of accurately capturing user action matches. Our insight is to divide the user actions into overlapping sliding windows in time and to process different sliding windows in parallel. This method not only mitigates the straggler issue by feeding each individual worker smaller chunks of data, but also reduces the overall computation for a large constraint object because user actions that reside in different sliding windows are effectively filtered out before the computation.

Figure 4.7 illustrates how we set up overlapping sliding windows to precisely capture all possible user-action matches with a matching window set to T_{sim}. In principle, a sliding window size $> T_{sim}$ and an overlapping period $\geq T_{sim}$ can guarantee the full coverage of user-action matches. This is because a sliding window size $> T_{sim}$ ensures that any user-action match with a maximum spanning period T_{sim} can be covered by a sliding window; an overlapping period $\geq T_{sim}$ ensures that the sliding windows are dense enough to cover sticky user-action matches across windows.

On the other hand, counting in each overlapping sliding window entails duplicates of user-action matches that appear in the overlapping area. The de-duplication of action matches could be complicated if the sliding window size and the overlapping period are

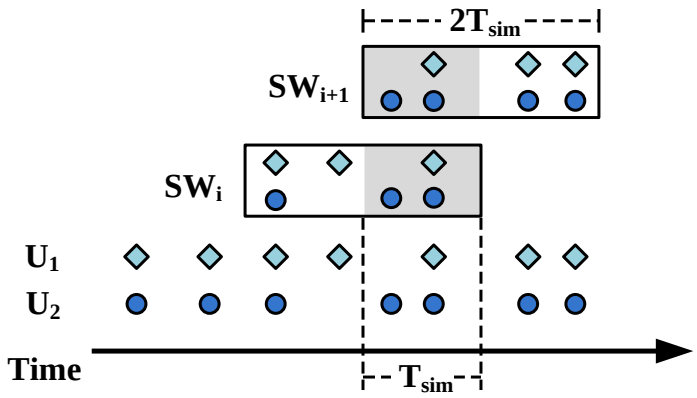


FIGURE 4.7: **Settings of the overlapping sliding windows in SynchroTrap. Action matches within the overlapping area (shaded) are exactly double counted. We depict actions of different users using different markers.**

not properly set. We choose to use a sliding window size of $2T_{sim}$ and an overlapping period of length T_{sim} which strike an appropriate balance between the effectiveness of cutting the data volume within each sliding window and the complexity of de-duplicating action matches. With such settings, we achieve single counting by simply discarding the user action matches within the second (or the first) half of each sliding window.

This counting scheme guarantees that one can exactly single count all user-action matches only by independently couting in each sliding window. Suppose we have a sequence of sliding windows $SW_i = [iT_{sim}, (i+2)T_{sim}]$ $(i \geq 0)$. Without loss of generality, let t_{a_1} and t_{a_2} be the timestamps of two matched user actions, where $t_{a_2} \in [t_{a_1}, t_{a_1} + T_{sim}]$. Suppose $t_{a_1} \in [jT_{sim}, (j+1)T_{sim})$ $(j \geq 0)$. We have $t_{a_2} < (j+2)T_{sim}$. There are two cases regarding to the location of the action pair: 1) $t_{a_2} < (j+1)T_{sim}$. Both t_{a_1} and t_{a_2} belong to the interval $[jT_{sim}, (j+1)T_{sim}]$, which is the overlapping area of two consecutive sliding windows SW_{j-1} and SW_j. Because we discard action matches within each second-half window, the action pair is only single counted in SW_j; 2) $t_{a_2} \in [(j+1)T_{sim}, (j+2)T_{sim})$. Only SW_j covers both t_{a_1} and t_{a_2}, because $t_{a_1} \in [jT_{sim}, (j+1)T_{sim})$. Hence the action pair is single counted in SW_j. We always append an empty half window after the last sliding window in order to deal with the extreme case at the end of the data stream.

4.5.3 Aggregation and clustering

Daily action matches and daily individual total actions are aggregated over a course of time. We join the aggregate action matches and the aggregate individual total actions, and compute the per-constraint similarity and overall similarity for each user pair, i.e., $Sim(U_i, U_j, C_k)$ and $Sim(U_i, U_j)$. After collecting similarity measures across all constraint objects, we determine similar user pairs by applying the user matching function. Finally, the algorithm of connected components is used to group those users.

We adopt a scalable implementation of the connected components algorithm [60], which identifies connected components in a graph by iteratively propagating component IDs. A component ID is defined as the smallest node ID in that component. At the beginning, each node uses its own node ID as its component ID. During each iteration, a node is allowed to propagate its component ID to all neighbors and update its component ID after receiving from its neighbors, i.e., to replace its ID with a neighbor's if the ID from the neighbor is smaller. At the convergence, all nodes in the same component reach consensus on their component ID, which enables us to identify them as a cluster.

4.5.4 Implementation

We implemented SynchroTrap's detection engine on top of the Hadoop MapReduce stack [89] at Facebook. The daily user comparison module and the aggregation module are implemented on Hadoop [20], and the clustering module is implemented on Giraph [19], a large-graph processing platform based on the Bulk Synchronous Parallel (BSP) model [70]. Giraph provides a parallel implementation of the connected components algorithm. Apart from the basic functions offered by Facebook's infrastructure, our implementation of SynchroTrap's detection engine consists of 2,500 lines of Java code and 1,500 lines of Python code.

4.6 Deployment at Facebook and Instagram

We deployed SynchroTrap in Facebook and Instagram to uncover malicious accounts and integrated SynchroTrap into the site-protecting stack at Facebook. In this section we discuss five use cases (§4.6.1) and describe how the findings of SynchroTrap can be used to better monitor and protect OSN services (§4.6.2).

4.6.1 Use cases at Facebook and Instagram

As discussed in §4.2.2, attackers can be constrained by their resources and their missions. We present SynchroTrap use cases according to the constraint by which an attack campaign is bound.

Resource-constrained synchronization. We designate source IP addresses as the constraint field in SynchroTrap and use the per-constraint similarity to detect the resource-constrained attacks. We deployed SynchroTrap with this configuration in Facebook user login and photo upload. An OSN could also set the SynchroTrap's constraint field to include certain security cookies [21, 29], which would enable the detection of resource-constrained attacks at a finer granularity.

Mission-constrained synchronization. The attacks constrained by missions exhibit loose synchronization on mission-related service entities. We deployed SynchroTrap at Facebook app installation and page like, and at Instagram user following by designating the constraint field as Facebook app ID, Facebook page ID, and Instagram followee ID, respectively. We used the overall similarity in these use cases.

4.6.2 Signatures and response

Apart from uncovering the loosely synchronized accounts and their activities, SynchroTrap generates attack signatures and takes action on malicious accounts and content.

Attack signatures. SynchroTrap extracts the common constraint objects on which suspi-

cious accounts in the same group manifest synchronization. The OSN entities embedded in those constraint objects are abusive, and thus used as attack signatures. They include rogue Facebook apps, Facebook pages with inappropriate content, abusive Instagram accounts soliciting excessive followers, etc. By tracking back to the complete use action log, SynchroTrap can provide the fingerprints of an attacker's machines, including IP addresses, user agents, browser cookies, etc. All of the above signatures are used to decide on proper responses and to build fast policies to suppress attacks in nearly real time [86].

Response. The response to attacks in SynchroTrap is multifold: large groups of detected accounts are challenged with CAPTCHAs; actions performed during the campaigns are invalidated in retrospect; and the user-created content, such as photos, is sent for automated sanity check (e.g., photoDNA [24]) or manual inspection.

4.7 Evaluation

In this section, we evaluate SynchroTrap using a one-month execution log from Facebook in August 2013. We answer the following questions to demonstrate that SynchroTrap provides a practical solution for large OSNs:

• Can SynchroTrap accurately detect malicious accounts while yielding a low false positive rate?

• How effective is SynchroTrap in uncovering new attacks?

• Can SynchroTrap scale up to Facebook-scale OSNs?

We evaluate SynchroTrap's detection accuracy by manually inspecting sampled accounts and activities it uncovered. We then study the new findings through cross-validation against existing approaches that run at Facebook. We examine the social connectivity of the identified accounts by using SybilRank [38], the first scalable social-graph-based fake account detection system for large OSNs. We also share the operation experience to shed light on how SynchroTrap works in practice over time. Last, we demonstrate the scalability of SynchroTrap using performance measurements obtained from a 200-machine cluster.

4.7.1 SynchroTrap parameter settings

The main parameters used in SynchroTrap include the action-matching window T_{sim} and the thresholds for per-constraint similarity (Sim_{pc}) and overall similarity ($Sim_{overall}$). As we will discuss in §4.7.6, the implementation of SynchroTrap is scalable with a broad range of parameter settings because of its pipelining architecture and data partitioning with overlapping sliding windows. Therefore, we are not concerned with the system performance when setting parameters. Nevertheless, determining parameter values with satisfactory accuracy is complex for a production system such as SynchroTrap. First, the volumes and synchronization levels of malicious actions are variant in different OSN applications. In

some cases, attackers may even change their strategies over time. Second, as a system that influences user experience, SynchroTrap is required to set the parameters conservatively so that it only yields a near-zero false positive rate, i.e., a near-zero fraction of the identified accounts are actually legitimate users.

We set proper values for SynchroTrap parameters with assistance from human operators. The values of the action-matching window T_{sim} and the user similarity thresholds have monotonic effect on false rates: a larger action-matching window only enables SynchroTrap to find a super set of matched actions for two users, and hence increases their similarity on a constraint object; a larger user similarity threshold only prunes more user pairs, which results in a subset of similar user pairs. This monotonic effect simplifies the process of setting parameters and reduces the need for human intervention. The operators of SynchroTrap can choose to tune parameter values up or down according to the false positive rate with the current settings. At Facebook we set SynchroTrap parameters to the values that meet the internal production requirements. We do not reveal the specific parameter settings due to confidentiality agreements.

