Toward Attack-Resistant Distributed Information Systems by Means of Social Trust

by

Michael Sirivianos

Department of Computer Science
Duke University

Date: _____

Approved:

_____
Xiaowei Yang, Advisor

_____
Jeffrey S. Chase

_____
Landon Cox

_____
Michael K. Reiter

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
2010

ABSTRACT

(Computer Science)

Toward Attack-Resistant Distributed Information Systems by Means of Social Trust

by

Michael Sirivianos

Department of Computer Science
Duke University

Date: _____

Approved:

_____
Xiaowei Yang, Advisor

_____
Jeffrey S. Chase

_____
Landon Cox

_____
Michael K. Reiter

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
2010

Abstract

Trust has played a central role in the design of open distributed systems that span distinct administrative domains. When components of a distributed system can assess the trustworthiness of their peers, they are in a better position to interact with them. There are numerous examples of distributed systems that employ trust inference techniques to regulate the interactions of their components including peer-to-peer file sharing systems, web site and email server reputation services and web search engines.

The recent rise in popularity of Online Social Networking (OSN) services has made an additional dimension of trust readily available to system designers: *social trust*. By social trust, we refer to the trust information embedded in social links as annotated by users of an OSN. This thesis' overarching contribution is methods for employing social trust embedded in OSNs to solve two distinct and significant problems in distributed information systems.

The first system proposed in this thesis assesses the ability of OSN users to correctly classify online identity assertions. The second system assesses the ability of OSN users to correctly configure devices that classify spamming hosts. In both systems, an OSN user explicitly ascribes to his friends a value that reflects how trustworthy he considers their classifications. In addition, both solutions compare the classification input of friends to obtain a more accurate measure of their pairwise trust. Our solutions also exploit trust transitivity over the social network to assign trust values to the OSN users. These values are used to weigh the classification input by each user in order to derive an aggregate trust

score for the identity assertions or the hosts.

In particular, the first problem involves the assessment of the veracity of assertions on identity attributes made by online users. Anonymity is one of the main virtues of the Internet. It protects privacy and freedom of speech, but makes it hard to assess the veracity of assertions made by online users concerning their identity attributes (e.g, age or profession.) We propose FaceTrust, the first system that uses OSN services to provide lightweight identity credentials while preserving a user's anonymity. FaceTrust employs a "game with a purpose" design to elicit the opinions of the friends of a user about the user's self-claimed identity attributes, and uses attack-resistant trust inference to compute veracity scores for the attributes. FaceTrust then provides credentials, which a user can use to corroborate his online identity assertions.

We evaluated FaceTrust using a crawled social network graph as well as a real-world deployment. The results show that our veracity scores strongly correlate with the ground truth, even when a large fraction of the social network users are dishonest. For example, in our simulation over the sample social graph, when 50% of users were dishonest and each user employed 1000 Sybils, the false assertions obtained approximately only 10% of the veracity score of the true assertions. We have derived the following lessons from the design and deployment of FaceTrust: a) it is plausible to obtain a relatively reliable measure of the veracity of identity assertions by relying on the friends of the user that made the assertion to classify them, and by employing social trust to determine the trustworthiness of the classifications; b) it is plausible to employ trust inference over the social graph to effectively mitigate Sybil attacks; c) users tend to mostly correctly classify their friends' identity assertions.

The second problem in which we apply social trust involves assessing the trustworthiness of reporters (detectors) of spamming hosts in a collaborative spam mitigation system. Spam mitigation can be broadly classified into two main approaches: a) centralized security infrastructures that rely on a limited number of trusted monitors (reporters) to detect

and report malicious traffic; and b) highly distributed systems that leverage the experiences of multiple nodes within distinct trust domains. The first approach offers limited threat coverage and slow response times, and it is often proprietary. The second approach is not widely adopted, partly due to the lack of assurances regarding the trustworthiness of the reporters.

Our proposal, SocialFilter, aims to achieve the trustworthiness of centralized security services and the wide coverage, responsiveness, and inexpensiveness of large-scale collaborative spam mitigation. It enables nodes with no email classification functionality to query the network on whether a host is a spammer. SocialFilter employs trust inference to weigh the reports concerning spamming hosts that collaborating reporters submit to the system. To the best of our knowledge, it is the first collaborative threat mitigation system that assesses the trustworthiness of the reporters by both auditing their reports and by leveraging the social network of the reporters' human administrators. Subsequently, SocialFilter weighs the spam reports according to the trustworthiness of their reporters to derive a measure of the system's belief that a host is a spammer.

We performed a simulation-based evaluation of SocialFilter, which indicates its potential: during a simulated spam campaign, SocialFilter classified correctly 99% of spam, while yielding no false positives. The design and evaluation of SocialFilter offered us the following lessons: a) it is plausible to introduce Sybil-resilient OSN-based trust inference mechanisms to improve the reliability and the attack-resilience of collaborative spam mitigation; b) using social links to obtain the trustworthiness of reports concerning spammers (spammer reports) can result in comparable spam-blocking effectiveness with approaches that use social links to rate-limit spam (e.g., Ostra [64]); c) unlike Ostra, SocialFilter yields no false positives. We believe that the design lessons from SocialFilter are applicable to other collaborative entity classification systems.

To my beloved wife, Vicky

Contents

List of Tables

List of Figures

List of Abbreviations and Symbols

Abbreviations

WWW	World Wide Web
OSN	Online Social Network
SMTP	Simple Mail Transfer Protocol
TCP	Transport Control Protocol
IU	Identity Uniqueness
DHT	Distributed Hash Table
IDS	Intrusion Detection System
CS	Computer Science

Acknowledgements

This is a most likely incomplete attempt to acknowledge the armies of people that have helped me reach the end of the road in the long journey to a Ph.D degree.

First and foremost, I am grateful to my advisor, Xiaowei Yang. To say that Xiaowei's advice and support has been instrumental in my academic voyage would be an understatement. She was there to guide me when the research seemed fuzzy and disheartening. She has shown me how to take an important and complex problem, and divide it in small manageable pieces that can be addressed in a innovative and practical way. Even more, she has taught me the importance of clarity of ideas and of concise technical speech.

I especially thank Michael Reiter, Landon Cox and Jeffrey Chase for their valuable guidance as members of my thesis committee. Their pressing questions during the preliminary and defense exams have helped to improve this work and to make me a better presenter and researcher. Many thanks also go to Bruce Maggs and Angelos Stavrou for taking the time to help me polish my job talk.

I am grateful to all my collaborators during my Ph.D studies. This long list includes, but is not limited, to Michael Goodrich, Athina Markopoulou, Jong Han Park, Jian Wei Gan, John Solis, Frederik Armknecht, Xiaoyuan Yang, Mario A. Sanchez, Rex Chen and Claudio Soriente. Special thanks go to Stanislaw Jarecki and Gene Tsudik for their help and advice during my first two years at UCI. I am indebted to Kyungbaek Kim for all his hard work on the research presented in this thesis. I am also grateful to Dirk Westhoff and Joao Girao for putting up with a fresh Ph.D student, while I was an intern at NEC's

Heidelberg labs. Many thanks go to all the anonymous reviewers, Nikitas Liogkas, and Eddie Kohler whose feedback has helped us improve our work, and to our ToN editor [79], Dan Rubenstein.

I express my gratitude to Pablo Rodriguez and Nikolaos Laoutaris for offering me the opportunity to work with them at Telefonica Research. Niko thanks so much for being such an inspiring supervisor and a good friend, and helping me navigate Barcelona. I am indebted to Chen Li for his inspiring mentoring when I was a TA for his classes and his generous recognition of my efforts. Special thanks go to Keith Marzullo and Geoffrey Voelker for supervising my Master's thesis.

This research has been supported by several NSF, USENIX, and ACM grants. I thank the department of Computer Science at the Universities of California, San Diego and Irvine for the many learning and teaching opportunities they offered me. I am grateful to the Cyprus Fulbright Committee for enabling me to come to the US for Master's studies. I also express my gratitude to the administrative personnel at UCSD, UCI and Duke for providing me with all the resources I needed and for ensuring my seamless transitions through all three phases of my graduate studies.

The long trip towards the Ph.D, would be a miserable and longer (or abruptly shorter) one if it was not for the people who have stood by me. My wife, Vicky, has been my pillar all those years. She has offered me love, comfort and assurance when things did not look as promising and she has been there to enjoy the better times with me. To my mother, Andromache, my father, Sirivianos, and brother, Petros, words cannot express how thankful I am to you for helping me evolve from an infant to a productive member of the society.

I am more indebted to my friends and cousins than the Greek government is to its lenders. To Fragkiskos Papadopoulos, Minas Gjoka, Ersin Uzun, Karim el Defrawy, I owe an elaborate support system that includes long conversations about "what's best in life" (WBL,) and excellent personal and academic help and advise. Many times I told myself

that if it was not for Fragkiskos' example, I would not be in this academic path. Spyros, Yianna, Panayiotis, Dhruv, Vlad, Luca, Mayur, Herodotos, Marios, Vijay, John, Vaggelis, Andreas, Maria, Hande, Ines, Kalliroe, Ioanna, Younsun, Peyman, Alpha, Angelos, Ivelin, Nedyalko, and Marianna thanks so much for filling my days with pure fun, for opening my eyes about various aspects of success, and for even more WBL and WIO conversations. Eduardo, Pablo, and Peter I promise to keep maintaining the "dailytroll" to pay you back for your awesomeness. Mike and Jolanda thanks for having me as your roommate last summer.

Finally, Xin, Ang, Mish, Qiang, Yanbin, and Amre, thank you all for being such great lab mates and collaborators. I learned a lot from you during our long discussions on research and life. I wish you all the best in your pursuit of greatness in the field of Computer Science.

1

Introduction

1.1 Overview

The Internet has enabled the deployment of a multitude of complex distributed computer systems that allow information sharing among devices and their human users. Prominent examples are the World Wide Web (WWW), the email infrastructure and peer-to-peer content distribution networks such as BitTorrent [36].

Components of a distributed system need to be able to assess whether the information shared and the actions taken by other components are reliable and have no malicious intend. Consequently, trust has always played a central role in the design of open distributed systems that span distinct administrative domains. For example, a misbehaving peer in a file sharing system may be serving content that does not correspond to the content's description. A malicious node in a Distributed Hash Table (DHT) [48] may drop or misroute queries.

Open distributed systems are also known to be vulnerable to the Sybil attack [39]. Sybil attacks subvert distributed systems by introducing numerous malicious identities under the control of an adversary. By using these identities the adversary acquires disproportional

influence over the system. For example, attackers can use Sybils to control $\frac{1}{3}$ or more of nodes in a Byzantine agreement [57] setting.

Consequently, there are numerous examples of distributed systems that employ algorithmic trust inference to regulate the interactions of their components. By *trust inference*, we refer to algorithms that derive trust values for each vertex in a trust graph. In these trust graphs, the vertices encode the components of the distributed system. The edges encode how much trust each component places on other components. The trust encoded by edges can be derived either by mining explicit trust assignments by each component concerning another (*e.g.*, a vote or a hyperlink,) or by comparing the actions of two components (*e.g.*, voting similarity.) A list of the components that vertices in the trust graph may encode includes but is not limited to: a) nodes or files in a peer-to-peer file sharing system [53, 87]; b) web sites [12, 11, 49]; c) recommended items [67, 27]; d) email domains or IP addresses [4, 17]; and e) a public key certificate in the PGP Web of Trust [99].

However, trust inference, as it is currently employed in most distributed systems, faces two shortcomings: a) it is itself susceptible to Sybil attacks [34, 35]; b) it does not exploit the social relationships the users of those systems may have, resulting in reduced resilience to malicious behavior.

Recently, a new application of the World Wide Web called Online Social Networks (OSN) has risen in popularity and is now rivaling the most popular web search engine [9] in terms of visitors. Social networking sites such as Facebook and LinkedIn have amassed over 400 million and 60 million users as of May 2010. Unlike most other web services that are structured around content, OSN services are structured around their users and their users' relationships. OSN services enable users to declare who their social acquaintances are and organize them in lists of stratified privileges. They also allow users to submit and share multimedia content, and communicate with their friends in the form of messages, status updates, content tags, and comments.

The broad availability and popularity of massive OSN services has made an additional

dimension of trust readily available to system designers: *social trust*. By social trust, we refer to the trust information embedded in social links as annotated by users of an OSN. The overarching contribution of this thesis is our proposed use of social trust that is inferable from OSN services. In particular, we use it to enhance existing trust inference techniques by rendering them more resilient to manipulation. We then use the enhanced trust inference to address two significant problems in distributed systems: a) assessing the veracity of identity assertions (statements) made by online users; b) assessing the trustworthiness of reporters (detectors) of spamming hosts in a collaborative spam mitigation system. To solve the first problem this thesis proposes FaceTrust, and to solve the second problem this thesis proposes SocialFilter.

Both FaceTrust and SocialFilter exploit social trust as follows: the vertices of the social graph encode components that classify other components. These components are online users for FaceTrust, and spam reporters managed by human administrators for SocialFilter. The edges encode user-defined and similarity-derived trust between socially-acquainted components. We use trust inference to determine the weight we should assign to the classifications of the users or spam reporters in quantifying the veracity of an identity assertion or the belief that a host is spamming, respectively. This is in contrast to previous OSN-based approaches, that used trust inference to determine whether a user is allowed to communicate over the social network (by sending an email or voting on an online item) [64, 85].

1.1.1 Rationale of Employing Online Social Networks

In designing both FaceTrust and SocialFilter, we treat trust and manipulation-resistance as first-order design goals. These goals lead us to structure our design around OSN services for the following two reasons:

- Establishing friend relationships requires substantial efforts by human users. Thus establishing edges in the social graph can be used as a resource test to defeat Sybils.

To this end, both systems exploit trust transitivity and social trust in their Sybil-resilient trust-inference methods.

- The intuitive user interface of OSN services allows users to easily manage their social acquaintances. Our designs augment the OSN user interface with an additional user interaction that seamlessly integrates with the existing OSN user experience: enabling users to declare how much they trust each friend's actions and statements.