4.7.2 Validation of identified accounts

We first validate the malicious accounts with support from the Facebook security team. We proceed with the investigation of the confirmed accounts to understand how adversaries managed to take control of them. Moreover, we study the network-level characteristics of the detected attacks, including the email domains and IP addresses used by malicious accounts.

Methodology. The main challenge to validate the detected accounts and their actions is the large volume. During the month of our study, SynchroTrap uncovers millions of accounts. Manually reviewing all those accounts imposes prohibitive workload. Still, because a large fraction of detected accounts appeared in new attacks (§4.7.3), comparing them to the attack corpus of existing approaches would not provide a holistic view of

the ground truth. Therefore, our approach is to inspect only representative samples of the detected accounts with assistance from the security specialists. We randomly sample representative sets of detected accounts for inspection and thereby obtain the false rates for the corresponding super sets of the accounts.

Table 4.2: Identified attacks and the true positive rate of the suspicious accounts. The true positive rate is derived from manual inspection of randomly sampled accounts by the Facebook security team.

Application	Page like	User follow	App install	Photo upload	Login
Campaigns	201	531	74	29	321
Accounts	730,656	589,582	164,664	120,918	564,804
Actions	357M	65M	4M	48M	29M
True positive rate	99.0%	99.7%	100%	100%	100%

True positive rate. Table 4.2 shows the suspicious activities uncovered by SynchroTrap and the true positive rate in each application. In total, SynchroTrap detected 1165 large campaigns involving more than 2 million malicious accounts, with a false positive rate of less than 1%. Table 4.2 shows that the large campaigns are comprised of millions of malicious user actions. Among the five deployed applications, attackers were more active in page like and user following, presumably because attacks/campaigns in these applications are more lucrative. By uncovering large campaigns, SynchroTrap allows Facebook and Instagram to identify and properly invalidate millions of malicious user actions in each application.

Post-processing to deal with false positives. False positives are detrimental to the OSN user experience. Along with adding human intervention to the process of setting parameters, we further reduce false positives through post-processing. First, we discard small user clusters and screen out only large clusters, which are more likely to result from large attacks. The Facebook security team sets a threshold of 200, above which almost all users in each cluster are malicious. Second, we do not invalidate all actions a malicious account has performed during a detection window T_p, but conservatively focus only on those that

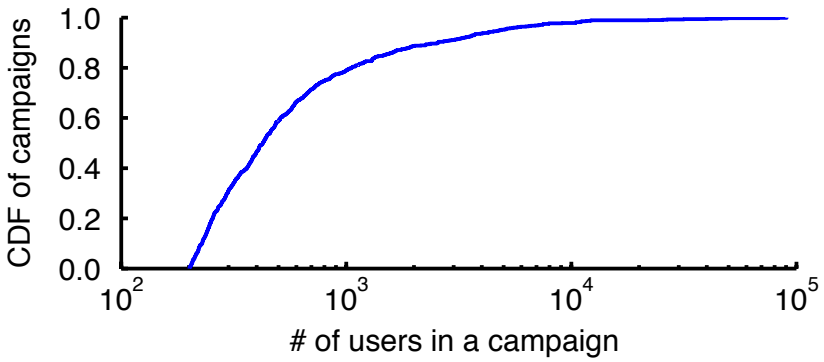


FIGURE 4.8: **CDF of campaigns with respect to the number of involved users. In large campaigns, attackers manipulate thousands of malicious accounts.**

match at least another action of each account in the same cluster. This helps to rule out the valid actions that accounts may have delivered before being compromised.

Scale of campaigns. Figure 4.8 shows the CDF of the scale of the campaigns after post-processing, in terms of the number of the controlled accounts involved. Whereas 80% of the campaigns involve fewer than 1000 accounts, we also find very large campaigns, in which attackers manipulate tens of thousands of accounts.

How are the malicious accounts taken under control? Because attackers have to use accounts to perform malicious activities in OSNs, it is critical for them to own or hijack a large number of accounts before launching their campaigns. To understand how adversaries take control of accounts, the Facebook security team classifies the reviewed accounts into categories based on how they were involved in campaigns. The means by which attackers harness malicious accounts include creating fake accounts with fraudulent user information [97, 38], compromising user accounts via malware [26], stealing user credentials and social connections by social engineering [57, 35], etc. A breakdown of the malicious accounts in app installation is shown in Table 4.3. In this application context, attackers manipulate accounts to promote rogue Facebook apps that can be used later to send out spam, steal user personal information, etc. Clearly, fake accounts, social engi-

neering, and malware are the dominant malicious account sources, accounting for more than 90% of the detected accounts.

Table 4.3: Classification of the malicious accounts in Facebook app install.

Fake accounts	Social Engineering	Malware	Others
28.6%	21.4%	42.9%	7.1%

Network-level characteristics. We study the email domains and IP addresses used by malicious users to shed light on the network-level characteristics of attacks.

An OSN account usually associates to a contact email address. Figure 4.9 shows the distribution of the email domains of the identified accounts in each application context. As we can see, the email credentials used by the controlled accounts are mainly from five domains, including those major email domains Yahoo, Hotmail, and Gmail. Email domains with accounts that can be obtained from the underground market (e.g., Yahoo, Hotmail, and AOL) are likely to be used to provide fraudulent contact email addresses for controlled accounts. Whereas Gmail accounts incur higher cost to attackers according to an underground market survey [88], a fraction of the identified accounts are found to use Gmail addresses. In addition, a non-negligible fraction of the contact email addresses are from the domain *.ru, which is dominated by mail.ru and yandex.ru. Because the identified accounts used a diverse set of email addresses, this result suggests that the email domain alone is not a reliable criterion to detect malicious accounts.

We further study the source IP addresses of the detected malicious activities. We found that the detected accounts have used $\sim$1.2 million IP addresses in total. Figure 4.10 plots the distribution of the IPv4 addresses used by attackers in each application context. As can be seen, the threats are launched from three major regions of the IP address space: 36.67.* – 44.99.*, 77.246.*–125.226.*, and 170.226.* – 207.78.*. The distributions of IP address in different applications are close to each other, except that attackers in app install

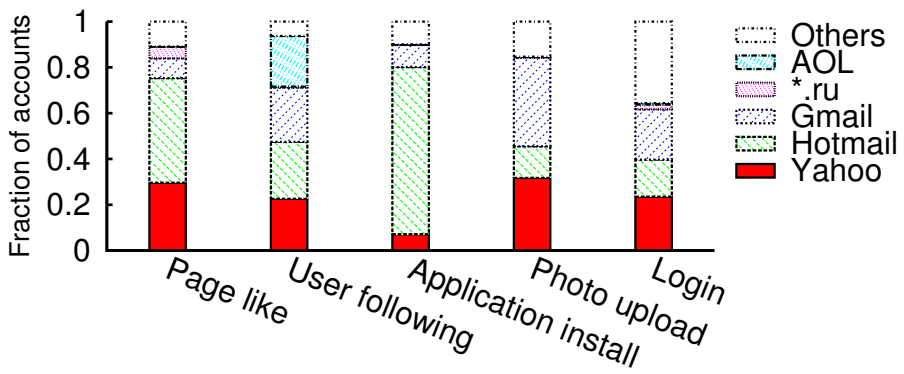


FIGURE 4.9**: Breakdown of top email domains associated to the malicious accounts in each Facebook application.**

more intensively use the IP addresses from the region 77.246.*–125.226.*. We investigate a random sample set of those IP addresses via queries to WHOIS servers, which provide the registration information of the domain names. Many IP addresses are administrated by large regional ISPs around the world (e.g., Global Village Telecom in Brazil and Saudi Telecom in Saudi Arabia). Some of those IP addresses are used to provide shared Internet access (e.g., for network proxies or public Internet access points). We also observed that a non-trivial fraction of the IP addresses are from hosting services such as GoDaddy and Singlehop, as well as from large cloud computing services such as Amazon AWS. This observation indicates that cloud-based services provide another avenue for threats to break into the Internet, which is in contrast to traditional botnet-based attacks [96].

4.7.3 New findings on malicious accounts

To evaluate SynchroTrap's ability to find malicious activities that were previously undetectable, we compare the malicious accounts detected by SynchroTrap against those detected by existing approaches inside Facebook. A large set of the existing approaches mainly deals with aggressive attacks by monitoring abrupt changes in certain types of user activities [86]. In each application, the accounts detected by SynchroTrap in August 2013 are compared to those detected by other approaches during the same period. Table 4.4

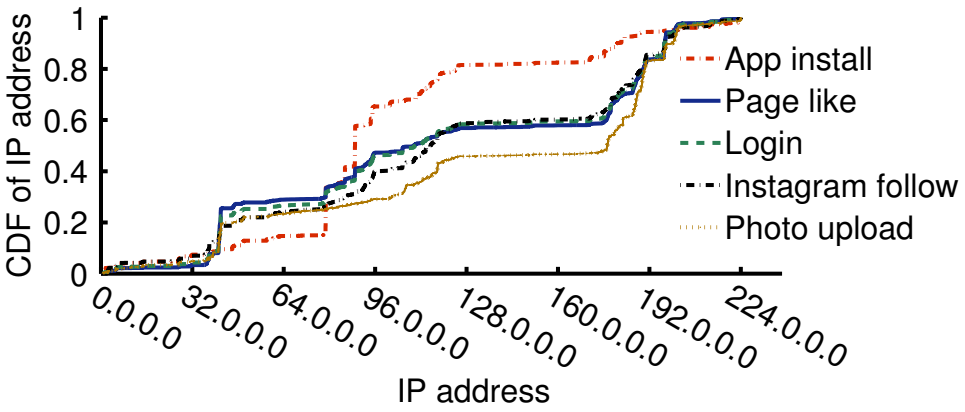


FIGURE 4.10**: Distribution of the IPv4 addresses used for campaigns in each Facebook application.**

shows the overlap of the malicious accounts that SynchroTrap and other approaches identified, as well as SynchroTrap's new findings. As we can see, SynchroTrap identified a large number of previously unknown malicious accounts. Specifically, in each application, at least 70% of the identified accounts were not discovered by existing approaches. We investigated the exact numbers of accounts detected by existing approaches. We cannot report them due to confidentiality, but SynchroTrap covers a fairly large portion of those accounts. We believe that full-fledged deployment of SynchroTrap in each application context on more service entities (e.g., certain fields of browser cookies) could yield more new findings and achieve higher recall of malicious accounts.

In particular, the large number of previously undiscovered malicious accounts indicates that the loosely synchronized attacks have been underestimated in existing countermeasures. SynchroTrap complements existing OSN countermeasures by effectively uncovering such attacks.

4.7.4 Social connectivity of malicious accounts

Social-graph-based defense mechanisms have attracted much attention from the research community [100, 99, 90, 93, 38]. We examine the social connectivity of the identified accounts by comparing them against the ranked list generated by SybilRank [38]. Sybil-

Table 4.4: New findings of SynchroTrap. It uncovers a significant fraction of malicious accounts that are used to launch new attacks. SynchroTrap is the first dedicated countermeasure in app install and photo upload at Facebook.

Context	Overlap with existing approaches	New findings by SynchroTrap
Page like	175,001	555,655
Instagram user follow	66,430	523,152
App instal	N/A	164,664
Photo upload	N/A	120,918
Login	12,769	552,035
Total	254,200	1,916,424

Rank discerns bulk accounts created at a low per-account cost. It ranks users based on their connectivity in the social graph. As a result, suspicious users with limited connections to legitimate users are ranked low.

We run SybilRank on a snapshot of the Facebook friendship graph obtained in August 2013. This social graph contains all Facebook users that have been perceived as benign by various existing countermeasures [86] until this study. We do not include the users already stopped by existing countermeasures before the graph snapshot. Figure 4.11 shows the CDF of the ranking percentile of the malicious accounts that SynchroTrap detects in each Facebook application. As we can see, a certain fraction of malicious users ($\sim$40% in login and $\sim$15% in each of other applications) are ranked at the bottom, That portion of users are comprised of fake accounts that have little engagement on the social graph. Whereas SybilRank gives low rankings to a large fraction of the identified malicious users (e.g., 80% of the detected users in app install are among the 25% lowest rankings), a non-negligible fraction of the users appear in the middle or even the top intervals of the ranked list. This indicates that attackers manipulate a variety of accounts in terms of the degree of their social connectivity to legitimate users. For example, part of the accounts caught in photo upload are ranked high, presumably because attackers prefer to use well-connected accounts to spread spamming photos to many of their friends. As described in §4.7.2,

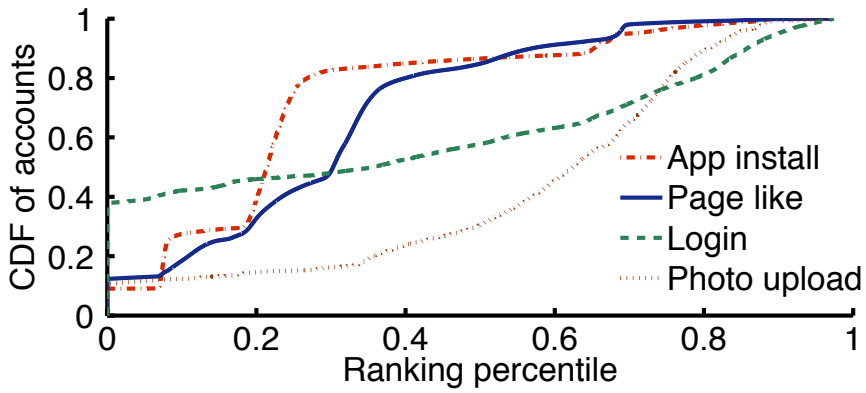


FIGURE 4.11: **CDF of the detected accounts with respect to the ranking percentile generated by SybilRank. The percentiles are calculated from the bottom of the ranked list.**

those kinds of well-connected accounts can be obtained via malware, social engineering, etc. The potential influence on the social graph and the higher cost of getting such accounts make them more valuable to attackers.

4.7.5 Operation experience

SynchroTrap has been in production for months. We perform a longitudinal study on the number of users caught by SynchroTrap for the first 11 weeks after SynchroTrap's deployment (Figure 4.12). From the beginning, the variation is small in app install, photo upload, and login. In contrast, the number of detected users decreases after the first month in page like and Instagram user following. It is then stabilized around 100K per week. Presumably this drop is because some of controlled accounts were switched to the dormant state as a result of SybilRank's deterrence. The stable detection numbers in each application suggest that SynchroTrap continued to effectively detect malicious accounts over time.

Although most of the detected accounts were being caught for the first time by SynchroTrap, we observed that a non-negligible fraction of them were repeatedly caught. Figure 4.13 shows the fraction of these users (who were caught at least twice). As we can see, 5%∼10% of the detected users in each application context are caught twice by SynchroTrap; the fraction of users caught more than 5 times is less than 1%. SynchroTrap enforces

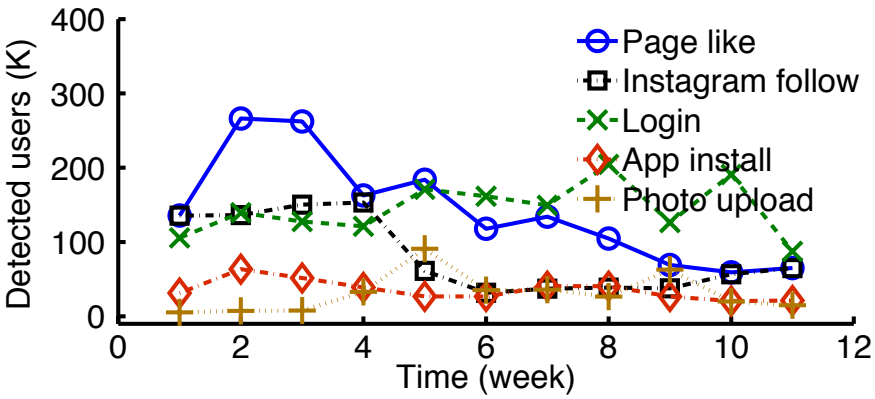


FIGURE 4.12: **Number of users detected by SynchroTrap per week over the course of 11 weeks.**

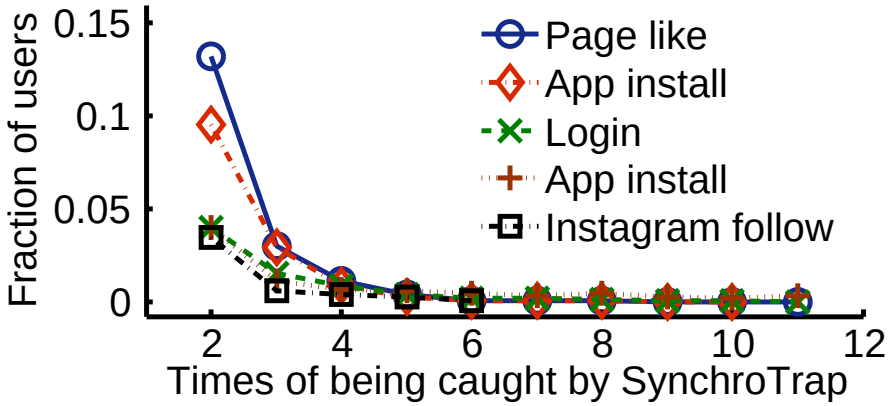
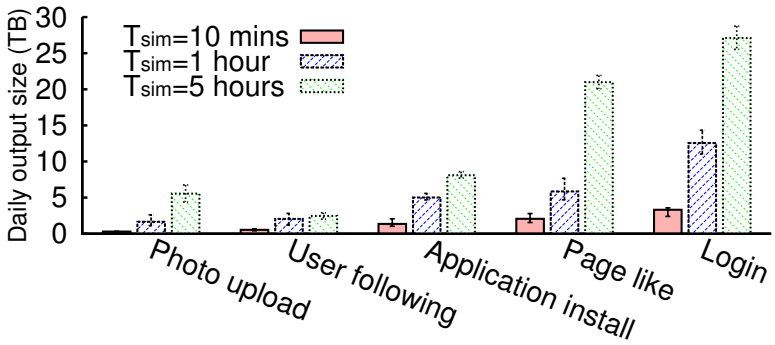


FIGURE 4.13: **Distribution of the users repeatedly caught by SynchroTrap. We depict the fraction of detected users with respect to the number of times they have been caught.**

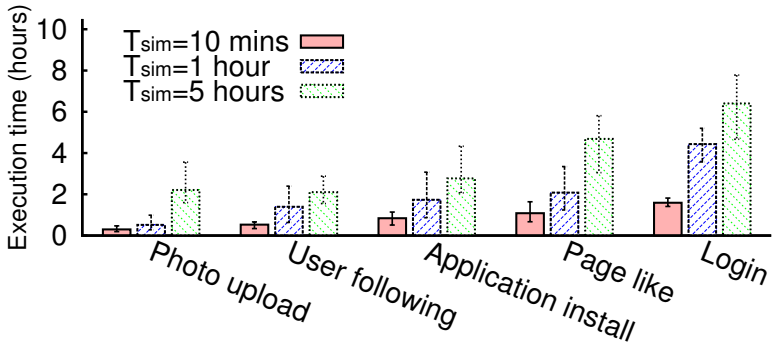
challenges to detected users such as CAPTCHAs, which indicates that either the attackers or the owners of the compromised accounts have cleared the challenges that were sent to the detected accounts.