1.1.2 Assessing the Veracity of Online Identity Assertions

The first problem in which we apply social trust involves the assessment of the veracity of assertions on identity attributes made by online users. Anonymity is one of the main virtues of the Internet. It protects privacy and freedom of speech, but makes it hard to assess the veracity of assertions made by online users concerning their identity attributes (e.g, age or profession.) We propose FaceTrust, the first system that uses OSN services to provide lightweight identity credentials while preserving a user's anonymity. FaceTrust employs a "game with a purpose" design to elicit the opinions of the friends of a user about the user's self-claimed identity attributes, and uses attack-resistant trust inference to compute veracity scores for the attributes. FaceTrust then provides credentials, which a user can use to corroborate his online identity assertions.

We evaluated FaceTrust using a crawled social network graph as well as a real-world deployment. The results show that our veracity scores strongly correlate with the ground truth, even when a large fraction of the social network users are dishonest. For example, in our simulation over the sample social graph, when 50% of users were dishonest and each user employed 1000 Sybils, the false assertions obtained approximately only 10% of the veracity score of the true assertions. We have derived the following lessons from the design and deployment of FaceTrust: a) it is plausible to obtain a relatively reliable measure of the veracity of identity assertions by relying on the friends of the user that made the assertion

to classify them, and by employing social trust to determine the trustworthiness of the classifications; b) it is plausible to rely trust inference over the social graph to effectively mitigate Sybil attacks; c) users tend to mostly correctly classify their friends' identity assertions.

FaceTrust bears similarity with the PGP Web of Trust [99] in that the edges of the trust graph encode social relationships. Furthermore, similar to the PGP Web of Trust, FaceTrust relies on the friends of a user to certify his identity. We justify this design choice in Sections 2.3.1 and 3.3.1.

1.1.3 Introducing Social Trust to Collaborative Spam Mitigation

Spam mitigation can be broadly classified into two main approaches: a) centralized security infrastructures that rely on a limited number of trusted monitors (reporters) to detect and report malicious traffic; and b) highly distributed collaborative systems that leverage the experiences of multiple nodes within distinct trust domains. The first approach offers limited threat coverage and slow response times, and it is often proprietary. The second approach is not widely adopted, partly due to the lack of assurances regarding the trustworthiness of the reporters. Hence, the second problem in which we apply social trust involves assessing the trustworthiness of reporters (detectors) of spammers in a collaborative spam mitigation system.

Our proposal, SocialFilter, aims to achieve the trustworthiness of centralized security services and the wide coverage, responsiveness, and inexpensiveness of large-scale collaborative spam mitigation. It enables nodes with no email classification functionality to query the network on whether a host is a spammer. SocialFilter employs trust inference to weigh the reports concerning spamming hosts that collaborating reporters submit to the system. To the best of our knowledge, it is the first collaborative threat mitigation system that assesses the trustworthiness of the reporters by both auditing their reports and by leveraging the social network of the reporters' human administrators. Subsequently,

SocialFilter weighs the spam reports according to the trustworthiness of their reporters to derive a measure of the system's belief that a host is a spammer.

We performed a simulation-based evaluation of SocialFilter, which indicates its potential: during a simulated spam campaign, SocialFilter classified correctly 99% of spam, while yielding no false positives. The design and evaluation of SocialFilter offered us the following lessons: a) it is plausible to introduce Sybil-resilient OSN-based trust inference mechanisms to improve the reliability and the attack-resilience of collaborative spam mitigation; b) using social links to obtain the trustworthiness of reports that identify spammers (spammer reports) can result in spam-blocking effectiveness that is comparable to approaches that use social links to rate-limit spam (e.g., Ostra [64]); c) unlike Ostra, SocialFilter yields no false positives. We believe that the design lessons from SocialFilter are applicable to other collaborative entity classification systems.

1.1.4 Rationale for our Choice of Trust Inference Methods

For the problem of assessing the veracity of online identity assertions, we design a new a max-flow-based trust inference method, which is computationally efficient ($O(|E|\log|V|)$). However the trust values it returns cannot be directly interpreted as a belief that an assertion is true. Therefore it is not informative and actionable at the absence of prior experience with those values. These values however, correlate strongly with the ground truth: a true assertion will have a higher veracity than a false assertion. Thus, a user or service can interpret them in a meaningful way if it has prior experience with the system and has set appropriate trust thresholds.

For the problem of assessing the trustworthiness of spammer reports, we rely on a commonly used [87] maximum trust path method. That method by itself is not Sybil-resilient, hence we combine it with a Sybil detection method [92]. This approach is more computationally expensive ($O(|V|\sqrt{|E|}\log|V| + |E|\log|V|)$), but its trust values have a partially Bayesian interpretation. That is, they can be used to derive a trust value that can

be explicitly interpreted as the belief that a host is spamming: a host with 0% *spammer belief* is very unlikely to be a spammer, whereas a host with 100% spammer belief is very likely to be one.

The rationale behind our choice to use two district trust inference methods for each problem lies in the trade-off between computational cost and how informative the trust values can be. In the context of spam mitigation the users for which we assess the trustworthiness are relatively limited in numbers: the number of email servers is less than the number of Online Social Networking users. Thus, we can afford to use a more expensive trust inference method that can be directly interpreted as a belief. However, when the number of users is very high, it is preferable to use a non-Bayesian but less informative metric.

1.1.5 Contributions

The first contribution of this thesis is the lessons from the design and evaluation of FaceTrust. FaceTrust is the first system that uses social networks to provide lightweight identity credentials without sacrificing a user's anonymity. In particular, this thesis makes the following sub-contributions with respect to the problem of assessing the veracity of identity assertions made by online personas:

- We have designed an attack-resistant veracity scoring mechanism that relies on the friends of a user to classify (tag) his online assertions on his identity attributes.
- We have designed a new context-specific credential scheme to obviate the need for client-side cryptography to improve usability.
- We have improved upon a prior max-flow trust inference method [60] by making it more attack resistant and computationally efficient.
- We have implemented the design, and evaluated it using simulations and a real-world deployment. Our results show the plausibility of leveraging friend tagging

on identity assertions and social trust to assess the trustworthiness of the tags. Our results also illustrate the limitations of our approach, namely its inability to ensure the correctness of veracity scores when attackers launch highly organized colluding and Sybil attacks.

The second contribution of this thesis is the lessons from the design and evaluation of SocialFilter. To the best of our knowledge, SocialFilter is the first OSN-based collaborative spam filtering system to use Sybil-resilient trust inference to assess the overall trustworthiness of a node's spammer reports. Our sub-contributions with respect to the problem of assessing the trustworthiness of spammer reports in a collaborative spam mitigation system are:

- We have combined existing trust inference mechanisms to derive an attack-resistant mechanism that assigns trust values (spammer belief) to hosts detected by a collaborative spam mitigation system.
- We have evaluated our design using simulations and compared it to Ostra [64] to illustrate our design's advantages and shortcomings.
- We have demonstrated the plausibility of using social trust to improve the reliability and attack-resilience of collaborative spam mitigation systems. We believe that our mechanisms are applicable to other collaborative entity classification systems.

2

FaceTrust: Assessing the Veracity of Online Identity Assertions via Social Networks

In this chapter, we describe how we use Online Social Networks to assess the veracity of identity assertions made by online users.

2.1 Introduction

The Internet has changed how people interact with each other. Rich social interactions take place online. Users read, shop, chat, work, and play on the Internet. However, unlike social interactions in the physical world, the Internet has largely hidden the identity of online users. "On the Internet, nobody knows you are a dog," says the famous Peter Steiner cartoon.

While anonymity has brought much benefit, including protecting user privacy and free speech, it also poses security threats to online activities, and makes what and who to believe challenging. On one hand, individuals that hide their real identity attributes may defraud naive users; on the other hand, honest users cannot make credible assertions about some of their identity attributes without sacrificing their anonymity. Consider these real

examples:

- Alice is shopping for a food scale and she finds a rave review "Worth DOUBLE the Money" [20] from a user claiming "I was a chef for many years." Should she believe it, given that she is aware authors or users with vested interests in a company have been caught creating fake positive reviews for their own books [22] or the company's products [2]?

- John met "Kayla" on Facebook, and sent her nude photos. "Kayla" was actually Anthony Stancl [23]. John was blackmailed and sexually assaulted.

Real-world remedies to this problem typically forgo a user's anonymity. Moreover, verifying a user's identity attributes can be costly and time consuming. For instance, Amazon provides a Real Name badge to a user that desires to sign the content he posts by his real name [21]. It verifies a user's name using his credit card information. Twitter is experimenting with a feature called Verified Account [86] to prevent identity confusion. It currently provides verified accounts only to celebrities, because of the cost and time required to verify an account [86].

This situation prompts the question: *can an online user establish identity veracity without sacrificing his anonymity?* One approach is to use personal digital certificates issued by a trusted authority (*e.g.*, VeriSign), and apply techniques such as *idemix* [33] to make the certificates anonymous, unlinkable, and non-transferable. However, we consider this approach heavyweight and inflexible. It involves centralized manual verification, and could be a financial [19] as well as a usability burden to users [89]. Each time a user desires to certify a new attribute (*e.g.*, a new home address), he would have to contact the trusted authority.

In Sections 2.2 and 2.3, we present FaceTrust, a system that enables online personas to cost-effectively obtain credentials that provide an additional indication of the trustworthiness of identity assertions. Our insight is that many realistic Internet settings do not

require strong authentication to guard critical resources. Instead, they may benefit greatly from partial or likely-to-be-true identity information. For example, a user may only need to know a reviewer's profession rather than his real name to trust his reviews. Similarly, it may suffice for a user of an online dating service to know that another user's location information is likely to be true.

FaceTrust mines and enriches information embedded in online social networks (OSNs) to provide lightweight and extensible digital credentials. We observe that OSNs already allow users to express a limited form of trust relationships using friend links. We propose to extend this ability by allowing users to declare whether they consider the identity assertions of their friends credible (Section 2.3.1).

In particular, a user that wishes to obtain a credential posts short assertions about himself on his OSN profile in the form of a poll, *e.g.*, "Am I really over 18?" The user asks his friends to respond to this poll by tagging (classifying) his assertion as `true` or `false`. The OSN employs an attack-resistant trust inference algorithm to compute a veracity score for this assertion, which we refer to as the veracity of the assertion (Section 2.3.2.) The OSN then issues to the user a credential in the form of {`assertion, veracity, context`} (Section 2.3.4.) Verifiers (online services or human users) can use this OSN-issued credential to regulate their interactions with the user. To improve usability, we design a context-specific credential scheme that allows a user to certify his online statements with a web interface without involving user-side cryptography (Section 2.3.4.)

The key challenge of FaceTrust lies in assessing the veracity of user identity assertions. It should be easy for trustworthy (honest) users to obtain high veracity for their true assertions, while it should be relatively difficult for untrustworthy (dishonest) users to make their false assertions have high veracity. Users may post false assertions, tag incorrectly or lie, and create Sybil accounts [39]. To address this challenge, we translate this problem into trust inference (Section 2.3.3.) The intuition is that honest users tend to tag correctly and similarly. We compare a user's tags with those from his friends on the set of asser-

tions they both tag, and use the similarity between the tags as trust values on social graph edges. We then compute a Sybil-resistant *tagger trustworthiness* score for each user, using a max-flow-based trust inference scheme. Finally, we derive an assertion's veracity by combining its tags weighted by their tagger's trustworthiness.

FaceTrust credentials can be used by verifiers to reject assertions with low veracity scores from the outset. Verifiers can treat credentials with high veracity as an additional indication of the trustworthiness of the user that presents the credential, but they should not take them at face value. For this reason, we refer to our credentials as relaxed.

We evaluate FaceTrust using simulations on a crawled 200K-user social network graph and a real-world deployment on Facebook (Section 2.6.) In our simulations, we randomly mark users as honest or dishonest. Honest users submit only true assertions and tags. Dishonest users strategize to make their false assertions appear credible. Our simulations show that FaceTrust is able to assign high veracity to true assertions and low veracity to false ones. This holds under an adversarial model of moderate strength (2.2.4,) even when a large fraction of the network is dishonest and employs the Sybil attack (Section 2.6.1.) For example, in our simulation over the sample social graph, when 50% of users were dishonest and each user employed 1000 Sybils, the false assertions obtained approximately only 10% of the veracity score of the true assertions.

For the real-world deployment, we developed the "Am I Really?" Facebook application, which uses a game-with-a-purpose design to collect data on real-world user assertion posting and tagging behavior.[1] We find that users of our application tag and post assertions sufficiently frequently to allow the system to function reliably, and that their tags are mostly correct (Section 2.6.2.)

[1] We have obtained Institutional Review Board (IRB) authorization to perform our study.

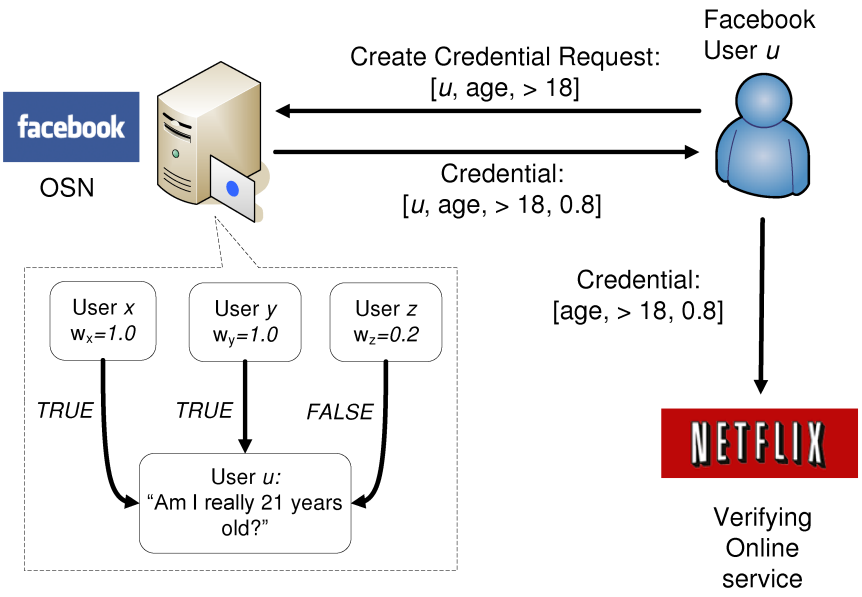

FIGURE 2.1: FaceTrust overview and an age verification example.

2.2 Overview

Figure 2.1 presents an overview of FaceTrust. In FaceTrust, the following three components interact with each other: a) the OSN provider that maintains the social network and its users' profiles, and performs veracity computations; b) online users that maintain accounts with the OSN and attempt to access online services by presenting OSN-issued credentials; and c) verifying online services or users that regulate access to their resources or characterize user inputs based on a user's credentials.