4.7.6 System performance

We evaluate the performance of SynchroTrap on a 200-machine cluster at Facebook. The daily raw activity data ranges from 100GB to 1TB in each application context. We draw tuples $\langle U, T, C \rangle$ from each raw activity record and store them in the HDFS across the cluster as the daily input for SynchroTrap. We evaluate the performance of each SynchroTrap's



(a) Daily output size



(b) Execution time of daily jobs

FIGURE 4.14: **The output volume and execution time of SynchroTrap's daily jobs in each deployed application context. We set T_{sim} to 10 minutes, 1 hour, and 5 hours to examine its impact. Error bars represent 95% confidence intervals.**

pipeline stage and examine the impact of its parameters.

Daily jobs. Daily jobs generate similar user pairs on a daily basis, which are in the form of $\langle U_i, U_j, C_k, |A^k_{i,t} \cap A^k_{j,t}|, |A^k_{i,t}|, |A^k_{j,t}| \rangle$ (§4.5.1). The action-matching window T_{sim} determines the size of the sliding comparison windows (§4.5.2). We set T_{sim} to 10 minutes, 1 hour, and 5 hours to examine its impact on the daily output volume and the execution time. Figure 4.14(a) shows the volume of daily user pairs generated in each application context. As we can see, the output volume increases as we set T_{sim} to higher values. This is because a higher T_{sim} value allows larger sliding comparison windows, which results in additional user pairs with matched actions. The increment of output volume varies in different applications because of the distinct user-activity patterns. For example, setting T_{sim} to 5 hours leads to at least twice as many user pairs than 1 hour in photo

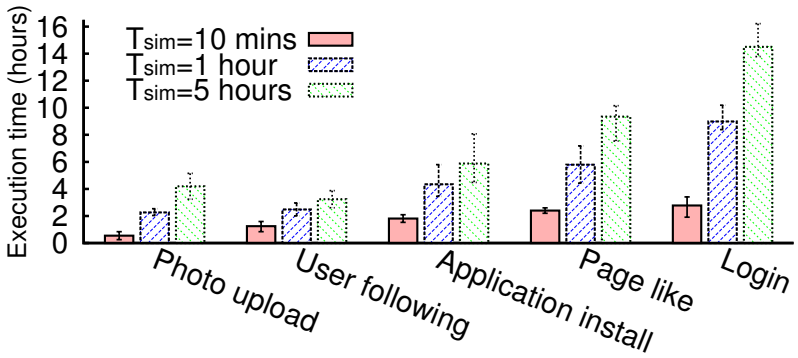
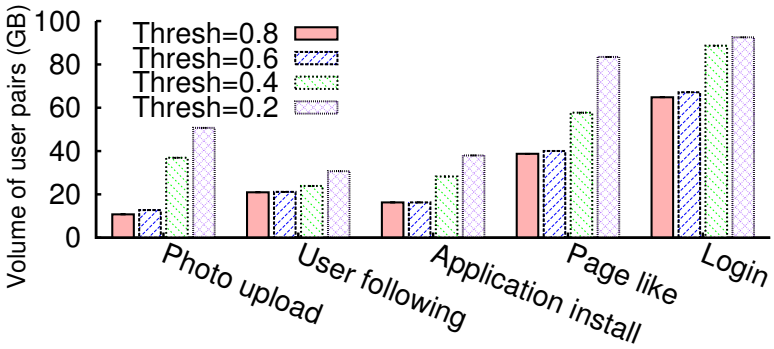


FIGURE 4.15: **Execution time of aggregation jobs in each application context. The input volume varies as we generate the daily output using different T_{sim} values (10 minutes, 1 hour, and 5 hours). Error bars represent 95% confidence intervals.**

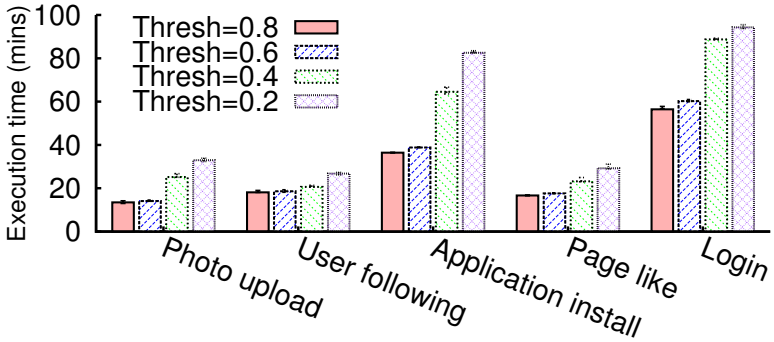
upload, page like, and login, whereas the increment is relatively small in Instagram user following and app install. Similarly, the execution time of daily jobs also increases with T_{sim} (Figure 4.14(b)). The extra execution time incurred by a higher T_{sim} is used for comparison among the additional users within a larger sliding comparison window. Due to our data partitioning with overlapping sliding windows, each daily job in an application context can finish within a few hours.

Aggregation. Aggregation jobs collect the matched actions of each user pair and the number of actions of each individual user over a time period (i.e., one week). It then computes the per-constraint and overall similarities for each user pair generated by daily jobs. Figure 4.15 plots the execution time of the aggregation jobs in each application context. The aggregation jobs are fed with larger volumes of user pairs when we increase T_{sim} in the daily jobs. As we can see, the aggregation jobs need longer time to process a larger volume of user pairs. In all application contexts with different T_{sim} values, each set of aggregation jobs completes within $\sim$15 hours.

Single-linkage hierarchical clustering on Giraph. We apply the algorithm for connected components after pruning user pairs below certain similarity thresholds. SynchroTrap's user matching function (§4.4.3) allows distinct thresholds for similarity on different



(a) Volume of user pairs above similarity thresholds



(b) Execution time of finding connected components

FIGURE 4.16: **The volume of similar user pairs and the execution time of the algorithm of connected components in each application. We set the thresholds in our user matching function to 0.2, 0.4, 0.6, and 0.8. Error bars represent 95% confidence intervals.**

granularities. We use a one-week data set to examine the performance of clustering under varying similarity thresholds. For simplicity we assign the same value to all similarity thresholds and set this value to 0.2, 0.4, 0.6, and 0.8, respectively. Figure 4.16(a) plots the volume of similar user pairs above the thresholds. As we can see, the user matching function prunes the volume of user pairs to tens of gigabytes. The same threshold value can result in different volumes of user pairs remaining in different application contexts, because of the different user action patterns. Figure 4.16(b) shows the execution time of finding connected components on Giraph. Because the connected components algorithm iteratively propagates component IDs until convergence (§4.5.3), the execution time is influenced by the topology of the user similarity graph in each application. As a result, the algorithm may take different time to process the applications even if the volumes of the

input similar-user pairs are close. Within each application, the execution time increases as we set the thresholds to lower values. Due to the efficiency of Giraph [27], SynchroTrap's clustering jobs can finish within $\sim$100 minutes under a wide range of threshold settings.

4.8 Discussion

In this section we discuss several limitations of our approach.

Aggressive attacks. Aggressive attacks could be launched by controlling accounts to perform bulk actions within a short time period. SynchroTrap could miss part of such spiky threats if either the user action-set size or the user-pair similarity does not meet the criteria of SynchroTrap's matching function. However, such attacks have been the focus of existing countermeasures [86], which seek the anomaly of abrupt changes in user activity. SynchroTrap works together with existing anomaly detection schemes and complements them by targeting the stealthier attacks.

Attackers' countermeasures. Attackers could adjust the coordination of the controlled accounts to evade detection. However, because SynchroTrap considers both the service entities and the timing, given a target amount of user actions, attackers have to "spread" user actions over more entities or a longer time period. This requirement increases the cost of resource-constrained attacks (i.e., investing more resources) and reduces the revenue of mission-constrained attacks (i.e., missing deadlines), making them less profitable.

4.9 Summary

In this chapter, we uncover large groups of active malicious accounts in online social networks (OSNs). We do so by leveraging the attackers' economic constraints. To this end, we designed a clustering-based defense system, SynchroTrap, that detects groups of loosely synchronized malicious users. To process the large volume of user activity data in an OSN, SynchroTrap is implemented as an incremental processing system on top of Hadoop and Giraph. We further optimize it using a data partitioning scheme based on overlapping sliding windows.

We deployed SynchroTrap in five applications at Facebook and Instagram. This led to the unveiling of 1165 large campaigns within one month that involved more than two million malicious accounts. Although we designed SynchroTrap for OSNs, it can be used to suppress large attacks in other online services such as web email, electronic commerce, etc. We believe that both the pipeline-processing-based implementation and optimization techniques we have explored are applicable to a wide range of applications that analyze time-dependent data sets.

5

Future Work

In this chapter we discuss our ongoing and future work to further enhance SybilRank and SynchroTrap.

5.1 Improving social graphs for SybilRank

Like many existing social-graph-based schemes, SybilRank relies on the limited social connections between fake accounts and real users to detect fake accounts. As a result, the genuineness of social connections in social graphs has significant influence on the effectiveness of SybilRank. We describe possible ways to sterilize social graphs for SybilRank or more broadly for the social-graph-based schemes.

5.1.1 Distilling strong social connections from reciprocal user interactions

Social connections in a social graph are assumed to represent mutual and strong social trust between users. However, some friendships that users expose on OSNs might not encode sufficient trust as expected. This is because people may add friends casually on OSNs with acquaintances or even strangers, without confirming their identities in the real world. Those weak social connections can impair the effectiveness of SybilRank if they

are obtained by fake accounts.

We noticed another potential robust source of social trust in OSNs: reciprocal user interactions. This additional source of trust could be used to distill strong social connections. The hypothesis is that multiple rounds of interactive activities between a pair of users, such as message exchanging, interactive posting and re-tweeting, etc, usually reveal a close relationship with strong social trust. It is possible that fake accounts initiate interactions to real users, but it is usually difficult to get real users to respond or to respond as frequently as they would to their real-world friends.

Our early thoughts on improving the social graphs involved distilling strong social connections from reciprocal user interactions and using only these strong connections for SybilRank's detection. To do so, one could analyze the user activities delivered through a social edge and prune those edges with few reciprocal user interactions. Statistical and machine-learning-based approaches can be applied to derive a proper decision on each OSN social connection.

5.1.2 Combating friend spam

SybilRank tolerates $O(\log n)$ fake accounts per attack edge, where n is the total number of users. Although this is the best bound offered by existing social-graph-based schemes, the number of undetectable fake accounts is proportional to the number of total attack edges those accounts have managed to obtain. To avoid the detection of social-graph-based approaches, fake accounts may choose to flood friend requests to real users in hope of gaining free social connections, which we call *friend spam*. Those additional social edges obtained through friend spam pollute the social graph and lead to degraded effectiveness of social-graph-based schemes.

A potential approach to mitigate friend spam is to leverage the readily available social rejection on OSNs, which appears in the forms of friend-request rejections and abuse-driven user reports. The high-level idea is that the social connections obtained by fake

accounts are usually accompanied by a large number of social rejections. For example, among the friend requests sent out by fake accounts, a significant fraction of them are rejected by cautious real users. Previous measurement study on RenRen [97] revealed a large gap between the rejection rates of a fake account's requests and a real user's requests. An empirical study also found a significant number of rejections on a set of live fake Facebook accounts [39].

One of our ongoing projects, called Rejecto, aims to exploit this observation to combat friend spam. It is based on the intuition that the large number of inevitable social rejections yields a low aggregate-acceptance ratio of the requests that the fake accounts send out. As a result, Rejecto augments a social graph with directional social rejection edges and uncovers the fake accounts from which friend spam emanates by finding the cut with the minimum aggregate-acceptance ratio.

5.2 Extending SynchroTrap

SynchroTrap is a generic framework to detect the loosely synchronized actions of malicious accounts. It represents a user action with a tuple abstraction: user ID, timestamp, and constraint. A straightforward extension of SynchroTrap would be to apply it to more OSN applications. We deployed it at Facebook and Instagram applications such as page like, application install, user following, photo upload, and login, but SynchroTrap could be applicable to many other applications such as messaging, wall post, video upload, etc. The key step to configuring SynchroTrap for an application is to designate proper action attributes as the constraint field.

To catch resource-constrained attacks, the source IP address of a user action is a good indicator of resource usage, as demonstrated by our deployment. Other useful action attributes include certain fields of browser cookies [21, 29] such as user agent, OS version, etc. Adding those cookie attributes into the constraint field enables detection at a finer

granularity, which in turn provides quality input for generating accurate attack signatures.

To catch mission-constrained attacks, SynchroTrap needs to be configured with a mission-related constraint field. The AssocID of interactive user actions, i.e., the object ID with which a user action interacts, usually reveals the action missions. For instance, the receiver ID of a message can be used to identify message spam that targets a particular set of users.

Fine-grained action missions can be encoded in the payload of user actions, because attackers may rely on the delivered content in the actions they perform to influence users. In applications like messaging, posting, and tweeting, text is the most commonly used medium. The words embedded in such text can reveal details about the missions of campaign attacks. For example, advertising phrases and malicious URLs usually reveal the intent of the malicious users who issue the text. As a result, if we include the payload text of an action into the constraint field, SynchroTrap would be able to directly detect groups of user actions that deliver spamming text. Because text tolerates variance of words and phrases, we would also need to extend SynchroTrap's constraint matching function to allow fuzzy match using text analysis.

5.3 Improving the effectiveness of responses

Whereas most existing efforts focus on effectively uncovering malicious accounts, the responses after account detection also play a significant role in completely throttling such accounts. According our experience, the responses to identified malicious accounts are multifold in practice. Accounts that are being caught for the first time are typically challenged with CAPTCHAs; accounts caught multiple times can be challenged via short message services (SMS), or barred from some OSN features for a certain period of time (e.g., being prohibited from uploading photos for a week); the last response is to shut down accounts. OSNs providers usually select responses according to a multitude of factors such as the time, scale, and the consequences of the attacks.

Although OSN providers always have the option to suspend accounts, they usually conservatively choose to send challenges before shutting down accounts. Challenges such as CAPTCHAs are not as reliable as expected, because people can hire cheap human solvers to resolve them [5]. Facebook has recently enabled photo-based social authentication [13]. This two-factor authentication approach raises the bar of bypassing challenges; unfortunately, people were able to locate its vulnerability [14]. Because attackers often attempt to rescue the detected accounts in order to reduce campaign cost, we advocate more efforts from both industry and the research community to improve response effectiveness.

6

Conclusion

Online social networking services (OSNs) pervade everyday life and are changing the way people connect and communicate. At the same time, the open nature of social networking platforms attracts a constant interest in attacking and exploiting them. Cyber criminals usually manipulate a large number of malicious accounts, including fake accounts and compromised accounts, to launch attacks that affect users for illegal purposes. The sheer volume of active users in OSNs and the diversity of their profiles and behaviors make it challenging to defend against malicious accounts. Although in practice OSN providers employ a large set of countermeasures, they fall short of the increasing attacks in terms of effectiveness, scalability, and robustness.

This dissertation aimed at comprehensively understanding and systematically defending against malicious accounts. It has demonstrated the efficacy of our approach, which leverages the inherent weaknesses on the attacker side to throttle large attacks in OSNs. We focused on two fundamental weaknesses of the attackers: 1) their social relationship constraints, i.e., limited social connections to real users, and 2) their economic constraints, e.g., specific mission goals and limited hosting infrastructure. We used the first weakness to aid the detection of fake accounts in large scale social graphs and the second to un-

cover large groups of active malicious accounts. The deployment of our systems in the real world, i.e., SybilRank at Tuenti and SynchroTrap at Facebook, has demonstrated the practicality of both the design and implementation of our approaches.

6.1 Building a scalable detection scheme using social graphs

We presented SybilRank, a social-graph-based Sybil detection scheme that can scale up to OSNs with hundreds of millions of users. SybilRank is based on the hypothesis that fake accounts often have disproportionally few social connections to real users. It then uses efficiently computable early-terminated random walks on the social graph to distinguish fake accounts. SybilRank starts the random walks from a small set of known real users. Due to the limited social connections between non-Sybil users and Sybils, such random walks have a higher degree-normalized probability of landing at non-Sybil nodes than at Sybil nodes. Consequently, SybilRank uses this landing probability to rank users, which yields a ranked list with a substantial fraction of Sybils at the bottom.

SybilRank has re-formulated the problem of Sybil detection in OSNs as scalable user ranking. This re-formulation was designed in response to our observation that binary Sybil/non-Sybil detectors usually result in high false rates and are computationally expensive for large OSNs. As a result, SybilRank achieves scalability through the random-walk-based user ranking, while enabling an OSN to focus its inspection efforts toward the end of the user-ranked list it generates. It bounds the number of Sybils that are ranked higher than non-Sybil users to $O(\log n)$ per attack edge. With only 11 commodity machines on Amazon EC2, a Hadoop MapReduce prototype of SybilRank can process a social graph with 160 million nodes within 33 hours. Furthermore, SybilRank has been successfully deployed at Tuenti, the largest online social network in Spain. In this deployment, $\sim 90\%$ of the lowest-ranked accounts at Tuenti actually warranted suspension.

6.2 Uncovering large groups of active malicious accounts by exploiting the economic constraints of attackers

We built SynchroTrap, a system that uncovers large groups of active malicious accounts, including both fake accounts and compromised accounts, by detecting their loosely synchronized actions. The design of SynchroTrap is based on the observation that malicious accounts typically perform loosely synchronized actions to accomplish an attack mission, due to the economic constraints an attacker faces. Consequently, SynchroTrap clusters user accounts according to the similarity of their actions and uncovers large groups of malicious accounts that act similarly at around the same time for a sustained period of time.

We implemented SynchroTrap on Hadoop and Giraph as an incremental processing system that can efficiently process massive user activity data in a large OSN. The performance results on a 200-machine cluster showed that it takes a few hours for SynchroTrap to process the daily data and $\sim$ 14 hours to process a weekly aggregation job at Facebook. SynchroTrap has been deployed at Facebook and Instagram, where it was able to unveil millions of malicious accounts and thousands of large attack campaigns per month.

Appendix A

Security Analysis for SybilRank

SybilRank leverages the gap of mixing time between the non-Sybil region and the entire graph and uses short random walks to bound the number of Sybils that rank high. We first present a primer on the mixing time gap, and then provide the proofs for SybilRank's security guarantees.