2.2.1 An Example

Before describing FaceTrust in detail, we use an age-verification example to shed light on how components interact. As shown in Figure 2.1, user u attempts to access an age-restricted movie at the Netflix website. At the same time, u is concerned with his anonymity and does not wish to reveal neither his real identity nor a linkable pseudonym to Netflix.

With FaceTrust, Netflix can demand an OSN-issued age credential from the user to

allow access to its content. To obtain this credential, the user u must have posted an age assertion on his OSN profile, and requested his friends to tag the veracity of his age assertion before he attempts to access the age-restricted content. In this example, user u has asserted that his age is 21, and three of his friends, users x, y, and z, have tagged the assertion with boolean values $TRUE$, $TRUE$, and $FALSE$ respectively. Since not all users are equally credible, the OSN provider has computed a trustworthiness score (w) for each tagger x, y, and z by analyzing the social graph and their tagging history as we soon describe in Section 2.3.3. The OSN provider computes an overall veracity score for user u's age assertion (0.8 in this example) by aggregating u's friends' tagged values weighted by their trustworthiness scores (Section 2.3.2).

As shown in Figure 2.1, the OSN issues an age credential with an overall veracity score that certifies that the user belongs to the restricted age group, and the user presents this credential to a Netflix software process to gain access to Netflix content. For ease of use, instead of using cryptographic digital certificates, FaceTrust implements web-based identity credentials using an XML API for online services such as Netflix, or through a simple web page for human users (Section 2.3.4.)

2.2.2 More Motivating Examples

In addition to age verification, we envision that FaceTrust credentials will benefit Internet users and online services in many other ways. A few more examples include but are not limited to:

Assessing the authority or relevance of online reviews or ratings with profession credentials: Many Internet users read online reviews before making purchase decisions. Intuitively, expert opinions of an online product may appear more authoritative to many readers. For instance, a reader may place more weight on a review for a networking textbook by a computer science professor than by an average user. With FaceTrust, if an expert user desires to appear more authoritative, he may request a profession credential from his

OSN provider and present this credential to an online review site when submitting his reviews.

We clarify at this point that the goal of FaceTrust is not to explicitly assess the correctness of the review. Instead, FaceTrust only aims at verifying the identity attribute (profession) of the user that posted the review. This is because FaceTrust is suitable for assessing the veracity of ground truth statements and not for assessing statements that are subject to personal taste or opinion.

Verifying participant eligibility: A citizen journalism site [18] may wish to verify that a user actually resides in a specified area before it accepts his report on an event that took place there. Similarly, Wikipedia, online auction sites, and online statistical surveys may wish to restrict participants to certain groups of people, such as people with particular expertise, residents in a certain geographic area, or people of a certain age range. FaceTrust can assist legitimate participants to obtain credentials that certify their eligibility.

Preventing online fraud: Scammers commonly respond to online postings alleging to be prospective participants in legitimate transactions (*e.g.*, a potential tenant of an apartment), and aiming to commit "advance-fee" fraud [13, 5]. Such attacks could possibly be averted if online users have a way have scammers were unable to lie about their location, affiliation, or age. To this end, a classifieds service such as Craigslist or its users could use FaceTrust to verify identity attributes of users that post or respond to ads.

2.2.3 Assumptions

In designing FaceTrust, we make the following assumptions:

Trustworthy users tag mostly truthfully and similarly, as well as post true identity assertions: We assume that trustworthy (honest) users tag assertions mostly correctly. We validate this assumption in §2.6.2. We also assume that when they tag the same assertion, their tags mostly match, since an assertion in our design is about ground truth rather than personal taste. We employ a weighted voting mechanism (§2.3.2) to characterize

assertions, which allows for a minority of trustworthy users to tag incorrectly. We also assume that trustworthy users mostly post true assertions. We treat users that consistently tag mistakenly not due to malicious intent, but due to lack of knowledge, as *dishonest* (Section 2.2.4.)

Users carefully vet OSN friend requests: This assumption implies that establishing friend connections in the FaceTrust social network is a resource-intensive task, as it may require that two users have actually met each other. Therefore, Sybil attackers cannot cheaply establish such connections as we assume that users would reject friend requests from users they have never met. In addition, users can use common-secret-based techniques to verify that a friend request originates from a real acquaintance and not an imposter [29]. Our assumption also implies that a user is careful to select only friends that will not try to harm him by tagging his true assertions as `false`.

Trusted credential issuing authority: We assume that the OSN provider reliably issues credentials to the best of its knowledge based on the input of its users. We also assume that the OSN provider does not reveal a user's tags to the assertion poster or others. Furthermore, users may wish to remain anonymous and untraceable by the verifiers. We assume that the OSN provider protects the privacy of its users by not revealing their identity and the list of online services or users that verify its users' credentials.

The FaceTrust administrators have an initial estimate of what portion of the social network is honest: We assume that FaceTrust administrators know approximately the portion of users that is honest. They use this estimate to initialize the trust inference method employed by FaceTrust.

The FaceTrust administrators have a' priori knowledge of fully trusted users in the social network: These users (trusted seeds) are used to seed the trust inference method that we employ. These fully trusted users can for example be FaceTrust employees. Fully trusted users can also be inferred by running the trust inference method initialized with FaceTrust employees to identify other highly trustworthy users. These highly trustworthy

users can then be added to the set of fully trusted users and be used to seed the trust inference method.

2.2.4 Threat Model

FaceTrust's design copes with the following threat model:

Dishonest assertion posters and taggers: Our threat model considers dishonest users that are primarily interested in posting false assertions to misrepresent their identities. These dishonest users may collude with other dishonest users that tag their false assertions as `true`.

Sybil Taggers: Dishonest users may launch the Sybil attack [39] by creating many fake accounts under their control. A dishonest user that creates Sybils can employ them to tag its false assertions or the dishonest assertions of the creator's colluders as `true`.

Sybil Assertion Posters: Dishonest users can create Sybils who post false assertions. These assertions are subsequently tagged by their creators and their colluders as `true`. The purpose of this attack is to create users that are connected only to dishonest users, thus their assertions are never tagged `false` by honest users. As a result, it is easier for those Sybils to make their assertions appear credible.

Camouflage attack: We also consider attacks under which attackers initially behave honestly to accumulate trust for themselves or their friends. They later attempt to defeat the system. This threat model resembles what Kamvar et al. [53] refer to as "malicious nodes with camouflage."

One manifestation of this attack is the *tagger camouflage attack*. Dishonest users attempt to build up trust with honest users by always tagging according to the observed majority, *i.e.*, voting according to the veracity that is currently displayed for the assertion. After they earn enough tagger trustworthiness, they tag dishonestly only for specific questions. Thus, we are also explicitly considering the case in which a user is not always tagging falsely, but instead has a mixed strategic behavior.

Another manifestation of the camouflage attack is the *assertion poster camouflage attack*. A dishonest user posts several true assertions. Both his honest and dishonest friends tag those assertions as `true`. As a result, if his dishonest and honest friends are also friends with each other, his dishonest friends build up trust with his honest friends. Consequently the dishonest friends' tagger trustworthiness increases.

Rational dishonest users: We assume that dishonest users, who post false assertions and tag dishonestly are rational. Dishonest users benefit by obtaining a credential that makes a false assertion appear true. At the same time, they incur a cost every time they create a Sybil account, i.e., the time needed to solve CAPTCHAs at registration time. They also incur a cost every time they coordinate with other dishonest users on which assertions to tag as `true`.

Verifiers can collude to survey users: We assume that credential verifiers may wish to track a user against its will. They may collude to link user accounts and derive a more accurate profile of a user's activities.

2.2.5 Goals

FaceTrust's design is driven by the following goals:

Attack-resistant: Our system aims at making it difficult for false assertions to appear trustworthy by having high veracity. The system should be resilient to errors made by benign users and to manipulations by dishonest users. Although our design is attack-mitigating, it cannot ensure the correctness of the veracity scores in the presence of devoted adversaries. Therefore, our system is not meant for guarding critical resources and the veracity scoring is relaxed to be within the range $[0, 1]$ rather than a binary `true` or `false` value. FaceTrust credentials alone cannot guarantee the truthfulness of a statement. For this reason we refer to our credentials as *relaxed*.

Informative: The actual goal of FaceTrust is to provide users with additional information on a statement's veracity, thus making it easier for them to reject false statements. Since

it is relatively easy for a user that posts a truthful assertion to find friends that vouch for him, if an assertion has low veracity, users should reject it from the outset. If however the assertion has high veracity, users should still employ other common sense verification mechanisms or be aware that they are taking a risk by accepting the statement as true.

The veracity of user assertions should correlate positively with the ground truth. When an assertion has higher veracity than another, this should indicate that it is more likely to be credible. Verifiers can define their own thresholds (possibly suggested by FaceTrust's admins) to map a veracity score to an action based on the application scenarios and their own risk tolerance.

Lightweight: We aim to provide credible identity information for online personas without centralized manual identity verification.

Flexible: Users should be able to obtain credentials on a variety of attributes, *e.g.*, age, location, profession, education etc. Users should also be able to conveniently obtain new credentials when their attributes change.

Practical: The system should be easy to use. It should not impose on users the burden of dealing with cryptographic primitives, shared secrets etc. It should require minimal upgrades of client software.

Secure: The credentials should satisfy authentication, *i.e.*, the verifier should be assured that a credential is issued by a trusted authority. Second, they should satisfy integrity, *i.e.*, the assertion, veracity, and context fields of the credential should not change once the credential is issued. This guarantees that a user cannot forge the veracity score of his assertions and that a user cannot use somebody else's assertions to verify his identity attributes. Third, the credentials should satisfy privacy and anonymity, *i.e.*, a user should be able to obtain credentials with no personally identifiable information and should be able to make the showings of his credentials untraceable.

Table 2.1: Key notations.

Name	Meaning
A_i^t	Assertion of type t posted by user j
w_j^t	Tagger trustworthiness of user j for type t
d_{ji}^A	User j's friend i's tag on j's assertion A
a_A	Veracity of assertion A
ts_{ij}^t	Tagging similarity between users i and j for assertions of type t
$T(V,E_t)$	Similarity-based trust graph for type t
S	Set of trusted sources (seeds)
p_d	Portion of the social graph that is dishonest

2.3 FaceTrust Design

We now describe our design in detail. Table 2.1 lists the key notations used in our description. For a more in-depth analysis of the assurances offered by FaceTrust's mechanisms, we refer the reader to 2.4.

2.3.1 Social Tagging

FaceTrust uses *social tagging* to obtain trustworthy identity information for online users. By social tagging, we refer to OSN users posting assertions about their attributes and their friends tagging them as `true` or `false`.

We take an approach similar to the PGP Web of Trust [99], in that we allow only friends of a user to tag (certify) the user's assertion. The rationale behind this choice is two-fold. First, most of the assertions posted by a user can only be reliably evaluated by people who know him. Those people are more likely to be his friends. Second, our system relies on the assumption that since a user has carefully vetted his friends, those friends will mostly not attempt to harm him by tagging his true assertions as `false`.

FaceTrust categorizes user attribute assertions into various types such as age, address, profession, expertise etc. A user posts his assertions of assorted types in his OSN profile.

For instance, for the type age, an assertion has the format `[{<,=,>}, number]`, *e.g.*, `[> 18]` means that the user claims to be older than 18. For the type location, the assertion has the format `[{country,state,city ...}, string]`.

For an assertion A_i^t of type t posted by a user i, i's friend j may tag it as d_{ji}^A. d_{ji}^A takes two values: a) `true` indicates that j believes i's assertion;and b) `false`, indicates that j does not believe it. A posted assertion and the associated tags are valid for a period of time set by the OSN provider depending on the assertion type. An assertion is uniquely identified by its {`type, assertion`} pair. A user cannot repost the same assertion and reset unfavorable tags before the assertion expires. The tags are stored by the OSN provider and are only known to the OSN and the taggers. They are never made available to other users, as they represent sensitive information.

In addition, the OSN provider does not reveal to users the veracity of an assertion unless the assertion has accumulated a threshold number of tags to protect taggers' privacy.

This design assumes that users are willing to tag their friends. This is a reasonable assumption because abundant evidence suggests that users may adopt social tagging. For example, "Friend Facts" [10], a micro-polling application that resembles our Facebook application "Am I Really?", also presents a user with questions about his friends and asks him to vote to let them know what he thinks about them (Figure 2.3.1.) It has amassed ~ 4.5 million monthly active users. We further validate this assumption in Section 2.6.2 using data from our real-world deployment of the "Am I Really?" Facebook application (Section 2.5.)

2.3.2 Assertion Veracity

A main challenge in FaceTrust's design is to assess the veracity of user assertions. This task is difficult because dishonest users may post false assertions and strategize to make them appear true, and benign users may make mistakes. To make this task tractable, we resort to providing a relaxed credential that binds an assertion to a veracity score between

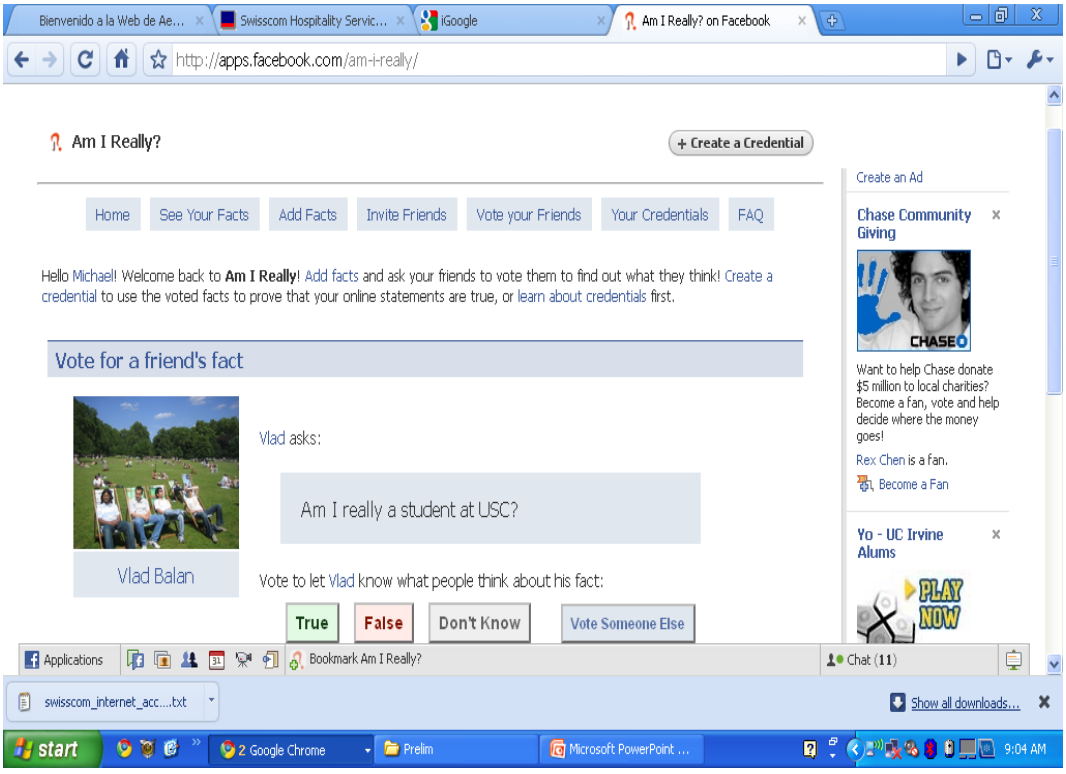

FIGURE 2.2*:* Illustration of social tagging using the "Am I Really?" Facebook application.