A.1 A primer on mixing time

Random walks on a graph can be modeled by a Markov chain. The mixing time is defined by the maximum walk length that a random walker should take to approximate the stationary distribution [33]. Due to the disproportionally small number of attack edges that Sybils can obtain, a large mixing time gap exists between the entire graph G and the non-Sybil region G_H. To facilitate this explanation, we introduce the concept of graph conductance $\Phi(G)$ for a graph $G = \langle V, E \rangle$:

$$\Phi(G) = \min_{X \in V} \frac{|\partial X|}{\min\{vol(X), vol(V \backslash X)\}} \tag{A.1}$$

The limited attack edges of Sybils result in a small conductance for the social graph.

The following summarizes how a small graph conductance leads to long mixing time [33]. Denote the transition matrix as the row-normalized adjacency matrix $\mathbf{P}$:

$$p_{uv} = \begin{cases} \frac{1}{deg(u)} & \text{if} (u,v) \in E \\ 0 & \text{else} \end{cases}$$

Suppose vector ρ is the stationary distribution of a random walk defined by the adjacency matrix $\mathbf{P}$. We denote $\Delta(t)$ as the maximal relative error when approximating the elements of the stationary distribution ρ by $p_{uv}^{(t)}$, where $p_{uv}^{(t)}$ is an element of matrix $\mathbf{P}^t$ [33].

$$\Delta(t) = \max_{uv} \frac{|p_{uv}^{(t)} - \rho_v|}{\rho_v}$$

Because $\mathbf{P}^t$ is row-normalized, it has the largest eigenvalue 1. Suppose λ_2 stands for the second largest $|\lambda|$, where λ runs through all eigenvalues of $\mathbf{P}$ ($-1 < \lambda \leq 1$). The following holds according to [33]:

$$\begin{cases} \Delta(t) \leq \frac{{\lambda_2}^t}{\min_j \rho_j} \text{ for all } t \\ \Delta(t) \geq {\lambda_2}^t \text{ for all even } t \end{cases} \tag{A.2}$$

The above equations indicate that the mixing time of graph is bound by λ_2. We fix $\Delta(t)$ to a small variation distance $\bar{\Delta}$ and denote the mixing time as $t(\bar{\Delta})$ with respect to $\bar{\Delta}$. An n-node graph is fast mixing if its mixing time is $O(\log n + \log \bar{\Delta}^{-1})$.

Meanwhile, λ_2 is bound by the graph conductance [33].

$$1 - 2\Phi_G \leq \lambda_2 \leq 1 - \frac{\Phi_G^2}{2} \tag{A.3}$$

As indicated by (A.2) and (A.3), a small graph conductance leads to long mixing time. Therefore, we transform the small graph conductance that Sybils introduce to the long mixing time of the graph under attack. Denote t_G and t_{G_H} as the mixing time of G and G_H respectively. Above observation can be formulated as following: $t_G \gg t_{G_H}$ holds due to $\Phi_G \ll \Phi_{G_H}$.

A.2 Proofs

Lemma 1. *In the* $(i+1)_{th}$ *iteration,* $\forall i \geq 0$*, the expected aggregate trust in the Sybil region increases by an amount of* $(1 - C_H - C_S)^i \times C_H T_G$*.*

Proof: This lemma is proved by induction. We define $T^{(i)}(H)$ and $T^{(i)}(S)$ as the aggregate trust in the non-Sybil user set H and the Sybil set S after the i_{th} iteration.

At the beginning, all trust is initiated in the non-Sybil region, i.e., $T^{(0)}(H) = T_G$ and $T^{(0)}(S) = 0$. During the iterations, the trust is conserved within the entire graph, i.e., for $\forall i \geq 0$, $T^{(i)}(H) + T^{(i)}(S) = T_G$.

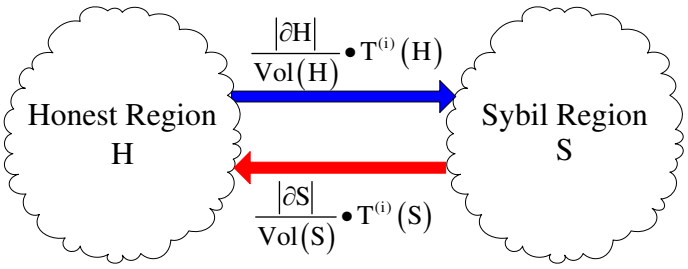


FIGURE A.1: **Trust exchange between the non-Sybil region and the Sybil region in the** $(i+1)_{th}$ **iteration (**$|\partial H| = |\partial S| = g$**)**

In each iteration, trust is re-distributed along edges in the entire network. This analysis assumes that the attack edges are randomly established on both ends. Suppose that we are at the $(i+1)_{th}$ iteration. From the non-Sybil region H, on average each edge carries $\frac{T(i)(H)}{vol(H)}$ trust. It is expected that $|\partial H|$ attack edges pass $\frac{|\partial H|}{vol(H)} T^{(i)}(H)$, i.e. $C_H T^{(i)}(H)$, trust to the Sybil regions. Similarly, in the Sybil region, $C_S T^{(i)}(S)$ trust is expected to flow out to the non-Sybil region. Figure A.1 illustrates the trust dynamics in the $(i+1)_{th}$ iteration. Thus, the following models the trust exchange between two consecutive iterations:

$$\begin{cases} T^{(i+1)}(S) = T^{(i)}(S) + C_H T^{(i)}(H) - C_S T^{(i)}(S) \\ T^{(i+1)}(H) = T^{(i)}(H) + C_S T^{(i)}(S) - C_H T^{(i)}(H) \\ T^{(i+1)}(S) + T^{(i+1)}(H) = T^{(i)}(S) + T^{(i)}(H) = T_G \end{cases}$$

The initial trust vector assigns $\frac{T_G}{K}$ trust to each of the K trust seeds. At the beginning $i = 0$, $C_H T^{(0)}(H) = C_H T_G$ and $C_S T^{(0)}(S) = 0$. We get

$$T^{(1)}(S) - T^{(0)}(S) = C_H T_G \tag{1}$$

This forms the base of this lemma. By induction, assume that $T^{(i)}(S) - T^{(i-1)}(S) = (1 - C_H - C_S)^{i-1} \times C_H T_G$, for all $i \geq 1$. Consider the trust dynamics in the $(i+1)_{th}$ iteration.

$$\begin{aligned}
& T^{(i+1)}(S) - T^{(i)}(S) && (2)\\
=& C_H T^{(i)}(H) - C_S T^{(i)}(S) && (3)\\
=& C_H((1 - C_H)T^{(i-1)}(H) + C_S T^{(i-1)}(S)) \\
& - C_S((1 - C_S)T^{(i-1)}(S) + C_H T^{(i-1)}(H)) && (4)\\
=& (1 - C_H - C_S)(C_H T^{(i-1)}(H) - C_S T^{(i-1)}(S)) && (5)\\
=& (1 - C_H - C_S)(T^{(i)}(S) - T^{(i-1)}(S)) && (6)\\
=& (1 - C_H - C_S)^i \times C_H T_G && (7)
\end{aligned}$$

(4) holds because of $T^{(i)}(S) = T^{(i-1)}(S) + C_H T^{(i-1)}(H) - C_S T^{(i-1)}(S)$ and $T^{(i+1)}(H) = T^{(i)}(H) + C_S T^{(i)}(S) - C_H T^{(i)}(H)$. (7) is derived from the induction base. □

Theorem 3. *Suppose the non-Sybil region G_H is fast-mixing, and the mixing time of the entire graph G is $r \log n$, where r is a large integer. After $w = w_0 \log n$ ($r = kw_0$) power iterations, where w_0 and k are positive integers, Sybils get a disproportionally small amount of aggregate trust. This amount is only a small fraction f of that in the stationary distribution, i.e., $T^{(w)}(S) = f\frac{vol(S)}{2m}T_G$, such that*

$f = \frac{1}{1+(1-\alpha)^{w_0 \log n}+(1-\alpha)^{2w_0 \log n}+\cdots+(1-\alpha)^{(k-1)w_0 \log n}}$

and $\alpha = C_H + C_S$.

Proof: As shown in Lemma 1, the expected aggregate Sybil trust grows monotonously with power iterations. In particular, the following shows the aggregate Sybil trust after

$w = w_0 \log n$ and $r \log n$ iterations, respectively.

$$T^{(w)}(S) = \sum_{0 \le i \le (w_0 \log n - 1)} (1-\alpha)^i C_H T_G \tag{8}$$

$$T^{(r \log n)}(S) = \sum_{0 \le i \le (r \log n - 1)} (1-\alpha)^i C_H T_G \tag{9}$$

Because $r \log n$ iterations approximate the stationary distribution of the entire graph, we have $T^{(r \log n)}(S) \simeq \lim_{t \to \infty} T^{(t)}(S) = \frac{vol(S)}{2m} T_G$. Thus

$$\begin{aligned} T^{(w)}(S) &= \frac{\sum_{0 \le i \le (w_0 \log n - 1)} (1-\alpha)^i}{\sum_{0 \le i \le (r \log n - 1)} (1-\alpha)^i} \frac{vol(S)}{2m} T_G \\ &= f \frac{vol(S)}{2m} T_G \end{aligned} \tag{10}$$

□

Theorem 4. *When an attacker randomly establishes g attack edges in a fast-mixing social network, the total number of Sybils that rank higher than non-Sybils is $O(g \log n)$.*

Proof: It is shown in §3.5.3 that at most $\frac{f}{c} \times vol(S)$ Sybils ranked higher than non-Sybil nodes. We prove Theorem 4 by providing an upper bound on $\frac{f}{c} \times vol(S)$.