0 and 1.

Definition 2.3.1. *We define* assertion veracity *as a score* $0 \leq a_A \leq 1$ *that indicates whether a ground truth (non-subjective) assertion A is true or false. This score strongly correlates with the truth, i.e., an assertion with higher veracity than another is more likely to be true.*

As shown in Figure 2.3, the inputs for computing an assertion's veracity are the tags on the assertion and their taggers' trustworthiness. A tagger j's trustworthiness w_j^t is a metric that estimates the trustworthiness of j's tags on assertions of type t. We compute this metric using the trust inference technique described in Section 2.3.3. We then weight an assertion's tags with their taggers' trustworthiness to compute the assertion's veracity. Let F_i denote the set of friends of user i that have tagged the assertion A_i^t of type t posted by user i. To compute the veracity score a_A of A_i^t , the OSN provider aggregates the tags d_{ji}^A by i's friends as follows:

$$a_A = \max(\sum_{j \in F_i} w_j^t \cdot d_{ji}^A / \sum_{j \in F_i} w_j^t,\ 0) \tag{2.1}$$

We make the scoring of the veracity of an assertion conservative by assigning -1 to `false` tags, 1 to `true` tags, and normalizing negative veracity scores to zero. For instance, if an assertion has two tags `true` and `false` from two equally trustworthy taggers, its assertion veracity will be 0, not 0.5. Equation 2.1 ensures that if the sum of the weights w_j^t of the dishonest users that have falsely tagged user i's false assertion is less than 0.75 of the sum of the weight of the users that have tagged the assertion, the assertion will have less than 0.5 veracity.

This design is biased towards making it difficult for dishonest users to make false assertions appear true, as an honest `false` tag weights more than a dishonest true tag. Dishonest users, however, can abuse this design to make a user's true assertions appear not credible. We are tackling this attack, by allowing only a user's friends to tag his assertions. We assume that the user could use his due diligence to choose honest friends.

It is possible to map `false` tags to 0 and combine tags in a different way such as weighing them by the logarithm of their taggers' trustworthiness. We adopt Equation 2.1 because our experiments show it works the best among the variations we have tried.

We use an additional condition that if the sum of the trustworthiness of the assertion A_i^t's taggers is below a specified threshold M, a_A is 0. M can be proportional to the mean tagger trustworthiness of users. We use this condition to discount assertions that have been tagged only by a few users with low tagger trustworthiness.

$$a_A = 0 \quad \text{if} \quad \textstyle\sum_{j \in F_i} w_j^t < M \tag{2.2}$$

We analyze the assurances provided by the assertion veracity mechanism in 2.4.1.

2.3.3 Tagger Trustworthiness

How can FaceTrust reliably determine a tagger's trustworthiness w^t on assertions of type t? We observe that the problem is similar to determining the trustworthiness of a user and thus resort to *trust inference* techniques. A trust inference algorithm refers to the process of computing the trustworthiness of a node in a trust graph by exploiting the transitivity of trust. The trust algorithm assumes that a few select nodes in the graph are fully trustworthy. It then analyzes the trust graph to determine how this trust propagates to other nodes.

Definition 2.3.2. *We define* tagger trustworthiness *of a user j as the score $0 \leq w^t_j \leq T_{max}$ that indicates whether a tagger j is honest or correct in his assessments of the veracity of assertions of a specific type. This score strongly correlates with the ground truth, i.e., a tag by a user with higher trustworthiness is more likely to correspond to the reality. T_{max} takes integer values.*

We face two challenges in determining the tagger trustworthiness w^t. First, trust inference uses a *trust graph*, where an edge between two users i and j is explicitly labeled with the degree of trust that i places on j. However, this explicit trust information is not available in a social network graph. Second, how should we compute the tagger trustworthiness w^t, given that different trust inference algorithms exist and each has its own strengths? We describe how we address each challenge in turn in Section 2.3.3 and Section 2.3.3.

Tagging Similarity

We address the first challenge by using tagging similarity between two friends to approximate explicit trust. Recall that our assumption is that honest users tend to tag similarly, and note that tagging similarity is transitive. The tagging similarity ts^t_{ij} between two friends i and j for an assertion type t is computed from two sources: a history-defined similarity hs^t_{ij} computed by comparing their tagging history, and a user-defined similarity us^t_{ij}.

We compute the history-defined similarity between two friends using a formula that

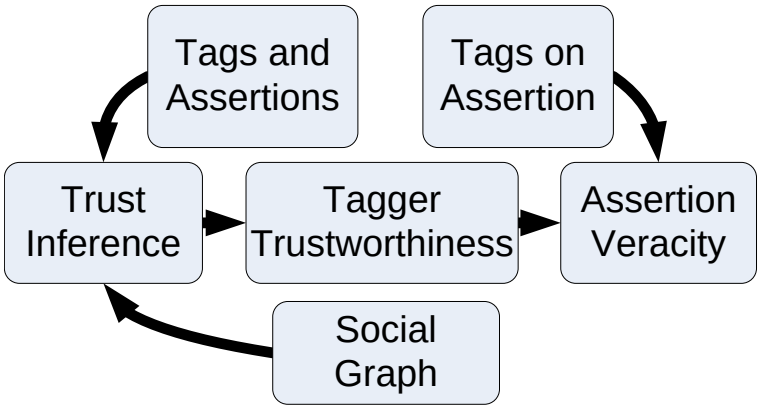

FIGURE 2.3*:* Combining social tagging with trust inference to derive the veracity of user assertions.

resembles the Jaccard index [52]. Let N_t be the total number of assertions of type t that friends i and j both have tagged. Let C_t be the number of tags on the set of common assertions for which i and j are in agreement. The history similarity hs_{ij}^t between i and j for type t is computed as $hs_{ij}^t = C_t/N_t$. If $N_t = 0$, the similarity is equal to 0.

We use special assertions for each type - "Do I honestly tag the <type> assertions of my friends?" - to derive user-defined similarity us_{ij}^t by a user i on his friend j for type t. Each user j posts these assertions on his OSN profile. If j's friend i tags it as `true`, us_{ij}^t equals 1; otherwise, it is 0.

We combine user-defined similarity with history-defined similarity to obtain the final tagging similarity between two users: $ts_{ij}^t \leftarrow a \cdot hs_{ij}^t + (1-a)us_{ij}^t$, where $0 \leq a \leq 1$. We vary the parameter a depending on how many common assertions N_t of the same type t that users i and j have tagged: the larger N_t is, the higher a should be. This is, when N_t is large, we presume that the history-defined similarity $hs_{ij} = C_t/N_t$ approximates the likelihood that two friends would tag an assertion with the same value in the future more accurately than a manually specified value us_{ij}. However, when N_t is small, we use the user-defined value us_{ij} to approximate this likelihood. The parameter a is computed using the logistic (S-shaped) function $a(N_t)$ of N_t:

$$a(N_t) = (1 + e^{b-N_t})^{-1} \tag{2.3}$$

where b is a small constant. For example, if we set $b = 5$, for $N_t \leq 2$, $a(N_t)$ would be small. However, when N_t exceeds the threshold 3, a increases drastically until it becomes 0.5 for $N_t = 5$. For $N_t = 10$, $a(N_t)$ approximates 1.0.

We then transform the social graph into a trust graph by assigning the tagging similarity ts^t_{ij} to be the weight of a trust graph edge from a friend i to a friend j. We refer to this augmented graph as the similarity-based trust graph $T(V, E_t)$. Note that this is a directed graph, as the user-defined similarity us_{ij} is directional.

We have a distinct similarity-based trust graph for each type of assertion to mitigate camouflage attacks (Section 2.2.4). Due to this design an attacker is forced to tag honestly many assertions of the same type in order to boost its tagger trustworthiness. As a result, he is less flexible in his choice of which assertions to tag and how.

We only compute tagging similarity between two friends to constrain the edges in the trust graph to be the same as the ones in the social graph. This design choice is critical in making our assertion veracity scoring algorithm resilient to Sybil attacks, as social edges are limited resources [92]. However, it raises the concern that if two friends do not share common friends, they may not have enough common assertions to tag. Fortunately, OSN measurement studies [63, 25] show that the clustering coefficient in social graphs is one to five orders of magnitude higher than in Erdös-Rényi [42] random graphs and preferential-attachment-constructed random power-law graphs [30]. This result implies that two friends of a single user are more likely to be friends as well.

Trust Inference

Once we have converted a social graph into a trust graph, the challenge is how to compute a tagger's trustworthiness using trust inference techniques. We consider the max-flow-based broad class of trust inference algorithms [60, 61, 73, 34, 85]. It has been shown that max-flow-based trust inference is more resilient to attackers because it considers multiple trust paths [74, 34]. It has also been shown that a group max-flow-based trust metric [60]

is sum-Sybilproof, i.e., an attacker cannot substantially increase the sum of the trust values of users under his control by introducing many Sybils.

The common element among trust inference methods is that trust flows from a few select *trusted seeds* and propagates to the other users in the trust graph. A seed is a highly trusted user, *e.g.*, a trusted employee of the OSN provider that also verifies and tags assertions of many of his acquaintances. The specifics of the trust inference method determine how trust propagates in the graph. Trust inference should assign high trust to users that are well-connected with the trusted seeds, and assign low trust to Sybil users created by dishonest users. A desirable feature that renders a trust metric Sybil-resilient is the *bottleneck property* [60], which we define as follows: "the trust that flows to the region of the graph that consists of dishonest users and their Sybils is limited by the edges connecting the dishonest region with the region that consists of trusted seeds and honest users."

In addition, the selection of the trusted seeds and the number of trusted seeds is paramount to the attack resilience of the system. This is because an attacker that manages to befriend trusted seeds and to build up high tagging similarity with them can greatly manipulate trust assignment. When the trust inference method employs numerous trusted seeds a dishonest user would need to identify and target many of them in order to be effective. Note that the complete trust graph itself is not made public, therefore locating a trusted seed is a difficult task for attackers. Nevertheless, it is possible for dishonest users to infer a portion of the topology and identify a trusted seed. Hence, one additional desirable feature of trust inference methods is that they have to be computationally efficient for numerous trusted seeds. To this end, the method's computation cost should be mostly independent of the number of trusted seeds.

We also note that determining which users in a social graph can be designated as trusted seeds is an important challenge. Gyongyi et al. [49] addressed this challenge in the context of TrustRank, an eigenvector-based trust inference method for web pages. Their solution also applies in our setting. The basic idea is to first run the trust inference method with

a few manually selected seeds. We then identify the nodes with the highest tagger trustworthiness and add them to the set of trusted seeds. Subsequently, repeat until we have a sufficiently large number of seeds with which we can initialize the algorithm.

Max-flow-based Tagger Trustworthiness

We now describe how we compute the tagger trustworthiness w_i^t using a max-flow based trust inference algorithm, inspired by the Advogato [60] trust metric. We call our trust inference method MaxTrust. Both Advogato and MaxTrust satisfy the *bottleneck property*. That is, they provide the assurance that the sum of the tagger trustworthiness of the dishonest user and its Sybils cannot exceed the sum of the capacity of the Sybil creator's incoming edges in the similarity-based flow graph. We further analyze the assurances provided by MaxTrust in Section 2.4.2.

Advogato determines the set of users that can be trusted by at least a certain level w on a trust graph, where a directed edge $u \rightarrow v$ indicates that user u trusts v by at least w. Advogato transforms the trust inference problem into a problem of maximum flow from a *single trusted seed user* to a virtual supersink user. The capacity of the users in the flow graph is distributed such that the sum of the capacity of users at the same shortest hop distance from the trusted seed is approximately equal to the capacity of the seed. It splits each user into two virtual users (+ and −) and draws an additional edge of capacity 1 from the + virtual user to the supersink. The capacity of the edge connecting the + to the − virtual trusted seed user is approximately the number of users in the trust graph that are expected to be trusted by at least x. The sum of the capacity of the +→− edges at each shortest hop distance from the trusted seed is approximately equal to the capacity of the trusted seed +→− edge. A user is considered trusted by at least w if the maximum flow solution has flow 1 on his +→supersink edge. Since Advogato chooses a single trusted seed as the source of its max-flow computation, a dishonest user that is close to the seed can have high capacity. As a result, it can have many of its Sybils accepted at the same

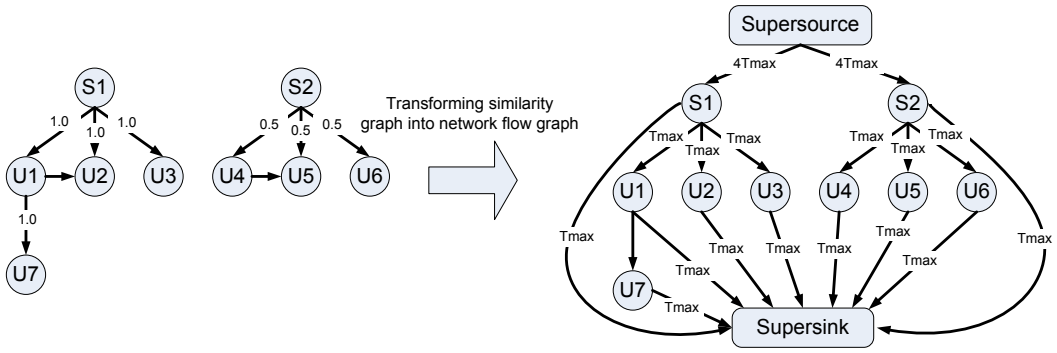

FIGURE 2.4*:* The tagging similarity-based trust graph and its conversion into a MaxTrust network flow graph. The capacity $C_{supersource}$ in the flow graph is $8 \cdot T_{max}$. MaxTrust results in all users except $U7$ having tagger trustworthiness equal to T_{max}.

trust level as him. To mitigate this problem, one has to use multiple trusted seeds from a set $S \subset V$, run the Advogato max-flow computation $|S|$ times, and average the resulting trust value across the runs.