The total trust is conserved within the graph after $O(\log n)$ iterations:

$$f \frac{vol(S)}{2m} T_G + c \frac{vol(H)}{2m} T_G = T_G \tag{11}$$

That is,

$$\frac{f}{c} vol(S) = \frac{vol(H)}{\frac{2m}{f vol(S)} - 1} \tag{12}$$

According to Theorem 3, we have

$$\frac{f}{c} vol(S) = \frac{vol(H)}{\frac{T_G}{T^{(w)}(S)} - 1} \tag{13}$$

By Lemma 1 and Corollary 2, we have

$$\begin{aligned} T^{(w)}(S) &= \sum_{0\le i\le(w-1)} (1-C_H-C_S)^i C_H T_G \\ &< \sum_{0\le i\le(w-1)} (1-C_H)^i C_H T_G \\ &= \sum_{0\le i\le(w-1)} ((1-C_H)^i - (1-C_H)^{i+1}) T_G \\ &= (1-(1-C_H)^w) T_G \end{aligned} \tag{14}$$

Combining (13) and (14), we have

$$\frac{f}{c} vol(S) < vol(H)((1-C_H)^{-w}-1) \tag{15}$$

We replace $(1-C_H)^{-w}$ with its Maclaurin series $(1-C_H)^{-w} = 1 + wC_H + o({C_H}^2)$.

$$\begin{aligned} \frac{f}{c} vol(S) &< vol(H)((1-C_H)^{-w}-1) \\ &= vol(H) O(C_H) w \\ &= vol(H) O(C_H) O(\log n) \\ &= O(g \log n) \end{aligned} \tag{16}$$

□

Bibliography

[1] An Explanation of CAPTCHAs. http://www.facebook.com/note.php?note_id=36280205765, 2008.

[2] Facebook connect. https://developers.facebook.com/blog/post/2008/05/09/announcing-facebook-connect/, 2008.

[3] Better Security through Software. http://www.facebook.com/notes/facebook/better-security-through-software/248766257130, 2010.

[4] Fake Accounts in Facebook - How to Counter It. http://tinyurl.com/5w6un9u, 2010.

[5] Re: CAPTCHAs: Understanding CAPTCHA-Solving Services in an Economic Context. In *Usenix Security*, 2010.

[6] Staying in Control of Your Facebook Logins. http://www.facebook.com/notes/facebook/staying-in-control-of-your-facebook-logins/389991097130, 2010.

[7] Stolen Facebook Accounts for Sale. http://tinyurl.com/25cngas, 2010.

[8] Tuenti, Spain's Leading Social Network, Switches on Local for a Location-based Future. http://tinyurl.com/yeygmw5, 2010.

[9] Why the Number of People Creating Fake Accounts and Using Second Identity on Facebook are Increasing. http://tinyurl.com/3uwq75x, 2010.

[10] Working Together to Keep You Secure. http://www.facebook.com/notes/facebook/working-together-to-keep-you-secure/68886667130, 2010.

[11] Google Explores +1 Button To Influence Search Results. http://tinyurl.com/7g927oy, 2011.

[12] Personal communication with the Manager of User Support and the Product Manager of the Core and Community Management teams in Tuenti, 2011.

[13] The Facebook Blog: A Continous Commitment to Security. https://www.facebook.com/blog/blog.php?post=486790652130, 2011.

[14] All your faces belong to us: Breaking Facebook's Social Authentication. http://blogs.computerworld.com/security/21449/all-your-faces-belong-us-breaking-facebooks-social-authentication, 2012.

[15] Facebook: 5-6% of accounts are fake. http://www.zdnet.com/blog/facebook/facebook-5-6-of-accounts-are-fake/10167, 2012.

[16] Stanford Large Network Dataset Collection. http://snap.stanford.edu/data/index.html, 2012.

[17] Tuenti: A Private, Invitation-Only Social Platform Used by Millions of People to Communicate and Share Every Day. http://www.tuenti.com, 2012.

[18] Amazon EC2 Pricing. http://aws.amazon.com/ec2/pricing/, 2013.

[19] Apache Giraph. http://giraph.apache.org/, 2013.

[20] Apache Hadoop. http://hadoop.apache.org/, 2013.

[21] Cookies, Pixels & Similar Technologies. https://www.facebook.com/help/cookies, 2013.

[22] Facebook Reports Fourth Quarter and Full Year 2012 Results. http://investor.fb.com/releasedetail.cfm?ReleaseID=736911, 2013.

[23] Facebook Reports Fourth Quarter and Full Year 2013 Results. http://investor.fb.com/releasedetail.cfm?ReleaseID=821954, 2013.

[24] Facebooks New Way to Combat Child Pornography. http://gadgetwise.blogs.nytimes.com/2011/05/19/facebook-to-combat-child-porn-using-microsofts-technology, 2013.

[25] Fighting Spam at Google. https://www.google.com/insidesearch/howsearchworks/fighting-spam.html, 2013.

[26] Malicious Chrome extensions on the rise. http://www.zdnet.com/malicious-chrome-extensions-on-the-rise-7000019913/, 2013.

[27] Scaling Apache Giraph to a trillion edges. https://www.facebook.com/notes/facebook-engineering/scaling-apache-giraph-to-a-trillion-edges/10151617006153920, 2013.

[28] Twitter Reports Fourth Quarter and Fiscal Year 2013 Results. https://investor.twitterinc.com/releasedetail.cfm?ReleaseID=823321, 2013.

[29] Types of cookies used by Google. http://www.google.com/policies/technologies/types/, 2013.

[30] L. V. Ahn, M. Blum, N. J. Hopper, and J. Langford. CAPTCHA: Using Hard AI Problems for Security. In *Eurocrypt*, 2003.

[31] S. Arora, S. Rao, and U. Vazirani. Expander Flows, Geometric Embeddings and Graph Partitioning. STOC, 2004.

[32] A.-L. Bárabási and R. Albert. Emergence of Scaling in Random Networks. *Science*, 286:509–512, 1999.

[33] E. Behrends. Introduction to Markov chains: with Special Emphasis on Rapid Mixing. In *Advanced Lectures in Mathematics*, 2000.

[34] A. Beutel, W. Xu, V. Guruswami, C. Palow, and C. Faloutsos. CopyCatch: Stopping Group Attacks by Spotting Lockstep Behavior in Social Networks. In *Proceedings of the 22nd International Conference on World Wide Web (WWW)*, 2013.

[35] L. Bilge, T. Strufe, D. Balzarotti, and E. Kirda. All Your Contacts Are Belong to Us: Automated Identity Theft Attacks on Social Networks. In *WWW*, 2009.

[36] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre. Fast Unfolding of Communities in Large Networks. In *Journal of Statistical Mechanics: Theory and Experiment*, 2008.

[37] Q. Cao, M. Sirivianos, X. Yang, and T. Pregueiro. Aiding the Detection of Fake Accounts in Large Scale Social Online Services. Duke University Technical Report, 2011.

[38] Q. Cao, M. Sirivianos, X. Yang, and T. Pregueiro. Aiding the Detection of Fake Accounts in Large Scale Social Online Services. In *NSDI*, 2012.

[39] Q. Cao and X. Yang. SybilFence: Improving Social-Graph-Based Sybil Defense with User Negative Feedback. Duke University Technical Report, http://arxiv.org/abs/1304.3819, 2012.

[40] A. Cheng and E. Friedman. Sybilproof Reputation Mechanisms. In *P2PEcon*, 2005.

[41] P.-A. Chirita, J. Diederich, and W. Nejdl. MailRank: Using Ranking for Spam Detection. In *CIKM*, 2005.

[42] G. Danezis and P. Mittal. SybilInfer: Detecting Sybil Nodes Using Social Networks. In *NDSS*, 2009.

[43] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *OSDI*, 2004.

[44] M. Dell amico and Y. Roudier. A Measurement of Mixing Time in Social Networks. In *5th International Workshop on Security and Trust Management*, 2009.

[45] J. R. Douceur. The Sybil Attack. In *IPTPS*, 2002.

[46] M. Egele, G. Stringhini, C. Krügel, and G. Vigna. Compa: Detecting compromised accounts on social networks. In *NDSS*, 2013.

[47] P. W. L. Fong. Preventing Sybil Attacks by Privilege Attenuation: A Design Principle for Social Network Systems. In *IEEE S&P*, 2011.

[48] H. Gao, J. Hu, C. Wilson, Z. Li, Y. Chen, and B. Y. Zhao. Detecting and Characterizing Social Spam Campaigns. In *IMC*, 2010.

[49] M. Gjoka, M. Kurant, C. T. Butts, and A. Markopoulou. A Walk in Facebook: Uniform Sampling of Users in Online Social Networks. In *IEEE INFOCOM*, 2010.

[50] G. Gu, R. Perdisci, J. Zhang, and W. Lee. BotMiner: Clustering Analysis of Network Traffic for Protocol- and Structure-Independent Botnet Detection. In *USENIX SECURITY*, 2008.

[51] G. Gu, J. Zhang, and W. Lee. BotSniffer: Detecting Botnet Command and Control Channels in Network Traffic. In *NDSS*, 2008.

[52] Z. Gyöngyi, H. Garcia-Molina, and J. Pedersen. Combating Web Spam with TrustRank. In *VLDB*, 2004.

[53] J. A. Hanley and B. J. McNeil. The Meaning and Use of the Area under a Receiver Operating Characteristic (ROC) Curve. *Radiology*, 143, 1982.