Compared to Advogato, MaxTrust has the advantage that it does not need to be run for each trusted seed. Instead in a single run (max-flow computation), it considers all the trusted seed users. This results in MaxTrust being $\Theta(|S|)$ times more efficient than Advogato.

To assign tagger trustworthiness to a user using Advogato we need to run a max-flow computation for each non-zero trust level w, pruning at each run the edges that correspond to pairwise trust less than w. Instead, MaxTrust computes the tagger trustworthiness $0 \leq w_i^t \leq T_{max}$ in one run, but the max-flow computation is approximately T_{max} times more expensive than the max-flow computation of Advogato for a single trust level. In choosing a value for T_{max} one has to consider the trade-off between computation cost and fine-granularity in assigning trust values to users.

We next describe MaxTrust, which proceeds in two phases: a) the phase in which we transform the trust graph into a network flow problem; and b) a heuristic suitable for approximating max flow in case every user in the graph is directly connected to the sink.

Both Advogato and MaxTrust can be efficiently computed with this heuristic.

Phase 1: Network Flow Graph Creation. In this phase we transform the trust graph into an edge-capacitated maximum flow graph. In the next phase we will use the max-flow on this trust graph to determine the trustworthiness of taggers. Figure 2.4 describes this transformation.

We first create an additional virtual supersource user. We add an edge from the supersource to each trusted user $s \in S$. We also add a directed edge from each user, except of the supersource, to an additional virtual supersink user. To prevent loops during the distribution of capacity among the users (described next,) we prune all edges that connect users at a higher distance from the supersource to users at a lower distance from the supersource. We also prune edges between users at the same distance from the supersource.

We now describe how we distribute capacity to the edges of the network flow graph. In the rest of this description we denote as $C_{supersource}$ the sum of the capacity of the outgoing edges of the supersource. We set $C_{supersource} = (1 - p_d)|V| \cdot T_{max}$, where p_d is the portion of users in the trust graph $G(V,E)$ that are dishonest. We make the implicit assumption that we know the approximate number of honest users at the time we initialize the trust inference method. Next, we assign capacity $C_s = C_{supersource}/|S|$ to each edge from the supersource to each trusted user s. In the rest of this description, we denote as C_u the sum of the capacity of the incoming edges of user u.

Subsequently, we recursively assign capacities to the rest of the edges in the trust graph. That is, for each user u, we distribute $C_u - T_{max}$ capacity among the outgoing edges that connect u with its neighbors in the pruned graph. The capacity C_{uv} of the outgoing edge from user u to its neighbor v in the pruned graph is assigned proportionally to the tagging similarity ts_{uv} between user u and v: $C_{uv} = (C_u - T_{max})\frac{ts_{uv}}{\sum_{z \in F_u} ts_{uz}}$, where F_u is the set of u's friends. We also assign capacity T_{max} to the edge $u \rightarrow supersink$. If $C_u < T_{max}$, we set $C_u = T_{max}$, and allocate no more capacity to u's neighbors. With this choice, we

bias tagger trustworthiness towards higher values for a smaller number of users, instead of higher values for a larger number of users. This further limits the effectiveness of Sybil assertion poster attacks 2.2.4.

Phase 2: Max-flow Computation We now describe how we compute the maximum flow from the supersource to the supersink. In our setting, edge capacity and flows take integer values. Thus, solving optimal max-flow with Edmonds-Karp costs $O(T_{max}(1-p_d)|V||E|)$ as it takes at most $C_{supersource} = T_{max}(1-p_d)|V|$ augmentations. This is computationally prohibitive (Section 2.6.3), therefore we resort to an efficient heuristic.

The heuristic executes T_{max} Breadth First Search (BFS) operations. The BFS operation starts from the supersource. It *visits* every user i in the flow graph once. When the heuristic visits a user i, it *scans* i's children in a random order. For each child, it stores the last parent user that the BFS operation visited before scanning the child. In the rest of this description, we denote the last visited parent of a scanned user j as $parent(j)$.

When the BFS operation scans i's child j, it backtracks from j to the supersource through i as follows. At first, it checks whether the edge $i \rightarrow j$ has at least capacity 1. If yes, it checks whether the capacity of the edge $parent(i) \rightarrow i$ is at least 1. If yes, it sets $i = parent(i)$ and repeats until $parent(i)$ is the supersource. If backtracking reaches the supersource, it adds 1 unit of flow to the edge $j \rightarrow supersink$. It also reduces the capacity of the edges along the backtracking path by 1 unit. If the edges on the backtracking path upstream of i do not have at least capacity 1, the algorithm does not scan any more of i's children. This step costs $O(\Delta)$, where Δ is the graph diameter.

If the algorithm adds 1 unit of flow to the edge $j \rightarrow supersink$, j is considered for a subsequent visit, but is not considered for a subsequent scan by the same BFS operation. If the algorithm does not add 1 unit of flow, j can be scanned from another parent. The BFS operation continues until there are no more users to be visited.

After the BFS operation ends, the heuristic starts a new one from the supersource. It

repeats this process until T_{max} BFS operations are executed. The capacities and flows of the edges remain as adjusted during the previous BFS operation. After T_{max} BFS operations, the flow on the edge $j \rightarrow supersink$ corresponds to j's tagger trustworthiness.

The algorithm performs a total of $\Theta(T_{max}|E|)$ user scans. At each scan it performs $O(\Delta)$ capacity updates for each of the user's ancestors. Thus, our heuristic costs $O(T_{max}|E|\Delta)$. The diameter of social graphs (small world networks) is typically $O(\log(|V|)$ (measured to be 9 to 27 in real OSNs [63].)

Our heuristic takes advantage of the fact that all users are connected to the supersink. Thus, it finds in $O(1)$ an approximation of the shortest residual path to the supersink. It maintains the guarantees required by the trust-inference method and offered by the optimal max-flow solution using Edmonds-Karp's algorithm: a) if there is flow on a link $j \rightarrow supersink$, there will be flow on this link in the optimal solution; b) if there is flow on j's outgoing links there will be flow on the link $j \rightarrow supersink$. The heuristic misses the cases in which it would be preferable to not use ancestor capacity to accept a child j but to use it for another child m, because child j may have another parent that can pass flow to it, while child m does not (Figure 2.3.3. However, in our $200K$-user network flow graph this was not often the case, as indicated by the fact that the max-flow achieved with our heuristic was typically $\sim 96\%$ of the optimal max flow.

2.3.4 OSN-Issued Credentials

After the OSN provider (Section 2.2.3) obtains the assertion veracity score for a user i's assertion A_i^t, it can issue a relaxed credential for this assertion. As shown in Figure 2.1, a credential issued by an OSN will include the assertion type t, the assertion A_i^t, and the assertion veracity score.

We use non-cryptographic web-based credentials that satisfy the goals listed in Section 2.2.5. Each credential as seen by the verifiers consists of:

- The `list of assertions` the user is verifying with their veracities and their types.

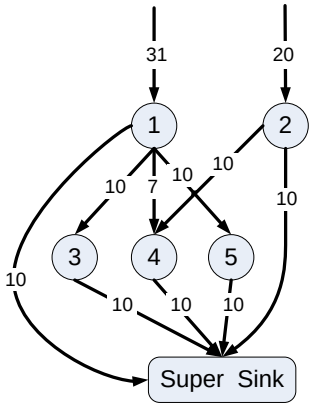

FIGURE 2.5*:* A flow graph topology case where our heuristic does not find the maximum flow. The edges are annotated with the allocated capacity. With the heuristic the edges $1 \rightarrow 4$, $2 \rightarrow 4$, $1 \rightarrow 5$ may get flow 7, 3, and 4, respectively. In this case, users 4 and 5 get trustworthiness 10 and 4, respectively. With Edmonds-Karp, the edges $1 \rightarrow 4$, $2 \rightarrow 4$, $1 \rightarrow 5$ will get flow 0, 10, and 10, respectively. This means that both users 4 and 5 get trustworthiness 10. In both cases all other nodes get trustworthiness 10.

- Content: An `excerpt` of the message (review, email, random string etc.) for which the credential is used.
- Context: A `URL` to or a `description` of the message for which the credential is used.

For example, a credential used for an online book review may include the following fields:

- [profession, CS professor, 100%, 17 tags]
- This is a great textbook and I highly recommend it ...
- `http://www.amazon.com/review/...`

This design binds a credential to the content and context it is used for, and ensures a credential's authenticity as it cannot be used to verify the assertion in a different content and context.

If a user wishes to obtain a credential to verify his identity attribute(s) when he posts

on an online forum, he first requests a unique HTTPS URL and ID for the credential from the OSN's credential issuing website. He then copies the credential URL or ID into the message he wishes to use, publishes the message, and obtains the message URL (context). Subsequently, he selects the assertions he wishes to certify, and submits them together with an excerpt of his message (content) and the URL to or ID of the credential issuing website. This completes the credential request process. The OSN provider creates a credential linked to by the credential URL, which includes the user selected assertions and their veracity scores, the excerpt of the message, and the message URL. To ensure integrity, once a credential is created, the user cannot modify it. Our implementation also includes the number of total tags an assertion has in a credential to give a verifier more information for better judgement.

To verify the credentials of a user, a human verifier can follow the HTTPS credential URL to view the credential through their browser. The FaceTrust site employs a CA-signed certificate. The use of secure HTTP addresses most challenges to the security of the web channel. However, the system still remains vulnerable to phishing attacks if the verifier is not trained to expect an HTTPS page or does not properly verify the certificate. Figure 2.3.4

In case the online message cannot include a credential URL, *e.g.*, Amazon strips URL's from user reviews, the user can instead include a specially formatted unique ID of the credential in his message when he requests the credential. A browser extension can read the specially formatted ID and prompt the user to view the credential.

In case a user wishes to use a credential to verify attributes to a human verifier with which he exchanges messages, the verifier and the user run the following protocol. The verifier sends to the user a randomly generated string. The user includes this string in the content field of the credential along with an excerpt of the message he wishes to send to the user.

FaceTrust's design also offers an XML web service API which can be accessed by

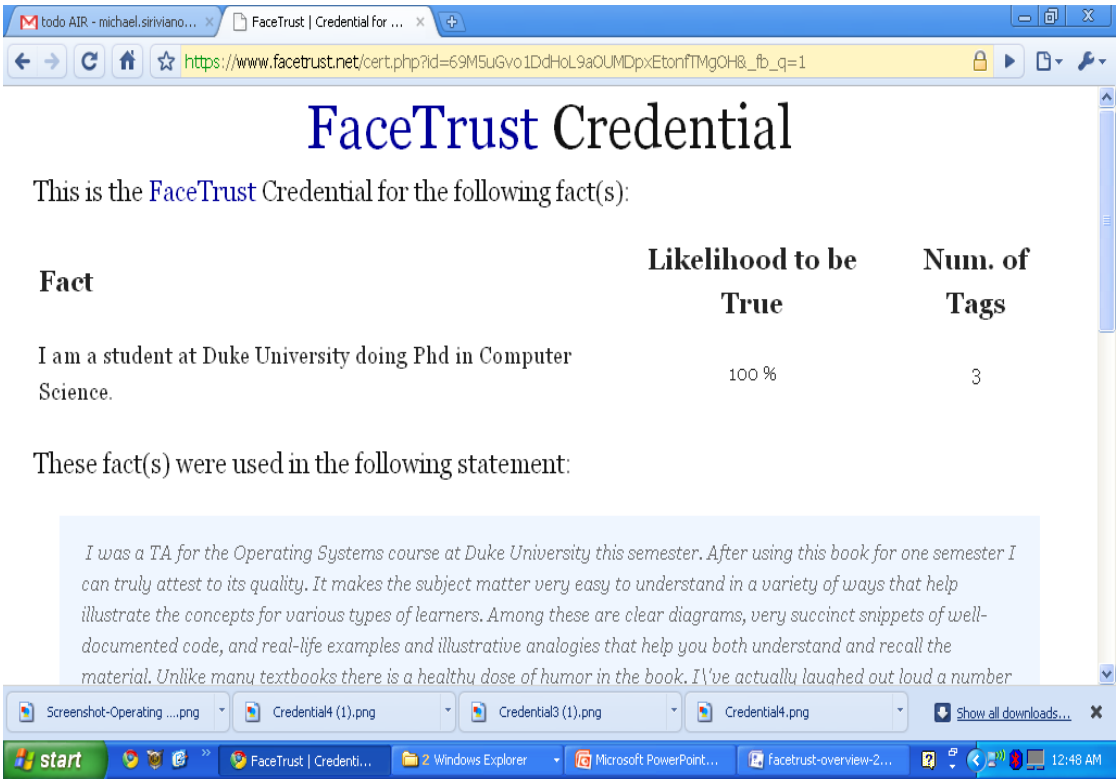

FIGURE 2.6*:* Illustration of a human-readable FaceTrust credential for an online book review.

online verifying services, such as web sites that perform automated access control based on the user's identity attributes (e.g., age.) The verifier presents a randomly generated string to the user, and the user generates a credential with the random string in the content field. Subsequently, the user posts to the online service the credential URL or ID that the user uses to certify his attributes, and the online service can retrieve the credential through an API call. A verifying web service is also configured a priori with FaceTrust's domain name.

Optionally Unlinkable Credentials. When a user creates a credential, he can choose to link the credential to his other credentials. This option is useful if the user is not concerned with verifying users or online services knowing what other credentials he created and (possibly) how he used them. He may instead desire to further reassure the verifier of the truthfulness or consistency of his assertions.

However, linked credentials may enable unscrupulous verifiers to survey the user against its will. The user is able to not choose this option if he is very concerned about his anonymity and untraceability. Note that the user has to generate a different credential each time it presents it to a verifier in order to preserve the unlinkability of his credential showings. Naturally, to preserve his anonymity a user has to ensure that he does not include any personally identifiable information in his assertions.

In an earlier design of FaceTrust [78], we proposed to use cryptographic anonymous, unlinkable, and non-transferable credentials [33]. We discarded this design to obviate the need for user-side cryptographic operations, as it has been shown it is difficult for average users to use cryptography [89].

2.3.5 Mitigating Sybil Assertion Posters

We now describe how we improve the above scheme to defend against the Sybil assertion poster attack (Section 2.2.4.) Under this attack groups of colluding dishonest users create a single or more Sybil accounts, post assertions on behalf of those Sybil accounts and tag them as `true`. The dishonest users subsequently share the Sybil accounts and use their assertions to create FaceTrust credentials.

Since honest users are not connected to the Sybil accounts and cannot tag their assertions, dishonest users do not need to tag differently from their honest friends. This results in high tagging similarity between honest and dishonest users. Subsequently dishonest users do not have lower tagger trustworthiness than honest users and their tags on the false assertions are not discounted.

We simultaneously employ two techniques to mitigate this attack. The first technique addresses the case in which a group of colluding users creates a single or a small number of Sybil accounts. We observe that colluders can create assertions on the few Sybil accounts, tag them as `true` and use them unimpeded to present multiple falsified credentials. We mitigate this attack by imposing a quota on the number of credentials each account can

issue. A reasonable approach in enforcing quotas is to impose an upper limit on the number of credentials a user can issue per month for each type of assertion, based on expected usage.

The second technique addresses the case in which the group of colluding users creates multiple Sybil accounts attempting to deal with the credential quotas. We modify the assertion veracity Equation 2.1 as follows. This solution relies on the assumption that honest users are typically both honest taggers and honest assertion posters. We can therefore use our Sybil-resilient tagger trustworthiness measure to infer how trustworthy their assertions are. To this end, we multiply the computed assertion veracity a_A of an assertion A posted by user j by a normalized value of the tagger trustworthiness of j.

$$a'_A = a_A \cdot min(1, c + (1-c)w^t_j/\overline{w}) \tag{2.4}$$

Where $\overline{w}$ is the tagger trustworthiness value for which $(1-p_d)|V|$ users have great or equal tagger trustworthiness. c is a tunable parameter, that assigns a minimum veracity $a_A \cdot c$ to assertion A in case the tagger trustworthiness w^t_j of j is 0. The factor $min(1, c + (1-c)w^t_j/\overline{w})$ is 1 for users with tagger trustworthiness higher than $\overline{w}$.

Since our trust inference method assigns low tagger trustworthiness values to multiple Sybils, this adjustment results in decreased veracity for assertions posted by colluders under this attack. Furthermore, since under several settings dishonest users have less tagger trustworthiness than honest users (Section 2.6.1,) this equation results in further decreasing the veracity of false assertions.

2.4 Attack-Resistance Analysis

We now discuss the attack-resistance of our design based on the assumptions and threats and goals listed in Sections 2.2.3, 2.2.4 and 2.2.5.

2.4.1 Assertion Veracity Analysis

We now discuss the assurances that the assertion veracity Equation 2.1 offers. For simplicity, we assume that the user j has higher than or equal tagger trustworthiness than $\overline{w}$, thus we do not need to consider the discounting introduced in Equation 2.4.

Theorem 2.4.1. *If the sum of the weights w_j^t of the dishonest users that have incorrectly tagged a false assertion by user i is less than* 0.75 *of the sum of the weight of the users that have tagged the assertion, the assertion will have less than* 0.5 *veracity.*

Proof. We denote as R the ratio of the sum of the tagger trustworthiness of the dishonest users that have tagged i's false assertion over the sum of the tagger trustworthiness of the users that have tagged the assertion. The dishonest friends of j tag the false assertion as `true` (1). The honest users tag the assertion as `false` (-1). The goal of the system is to make the veracity of the false assertion be less than 0.5.

Using Equation 2.1, we derive the inequality: $(R)1 + (1-R)(-1) < 0.5 \Leftrightarrow R < 0.75$.

□

We now examine Equation 2.2. The threshold M dictates how many dishonest users with a given tagger trustworthiness need to collude in order to make an assertion have non-zero veracity. A reasonable value for M is a multiple of the average tagger trustworthiness of honest users as derived by simulations (Section 2.6.1.) In the worst case, we need to consider dishonest users that have so far been tagging honestly and thus have obtained as high tagger trustworthiness as honest users. Assuming that that honest users have on average w tagger trustworthiness, at least $\frac{M}{w}$ dishonest users have to tag an assertion in

order for it to have non-zero veracity.

2.4.2 Tagger Trustworthiness Analysis

We now discuss the security assurances offered by the trust inference method used to derive a user's tagger trustworthiness (Section 2.3.3). We observe that our max-flow-based trust inference method satisfies the *bottleneck property*, by ensuring that the sum of the tagger trustworthiness of the dishonest user and its Sybils cannot exceed the sum of the capacity of the Sybil creator's incoming edges.

The purpose of our analysis is to derive the ratio R_{all} of the maximum sum of the tagger trustworthiness of dishonest users over the sum of the tagger trustworthiness of all users, when dishonest users are placed randomly in the social graph. Assuming that this ratio is on average maintained among the tagger trustworthiness of the friends of a user, R_{all} becomes a useful measure of the security of the assertion veracity, as is equivalent of the ratio R defined in 2.4.1.

We proceed by providing an upper bound M_d on the sum of the tagger trustworthiness of dishonest users and their Sybils. The tagger trustworthiness assigned to the Sybils of a dishonest user equals to the sum of the flows on the edges connecting the Sybils to the supersink. It is upper-bounded by the capacity C_u of the incoming edges of the dishonest user u that creates the Sybils. The closer the dishonest user is to the supersource, the higher the capacity of its incoming edges are. This is an inherent issue for all trust metrics, including TrustRank [49] and Sumup [85]: the closer a node is to the trusted seed users the more effective a Sybil attack is. The capacity of the incoming edges of the dishonest creator of Sybils depends on the distance from the supersource, the number of outcoming edges of users (fanout), and the capacity of the edges connecting the supersource to the trusted seeds. Given this observation, the Sybil-limiting assurances of this method rely on how capacity is distributed among the graph edges.

For simplification, the following analysis assumes that the similarity-based trust graph

consists of trees rooted at the trusted seeds. We also do not consider tagging-similarity in the edge capacity allocation during the transformation of the trust graph $T(V,E_t)$ to the network flow graph, thus assuming that all edges in E_t denote equal similarity. By not considering tagging similarity, we provide a pessimistic upper bound for M_d as the use of tagging similarity in assigning capacities results in the C_u of dishonest users to decrease. We also assume that there are no outgoing edges from dishonest users to honest users, which maximizes the tagger trustworthiness the Sybils of dishonest users can obtain.

As described in Section 2.3.3, the out-degree of the supersource is equal to the number of trusted seeds $|S|$. We assume that the out-degree of all non-leaf users u is f. Each trusted seed is the root of a subtree of $|V|/|S|$ nodes. V includes honest and dishonest users and does not include Sybils that dishonest users may create. Each user u is connected with an edge of capacity T_{max} with the supersink. Let p_d denote the portion of users in the similarity graph $T(V,E_t)$ that are dishonest. According to Section 2.3.3 (Phase 1), $C_{supersource} = T_{max}(1-p_d)|V|$ and the capacity C_s of the edge from the supersource to a trusted seed s is $C_{supersource}/|S|$. Also, $d = \lfloor \log_f C_s \rfloor$ denotes the maximum distance of users from a trusted seed.

Theorem 2.4.2. *For a given size $|V|$ of the trust graph, given out-degree f and given number of trust levels T_{max}, the maximum sum M_d of the tagger trustworthiness of dishonest users and their Sybils is only dependent on the portion of users that are dishonest p_d and the number of trusted seeds $|S|$. It does not depend on the number of Sybils that dishonest users employ. M_d is expressed as follows:*

$$M_d = p_d \cdot |S|(C_s \frac{(1-(1-p_d)^d}{p_d} + \frac{f\, T_{max}((f(1-p_d))^d - 1)}{(1-f)(f(1-p_d)-1)} + \frac{T_{max}((1-p_d)^d - 1)}{(1-f)p_d)}) \tag{2.5}$$

Proof. If a user u is at distance $1 \leq k \leq b$ from a trusted seed s, the capacity of its incoming

edges $C(u)_k$ is:

$$C_u(k) = \frac{C_s - T_{max}\sum_{i=0}^{k-1} f^i}{f^k} = \frac{C_s - T_{max}\frac{f^k-1}{f-1}}{f^k} \tag{2.6}$$

We aim at determining the sum of the flow M_d that can be allocated to the links that connect dishonest users and their Sybils with the supersink when we solve the max-flow from the supersource to the supersink. M_d corresponds to the maximum sum of the tagger trustworthiness of the dishonest users and their Sybils. It is equivalent of the sum of the capacity of the incoming edges of the dishonest users. This is because in the tree topology we are considering, a dishonest user can ensure that all its incoming edge capacity is utilized by creating enough Sybils and connecting its outgoing edges to them.

We consider the case when dishonest users are placed randomly in the graph. This is in fact a pessimistic assumption that results in a higher M_d, obtains dishonest users are less likely than honest users to be placed close to trusted seeds.

To derive the total maximum sum of the flow M_d, we sum up the capacities $C_u(k)$ of dishonest users across varying distances from the trusted seed. Under our assumptions, the capacity of the incoming edge $C_u(k)$ of a dishonest user is assigned exclusively to Sybils or other dishonest users. Thus, when we account for the capacity of dishonest users at distance k, we should not account for the capacity of dishonest users that have dishonest ancestors.

At distance $1 \leq k \leq d$ from the trusted seed, the average number of users (honest and dishonest) with no dishonest ancestor is $f^k(1-p_d)^{k-1}$. Correspondingly, the number of dishonest users at distance k with no dishonest ancestor is on average $p_d f^k(1-p_d)^{k-1}$. We derive that the maximum sum of the tagger trustworthiness M_d of dishonest users when dishonest users are placed randomly in the social graph and utilize all their capacity C_u to provide trust flow to dishonest users and Sybils is:

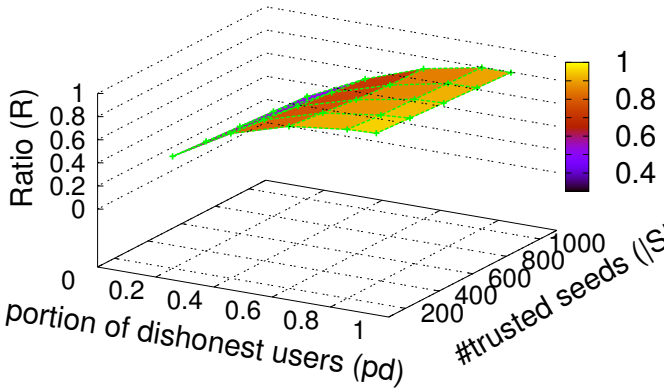

FIGURE 2.7: Ratio R_{all} of the maximum sum of the tagger trustworthiness of honest users over the sum of the tagger trustworthiness of all users, when dishonest users utilize all their capacity to provide flow (tagger trustworthiness) to dishonest users and Sybils. $|V| = 200K$, $T_{max} = 10$ and $f = 10$

$$M_d = p_d \cdot |S| \sum_{k=1}^{d} f^k (1-p_d)^{k-1} C_u(k) \tag{2.7}$$

The above equation is equivalent to Equation 2.5.

□

We can now provide the ratio R_{all} of the maximum sum of the trustworthiness of dishonest taggers over the sum of the trustworthiness of all taggers, when dishonest users are placed randomly in the social graph and utilize all their capacity to provide flow to dishonest users and Sybils.

$$R_{all} = \frac{M_d}{C_{supersource}} = \frac{M_d}{T_{max}|V|(1-p_d)} \tag{2.8}$$

According to Equation 2.1, a good ratio R_{all} to prevent dishonest users from making their assertions appear credible on average is 0.75. As can be deduced from the above equations and is illustrated in Figure 2.4.2, the ratio R_{all} increases substantially as a function of the portion of dishonest users p_d and decreases slightly as a function of the number

of trusted seeds$|S|$. It does not depend on the number of Sybils that dishonest users employ. Under the examined setting ($|V| = 200K, f = 10, T_{max} = 10$, increasing $|S|$ from 100 to 1000 results in R_{all} decreasing by ~ 0.1. In addition R remains below 0.75 as long as p_d is below 0.4.

The ratio R is large even when p_d is small (e.g, $R = 0.44$ when $p_d = 0.2$ and $|S| = 400$) because our model examines the worst case scenario: a dishonest user never distributes capacity to its children that are honest. This corresponds to dishonest users assigning 0 tagging similarity to their honest friends, while their honest friends assign 1.0 tagging similarity to them. Under our model, as the number of trusted seeds $|S|$ increases, honest users have more opportunities to receive capacity from the trusted seeds or other honest users. This is the reason we observe the decrease of R_{all} as $|S|$ increases.

Our simulation-based evaluation (Section 2.6.1) considers a real social graph (under which honest users can be connected to trusted seeds via multiple paths), thus it yields higher trustworthiness for honest taggers. Under our tree-topology model, the existence of multiple seeds also mitigates the impact of a focused adversary that manages to be close to a trusted seed and establish high tagging similarity with it. This is because this focused dishonest user at distance k from the trusted seed obtains at most $(C_{supersource}/|S| - T_{max}\frac{f^k-1}{f-1})/f^k$ capacity for its incoming edges instead of $(C_{supersource} - T_{max}\frac{f^k-1}{f-1})/f^k$.

2.5 Implementation

We implemented FaceTrust as a three-tier web application so that we can evaluate the design and its assumptions using a real-world deployment. The front-end of FaceTrust is the "Am I Really?" Facebook application, which is implemented using the PHP Facebook developer API [8]. The FaceTrust application server serves the HTTP content, collects assertions and tags, as well as maintains the social graph. The FaceTrust server employs MySQL to store the user, assertion, tagging and social graph information. It uses a Java implementation of MaxTrust to perform the veracity computations.