[54] S. Hao, N. A. Syed, N. Feamster, A. G. Gray, and S. Krasser. Detecting Spammers with SNARE: Spatio-Temporal Network-Level Automatic Reputation Engine. In *USENIX SECURITY*, 2009.

[55] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl. Evaluating Collaborative Filtering Recommender Systems. *ACM Trans. Inf. Syst.*, 22, 2004.

[56] P. Jaccard. The Distribution of the Flora in the Alpine Zone. *New Phytologist*, 11(2), 1912.

[57] T. N. Jagatic, N. A. Johnson, M. Jakobsson, and F. Menczer. Social Phishing. *Communications of the ACM*, 50(10), 2007.

[58] A. K. Jain, M. N. Murty, and P. J. Flynn. Data Clustering: a Review. *ACM Computing Surveys*, 31(3), 1999.

[59] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina. The EigenTrust Algorithm for Reputation Management in P2P Networks. In *WWW*, 2003.

[60] U. Kang, C. E. Tsourakakis, and C. Faloutsos. PEGASUS: Mining Peta-Scale Graphs. *Knowl. Inf. Syst.*, 27(2), 2011.

[61] C. Kanich, C. Kreibich, K. Levchenko, B. Enright, G. M. Voelker, V. Paxson, and S. Savage. Spamalytics: An Empirical Analysis of Spam Marketing Conversion. In *ACM CCS*, 2008.

[62] D. R. Karger. Random Sampling in Cut, Flow, and Network Design Problems. STOC, 1994.

[63] G. Karypis and V. Kumar. Multilevel Algorithms for Multi-Constraint Graph Partitioning. In *Proceedings of the 1998 ACM/IEEE conference on Supercomputing*, 1998.

[64] H.-P. Kriegel, P. Kröger, and A. Zimek. Clustering High-Dimensional Data: A Survey on Subspace Clustering, Pattern-Based Clustering, and Correlation Clustering. *ACM Trans. Knowl. Discov. Data*, 3(1), 2009.

[65] A. N. Langville and C. D. Meyer. Deeper Inside Pagerank. *Internet Mathematics*, 1:2004, 2004.

[66] J. Leskovec and C. Faloutsos. Sampling from Large Graphs. In *ACM SIGKDD*, 2006.

[67] C. Lesniewski-Laas and M. F. Kaashoek. Whanau: A Sybil-proof Distributed Hash Table. In *NSDI*, 2010.

[68] J. Lin and A. Kolcz. Large-scale Machine Learning at Twitter. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, SIGMOD '12, 2012.

[69] J. Lin and M. Schatz. Design patterns for efficient graph algorithms in mapreduce. MLG, 2010.

[70] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. Pregel: a System for Large-Scale Graph Processing. In *SIGMOD*, 2010.

[71] A. Mislove, A. Post, P. Druschel, and K. P. Gummadi. Ostra: Leveraging Social Networks to Thwart Unwanted Traffic. In *NSDI*, 2008.

[72] A. Mislove, B. Viswanath, K. P. Gummadi, and P. Druschel. You are Who You Know: Inferring User Profiles in Online Social Networks. In *ACM WSDM*, 2010.

[73] M. Mitzenmacher and E. Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005.

[74] A. Mohaisen, N. Hopper, and Y. Kim. Keep Your Friends Close: Incorporating Trust into Social-Network-based Sybil Defenses. In *INFOCOM*, 2011.

[75] A. Mohaisen, A. Yun, and Y. Kim. Measuring the Mixing Time of Social Graphs. In *ACM IMC*, 2010.

[76] M. Mondal, B. Viswanath, A. Clement, P. Druschel, K. P. Gummadi, A. Mislove, and A. Post. Limiting Large-scale Crawls of Social Networking Sites. In *SIGCOMM Poster Session*, 2011.

[77] D. Moore, C. Shannon, G. M. Voelker, and S. Savage. Internet Quarantine: Requirements for Containing Self-Propagating Code. In *INFOCOM*, 2003.

[78] M. Motoyama, D. McCoy, K. Levchenko, G. M. Voelker, and S. Savage. Dirty Jobs: The Role of Freelance Labor in Web Service Abuse. In *Proceedings of the USENIX Security Symposium*, 2011.

[79] A. Nazir, S. Raza, C.-N. Chuah, and B. Schipper. Ghostbusting Facebook: Detecting and Characterizing Phantom Profiles in Online Social Gaming Applications. In *WOSN*, 2010.

[80] R. Nishtala, H. Fugal, S. Grimm, M. Kwiatkowski, H. Lee, H. C. Li, R. McElroy, M. Paleczny, D. Peek, P. Saab, D. Stafford, T. Tung, and V. Venkataramani. Scaling Memcache at Facebook. In *NSDI*, 2013.

[81] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank Citation Ranking: Bringing Order to the Web. Technical report, Stanford, 1999.

[82] A. Post, V. Shah, and A. Mislove. Bazaar: Strengthening User Reputations in Online Marketplaces. In *USENIX NSDI*, 2011.

[83] Z. Qian, Z. M. Mao, Y. Xie, and F. Yu. On Network-level Clusters for Spam Detection. In *NDSS*, 2010.

[84] D. Quercia and S. Hailes. Sybil Attacks Against Mobile Users: Friends and Foes to the Rescue. In *INFOCOM*, 2010.

[85] P. Resnick and R. Sami. Sybilproof Transitive Trust Protocols. In *ACM Electronic Commerce Conference*, 2009.

[86] T. Stein, E. Chen, and K. Mangla. Facebook immune system. In *Proceedings of the 4th Workshop on Social Network Systems*, SNS, 2011.

[87] K. Thomas, C. Grier, V. Paxson, and D. Song. Suspended Accounts in Retrospect: An Analysis of Twitter Spam. In *IMC*, 2011.

[88] K. Thomas, D. McCoy, C. Grier, A. Kolcz, and V. Paxson. Trafficking Fraudulent Accounts: The Role of the Underground Market in Twitter Spam and Abuse. In *USENIX SECURITY*, 2013.

[89] A. Thusoo, Z. Shao, S. Anthony, D. Borthakur, N. Jain, J. Sen Sarma, R. Murthy, and H. Liu. Data Warehousing and Analytics Infrastructure at Facebook. In *SIGMOD*, 2010.

[90] D. N. Tran, B. Min, J. Li, and L. Subramanian. Sybil-Resilient Online Content Rating. In *NSDI*, 2009.

[91] N. Tran, J. Li, L. Subramanian, and S. S. Chow. Optimal Sybil-resilient Node Admission Control. In *INFOCOM*, 2011.

[92] C. E. Tsourakakis, C. Gkantsidis, B. Radunovic, and M. Vojnovic. Fennel: Streaming Graph Partitioning for Massive Scale Graphs. Microsoft Technical Report MSR-TR-2012-113, 2012.

[93] B. Viswanath, A. Post, K. P. Gummadi, and A. Mislove. An Analysis of Social Network-based Sybil Defenses. In *ACM SIGCOMM*, 2010.

[94] G. Wang, T. Konolige, C. Wilson, X. Wang, H. Zheng, and B. Y. Zhao. You are How You Click: Clickstream Analysis for Sybil Detection. In *USENIX SECURITY*, 2013.

[95] Y. Xie, F. Yu, K. Achan, E. Gillum, M. Goldszmidt, and T. Wobber. How Dynamic Are IP Addresses? In *SIGCOMM*, 2007.

[96] Y. Xie, F. Yu, K. Achan, R. Panigrahy, G. Hulten, and I. Osipkov. Spamming Botnets: Signatures and Characteristics. In *SIGCOMM*, 2008.

[97] Z. Yang, C. Wilson, X. Wang, T. Gao, B. Y. Zhao, and Y. Dai. Uncovering Social Network Sybils in the Wild. In *IMC*, 2011.

[98] H. Yu. Sybil Defenses via Social Networks: A Tutorial and Survey. In *ACM SIGACT News, Distributed Computing Column*, 2011.

[99] H. Yu, P. Gibbons, M. Kaminsky, and F. Xiao. SybilLimit: A Near-Optimal Social Network Defense Against Sybil Attacks. In *IEEE S&P*, 2008.

[100] H. Yu, M. Kaminsky, P. B. Gibbons, and A. Flaxman. SybilGuard: Defending Against Sybil Attacks via Social Networks. In *SIGCOMM*, 2006.

[101] H. Yu, C. Shi, M. Kaminsky, P. B. Gibbons, and F. Xiao. DSybil: Optimal Sybil-Resistance for Recommendation Systems. In *IEEE S&P*, 2009.

[102] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica. Resilient Distributed Datasets: A Fault-tolerant Abstraction for In-Memory Cluster Computing. In *NSDI*, 2012.

[103] Y. Zhang, J. Wang, Y. Wang, and L. Zhou. Parallel Community Detection on Large Networks with Propinquity Dynamics. KDD, 2009.

[104] Y. Zhao, Y. Xie, F. Yu, Q. Ke, Y. Yu, Y. Chen, and E. Gillum. BotGraph: Large Scale Spamming Botnet Detection. In *NSDI*, 2009.

[105] C. C. Zou, W. Gong, and D. Towsley. Worm Propagation Modeling and Analysis Under Dynamic Quarantine Defense. In *Proceedings of the 2003 ACM workshop on rapid malcode (WORM)*, 2003.

Biography

Qiang Cao was born in Yiyang, Hunan, China on August 22nd, 1985. He defended his PhD thesis at Duke University in April 2014. His research interests include security, computer networking, and large-data processing. He received an M.S. in Computer Science from Duke University in 2012. Before that, he received a B.S. and an M.S. both in Computer Science from Wuhan University, China.