Game with a Purpose: We built the "Am I Really?" (AIR) Facebook application [1] using a "game with a purpose" design to incentivize social tagging. AIR is a "micro-polling" application. Users post facts about themselves and other statements and ask their AIR friends to tag them as `true` or `false`. For example, a user may post the question "Am I really older than 18?" or questions of lighter nature, such as "Am I really good at baseball?" AIR users can view the veracity of a friend's assertion only after they have tagged his assertion, and after the assertion amassed a threshold number of tags.

According to Facebook's terms of use, we are not allowed to store long term the friends of a user that the Facebook API provides. To circumvent this restriction, AIR asks a user to declare which of his friends he would like to have as friends on AIR. To further ensure that connections in the AIR social graph correspond to real life acquaintances (Section 2.2.3) and to address the problem of promiscuous users that naively establish Facebook connections with malicious users, we explicitly ask a user to declare as AIR friends only persons with whom he has met in person.

We also build a credential issuing website `https://www.facetrust.net`, which links each user's account to their Facebook account. A user that wishes to prove an identity attribute to other users or online services requests a credential as we describe in Section 2.3.4. The issuing website pulls the user's assertions and their veracity scores from

the AIR database back-end.

2.6 Evaluation

By evaluating FaceTrust, we aim to understand the following aspects of its operation:

- Effectiveness: How well do the assertion veracity and tagger trustworthiness scores correlate with the truth, and how well does the design withstand incorrect user tagging, colluder attacks and Sybil attacks?
- Practicality and usage: How often and how accurately does a user tag his friends to help them obtain credentials?
- Computational feasibility: A social network may consist of several hundreds of millions of users. Will an OSN provider have sufficient computational resources to mine the social graph and derive tagger trustworthiness scores?

We use simulations on a crawled Facebook social graph and a real-world deployment to answer these questions. We discuss each in turn.

2.6.1 Effectiveness

Our effectiveness evaluation aims at examining whether true assertions obtain high veracity and false assertions obtain low veracity, even in the presence of dishonest users and Sybil attacks. It also aims to determine the limits of our approach, i.e., under which conditions false assertions can obtain high veracity and what strategies the attackers need to deploy in order to defeat the system.

We start by evaluating the ability of our max-flow-based trust inference scheme (Section 2.3.3) to assign low tagger trustworthiness to dishonest users and Sybils. We then proceed to analyze the effectiveness of the assertion veracity mechanism (Section 2.3.2), which weighs user tags based on tagger trustworthiness.

For a more realistic evaluation, we use a crawled sample of the Facebook social graph [44]. The social graph we use is a $200K$-user connected component obtained from a $50M$-user sample via the "forest fire" sampling method [58]. It has been shown [58] that the forest

fire technique outperforms random node, random edge or random walk sampling in preserving important statistical graph properties (e.g. degree distribution, distribution of sizes of connected components, the distribution of the clustering coefficient and the diameter) in order to capture a snapshot of the network at an earlier point in time.

"Forest fire" sampling proceeds as follows: a) We randomly choose a seed node s; b) We generate a random variable x that is binomially distributed with mean $(1p_f)^1$, with $p_f = 0.7$. We sample ("burn") x randomly selected edges of s and their incident nodes. Let $w_1, w_2, ..., w_x$ denote the other ends of these selected links. (c) We apply recursively step (b) from nodes $w_1, w_2, ...w_x$. Already sampled (burned) nodes cannot be revisited.

The average number of friends of each user in the sample graph is ~ 24 and the maximum number of friends is 313. The diameter of this graph is 18. The clustering coefficient is 0.159.

General Simulation Settings

Each user in the social graph posts a single assertion of the same type on his profile. We have two types of users: honest and dishonest. Honest users always post true assertions and dishonest users always post false assertions. Furthermore, the honest users tag their friends truthfully, that is, they tag as `true` the assertions made by their honest friends and as `false` the assertions made by their dishonest friends.

Unless specified otherwise, the dishonest users tag all assertions as `true`, regardless of whether the users that post them are honest or not. By doing so, dishonest users collude to increase the veracity of each other's assertions. By truthfully tagging the assertions of honest users, dishonest users attempt to have common tags with other honest users in order to increase their tagging similarity with trustworthy users (the tagger camouflage attack described in Section 2.2.4.) Thus we are also explicitly considering the case in which a user is not always tagging falsely, but instead has a mixed strategic behavior.

Unless specified otherwise, both honest and dishonest users are randomly distributed

in the social graph. In addition, each user tags the assertions of at most F of his friends. We vary F to reflect various degrees of adoptability of social tagging.

We obtain the tagger trustworthiness as described in Section 2.3.3. We do not consider the user-defined similarity (Section 2.3.3), as we use no notion of a priori trust between users. We set $T_{max} = 100$ (Section 2.3.3). For each experiment, we set the minimum sum of the trustworthiness of taggers M (Section 2.3.2) equal to the average tagger trustworthiness of honest users. We set the parameter c (Section 2.3.5) equal to 0.2. For all experiments we employ 1000 trusted seeds which are randomly selected among the honest users. For each configuration we repeat the experiment 5 times and plot the mean, and 95% confidence intervals (too small and not visible in most configurations).

Tagger Trustworthiness Effectiveness

As described in §2.3.2, the tags on assertions are weighted by their tagger's trustworthiness. Therefore, we first need to examine the effectiveness of tagger trustworthiness under various strategies employed by dishonest users. We consider the tagger trustworthiness scheme effective if: a) it assigns substantially lower trustworthiness to Sybil users than to honest users; and b) it does not assign higher trustworthiness to dishonest users than to honest ones.

Dishonest users do not employ Sybils: In this series of experiments, dishonest users do not employ Sybils. As can be seen in Figure 2.6.1, the tagger trustworthiness of honest users is substantially higher than the one of dishonest users when the portion of honest users is small. Honest and dishonest users have the same connectivity. They differ only in terms of tagging. When the portion of honest users is relatively low and honest and dishonest users are placed randomly, there are many opportunities for honest and dishonest users to tag dissimilarly. We also observe that the trustworthiness of both honest and dishonest users decreases with the portion of honest users. This is because the more dishonest users exist in the network, the more likely it is for the honest users to be disconnected from

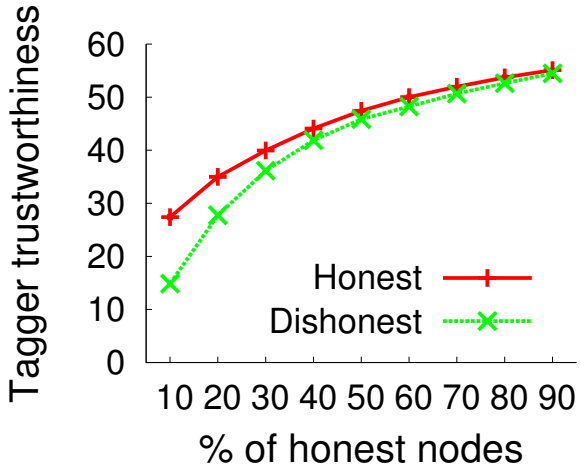

FIGURE 2.8: Mean tagger trustworthiness as a function of the fraction of honest nodes when the maximum number F of friends a user tags is 20 ($F = 20$) and dishonest users do not employ Sybils.

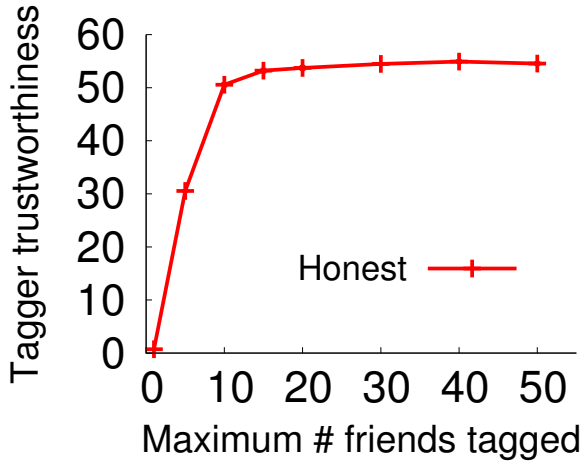

FIGURE 2.9: Mean tagger trustworthiness of honest users as a function of F when 80% of users are honest and dishonest users do not employ Sybils.

trusted seeds.

Tagging similarity captures the difference in tagging behavior between dishonest and honest users, and this translates to low pairwise trust between them. Since trust is seeded at honest users, MaxTrust's transitive trust mechanism assigns lower tagger trustworthiness to dishonest users. This result demonstrates the importance of using tagging similarity. When dishonest users tag incorrectly very often, this mechanism results in them getting

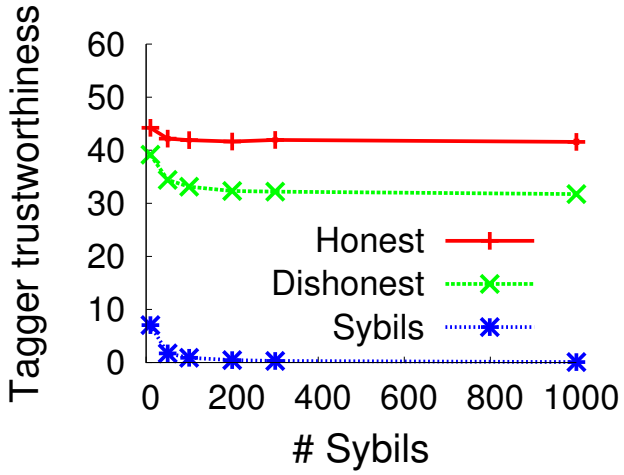

FIGURE 2.10*:* Mean tagger trustworthiness as a function of the number of Sybils each dishonest user creates when 50% of users are honest and $F = 20$.

substantially decreased tagger trustworthiness, thus having less influence on the system's operation.

Figure 2.6.1 shows the trustworthiness of honest users as a function of the maximum number of friends F each user tags. As F increases, the number of common tags C_t (Section 2.3.3) used to derive the tagging similarity increases. For $F < 10$, the tagging similarities between users are almost 0 and the similarity-based trust graph is disconnected, resulting in honest users getting very low trustworthiness. As F increases, the trust graph becomes more connected and honest users obtain increased tagger trustworthiness.

When tagging is infrequent, a large fraction of edges between honest users do not have high tagging similarity, as it becomes less likely for honest users to tag the same assertions. As a result, honest users get relatively low veracity. As can been seen in Figure 2.6.1, honest users to achieve high tagger trustworthiness, F should be ≥ 10.

Dishonest users employ Sybil Taggers: To evaluate the scheme's resilience to Sybil attacks, all the dishonest users create a varying number of Sybils. The dishonest users connect to all the Sybils they create and the Sybils connect only to their creator. Sybils tag the false assertions of their creator as `true` to increase the veracity of those assertions. The creator always has tagging similarity 1.0 with all its Sybils. This corresponds to the

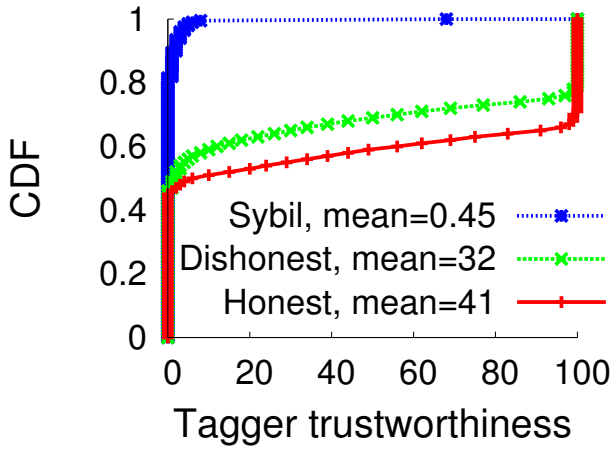

FIGURE 2.11*:* CDF of the tagger trustworthiness of honest, dishonest and Sybil users when 50% of users are dishonest, $F = 20$ and dishonest users employ 200 Sybils each.

configuration that maximizes the tagger trustworthiness of Sybils.

As can be seen in Figure 2.6.1, when the number of Sybils is 200, the tagger trustworthiness of Sybils is on average 90 and 70 times lower than the trustworthiness of honest and dishonest users, respectively. This is due to the bottleneck property of our trust inference mechanism (Section 2.3.3), which limits the amount of trust that can be assigned to the Sybils of a dishonest user.

We also observe that as the number of Sybils increases from 10 to 200, the tagger trustworthiness of honest and dishonest users also decreases by 12% and 30%, respectively. The reason is that the incoming flow that passes through a dishonest user is now assigned to the Sybil users instead of other users downstream. Nevertheless this decrease is not substantial, especially for honest users, because there are multiple trust paths through which flow can reach these users.

We now examine how tagger trustworthiness is distributed among the users. Figure 2.6.1 depicts the CDF of the tagger trustworthiness of honest, dishonest and Sybil users. As can be seen, there is substantial variance in the trustworthiness values of honest and dishonest taggers. Nevertheless, on average dishonest users have substantially lower trustworthiness due to their decreased tagging similarity with honest users. Furthermore, al-

most 90% of Sybil users has 0 tagger trustworthiness.

Since we set the assertion veracity threshold (Equation 2.2)) to be close to the mean tagger trustworthiness, the tagger trustworthiness distribution greatly affects how assertion veracity is computed. In MaxTrust we can safely set the threshold equal to the mean tagger trustworthiness for that portion of honest users, and in many cases the honest users that tag an honest assertion have sufficient tagger trustworthiness.

Assertion poster camouflage attack: We also evaluate the resilience of FaceTrust when dishonest users use the assertion poster camouflage attack. Section 2.2.4. Honest users post one true assertion, and dishonest users post one false assertion with a varying number (1-10) of true assertions for camouflage. We obtain the average tagger trustworthiness of honest and dishonest users under 80% honest nodes, 200 Sybils per dishonest users and $F = 20$. We observe that the effect of the camouflage attack on the resulting tagger trustworthiness is insignificant.

Although it is not demonstrated in our experimental setting, this attack can be further mitigated by assigning distinct tagger trustworthiness values for each type. As a result the camouflage attack cannot occur for many assertions of the same type because the tagging similarity of the user's dishonest friends with honest friends for that particular type will drop, and so will their tagger trustworthiness.

Trustworthiness evaluation conclusions: The above results illustrate that under our tagging-similarity-based trust inference mechanism dishonest users obtain substantially lower trustworthiness than honest users. In addition, we show that Sybil users obtain almost two orders of magnitude less trustworthiness, under common Sybil strategies. Our results have also illustrated the importance of the frequency of tagging, as modeled by the parameter F.

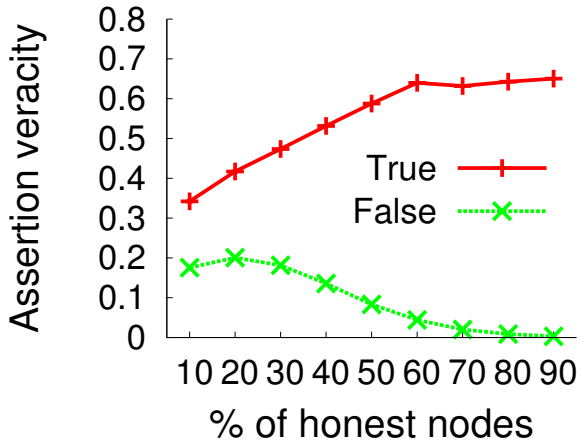

FIGURE 2.12*:* Mean veracity of true and false assertions as a function of the fraction of honest nodes when $F = 20$ and dishonest users do not employ Sybils.

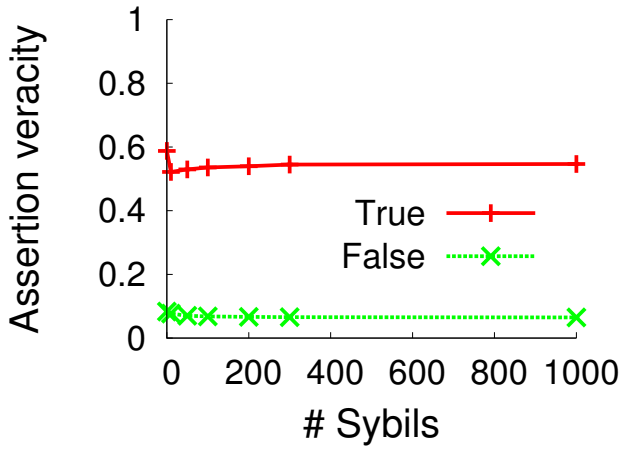

FIGURE 2.13*:* Mean veracity of true and false assertions as a function of the number of Sybils each dishonest user creates when 50% of users are honest.

Assertion Veracity Effectiveness

The assertion veracity scoring is dependent on the mechanism for determining the weight of the taggers, which was evaluated in the previous section. We now evaluate the assertion veracity scoring technique itself, which was introduced in Section 2.3.2. We evaluate the resilience of the assertion veracity equation under varying attack scenarios.

Dishonest users do not employ Sybils: Figure 2.6.1 plots the mean veracity of true and

false assertions as a function of the portion of the users that are honest. In this experiment, dishonest nodes do not employ Sybils. We observe that when the fraction of honest users exceeds 50%, the mean veracity of true assertions substantially exceeds that of false ones. Unlike plain majority voting approaches, our mechanism assigns low veracity to false assertions even when the fraction of dishonest users is large. This is because MaxTrust assigns lower tagger trustworthiness to dishonest users and their tags are discounted.

As the portion of dishonest users increases, the number of users that obtain very low tagger trustworthiness increases. Consequently, multiplying the veracity of an assertion by the normalized tagger trustworthiness of its poster (Section 2.3.5) decreases the veracity of both true and false assertions.

Dishonest users employ Sybil taggers: Figure 2.6.1 shows the veracity of true and false assertions when dishonest users employ Sybils. Each dishonest user creates a varying number of Sybils. The Sybils are connected only to their creator and tag all its assertions as `true`. As can be seen, the dishonest users gain little benefit by using Sybils in our setting. Although there are many Sybil taggers for false assertions, most of them have very low (or 0) tagger trustworthiness and the sum of tagger trustworthiness of Sybil taggers is most often below the threshold M (Section 2.3.2.) Moreover, because the mean tagger trustworthiness of dishonest users which employs Sybils decreases, multiplying by the normalized tagger trustworthiness of the poster reduces the mean veracity of dishonest users.

Figure 2.6.1 shows how veracity is distributed among true and false assertions. We depict the CDF of the assertion veracity of all $200K$ assertions. We observe that 57.5% and 12.5% of true assertions obtain veracity equal to 1 and 0.2, respectively. 24.3% of true assertions obtain 0 veracity. The true assertions with 0.2 veracity belong to honest users with 0 tagger trustworthiness. The true assertions with 0 veracity are the ones for which the sum of their taggers' trustworthiness scores are below M. The number of these

incorrectly assessed true assertions can be reduced by increasing the maximum number of friends that users tag (F), i.e., increasing the adoption of social tagging. Incorrectly assessed assertions can be further avoided designating more trusted seeds (S.)

Unlike true assertions, most of the false assertions, 85%, obtain 0 veracity. Only 1.5% of false assertions obtain veracity 1. This result suggests that FaceTrust's assertion veracity scoring mechanism is effective, but not absolutely accurate. Thus, it should not be used to control access to critical resources.

Dishonest focused colluders: We also evaluate the case in which dishonest users form colluders groups. The dishonest colluders in a group are connected to each other and tag each other's assertions, as `true`. This experiment differs in that it is guaranteed that each dishonest user has a specified minimum number of dishonest colluders. This corresponds to a more focused and coordinated attack. Figure 2.6.1 depicts the mean veracity of the assertions posted by the dishonest users as a function of the size of the colluder groups.

We observe that colluders can get higher average veracity of assertions than true assertions only if their size exceeds a relatively high threshold (in this case 30). This is due to: a) the increased number of dishonest taggers; and b) the increased tagger trustworthiness of the colluders. The tagger trustworthiness of colluders increases because users closer to seeds can get higher tagger trustworthiness in MaxTrust and the colluders are all connected to each other. If a single colluder is close to the trusted seeds, all the colluders may get reasonably high tagger trustworthiness. As the number of colluders increases, both sources of increased tagger trustworthiness become more prominent and the assertions of colluders get high veracity.

This result reveals a limit of our approach. If a substantial number of colluders coordinates, they can ensure that their assertions have high veracity. Nevertheless, rational colluders need to expend effort (Section 2.2.4) to perform this attack, thereby they can be discouraged. For this reason, FaceTrust credentials should not be treated as the absolute

truth and they should instead be used as an additional indication of veracity.

Dishonest users employ Sybil assertion posters: We now evaluate our system when a group of dishonest users performs the Sybil assertion poster attack (Section 2.2.4.) Each group of colluders creates Sybils to which all the colluders connect to. The Sybil users post assertions and all the colluders tag them as `true`. At the same time dishonest users tag honestly for all other assertions in an attempt to establish high tagging similarity with honest users. This attack is equivalent to dishonest users who choose to connect only to other dishonest colluders. As explained in Section 2.2.4, this attack results in the colluders having access to assertions that cannot be voted as `false` by other honest users, thus they are expected to have high veracity.

In Figure 2.6.1, when the number of Sybil assertion posters is small, e.g., 10, we observe that the assertion veracity is high. Since the number of Sybils is small, MaxTrust does not assign low tagger trustworthiness to them. Consequently, Equation 2.4 (Section 2.3.5) does not mitigate this attack, because both the colluding dishonest taggers and the Sybil posters have relatively high tagger trustworthiness. This result reveals another limit of our approach. Nevertheless, FaceTrust prevents dishonest users from using the assertions of those Sybils in multiple contexts by imposing a quota (Section 2.3.5) on the number of credentials each user can request.

When the colluders create many Sybils to overcome the quotas, they have to cope with the fact that the tagger trustworthiness of the Sybils is reduced. Consequently, the mean assertion veracity is reduced as shown in Figure 2.6.1. This result indicates the importance of multiplying the assertion veracity by the poster's normalized tagger trustworthiness as described in Section 2.3.5. Furthermore, rational dishonest users incur a cost to create Sybils (Section 2.2.4), which further limits their ability to subvert the system.

Veracity evaluation conclusions: The above results illustrate that our assertion veracity scoring technique results in false assertions obtaining substantially lower veracity than true

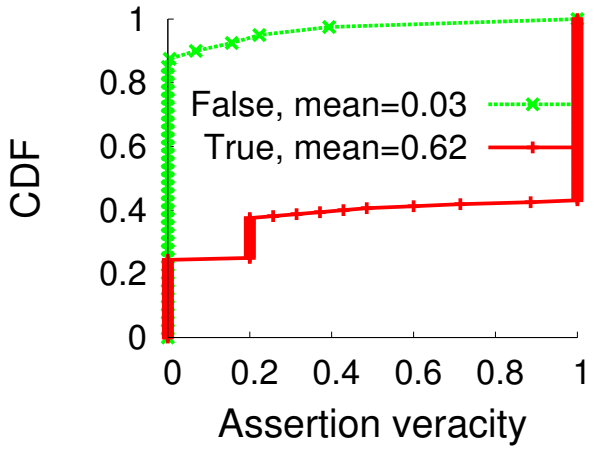

FIGURE 2.14*:* CDF of assertion veracity, when 80% of users are honest, $F = 20$, and dishonest users employ Sybils.

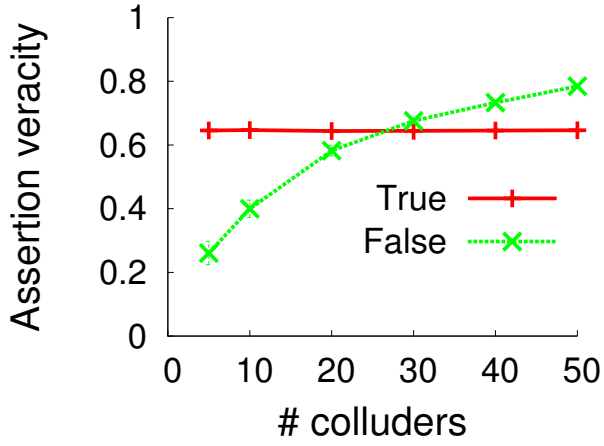

FIGURE 2.15*:* Mean veracity of assertions posted by dishonest colluders as a function of the colluder group size, when 80% of users are honest and $F = 20$.

ones. We show that this holds even under commonly deployed attack strategies (Sybils and colluders). In addition, we demonstrate the limits of our approach by explicitly describing elaborate colluding attacks that FaceTrust does not sufficiently mitigate.

2.6.2 Facebook Deployment

FaceTrust requires a new form of user input: assertions and tags. In addition, in order for the veracity scores to correlate positively with the ground truth, it requires trustworthy

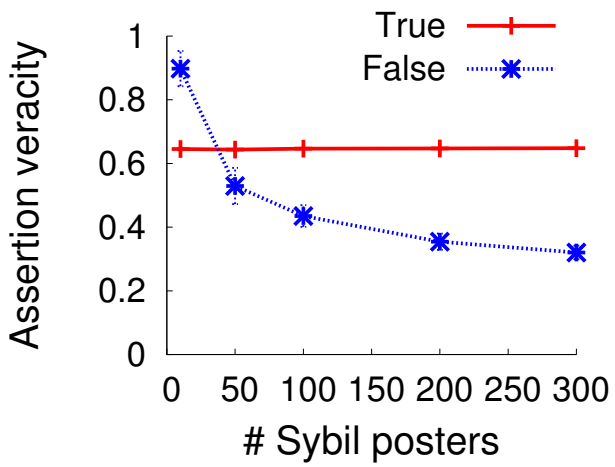

FIGURE 2.16*:* Mean veracity of assertions posted by Sybil posters as a function of the number of Sybil assertion posters in the group, when 80% of users are honest, $F = 20$, and the group size is 30.

users to tag honestly and similarly. These facts motivate us to ask: Are users willing to tag their friends' tags? How often and honestly will they tag? To answer these questions, we implemented the "Am I Really?" (AIR) Facebook application (Section 2.5) for users to post and tag assertions, we invited our friends to use it, and advertised it on Facebook. The Facebook advertisement resulted in approximately 100 installations of AIR.

We collected a data set consisting of 1108 users. 395 of those users chose to declare that they are friends with at least one AIR user, thus having one or more neighbors in the AIR social graph. For the rest of this evaluation we provide statistics concerning those 395 users, since they are the only ones that can tag friends in AIR. Our data set includes 2410 social connections established between Sept. 1st, 2009 and Jan. 10, 2010. These connections form several connected components, the largest of which includes 182 users. The average number of friends a user has in that largest component is 3.8 and the diameter of the component is 4. Our live system computes tagger trustworthiness using MaxTrust. We employ 10 trusted seeds, set $T_{max} = 10$ and assume that 90% of the network consists of honest users.

To protect user privacy, we anonymize all Facebook and AIR-specific identifiers and

exclude the assertions that include personally identifiable information prior to data processing. In addition, the application informs its users that their personal data will not be published.

We incorporate user-defined similarity (Section 2.3.3) in the computation of tagging similarity, using $b = 5$. We again set the parameter c (Section 2.3.5) equal to 0.2. On average, friends have N_t (as defined in Section 2.3.3) equal to 5.2, 10.4, 3.7. and 2.8 for type t of age, location, profession, and gender, respectively.

Figure 2.6.2 shows the complementary cumulative distribution of users as a function of the number of tags they post. We observe that even in this small social graph, more than half of the users have tagged at least 8, 6, 4, and 1 time for type age, profession, location and gender, respectively. We also find that users tag on average 14.4, 10.4, 7.5, and 4.6 times for assertions of type age, profession, location, and gender. We believe that when the system is widely adopted, in a larger social graph, users will have on average many more friends to tag. Thus, we speculate that the number of assertions users tag is likely to exceed 10, the number needed to obtain accurate tagger trustworthiness (Figure 2.6.1 in Section 2.6.1).

Figure 2.6.2 shows the complementary cumulative distribution function for the number of assertions users post for each assertion type. More than one quarter of the 395 users have posted at least 8, 6, 4 and 2 assertions of type age, profession, location and gender, respectively. We find that users post on average 5.6, 3.6, 2.6, and 0.9 assertions of types age, profession, location, and gender, respectively. This is indicative of the fact that users use this application as intended and do not feel uncomfortable reporting such information to their friends and FaceTrust. We also observe that users tend to post more assertions that concern their age or profession than their location. This is possibly because users are not as motivated to ask others what they think about their location or to certify their location.

Next, we take a closer look at the AIR profiles of 10 out of the 395 users, for which we know the correct answer for their age, gender, location and profession assertions. We

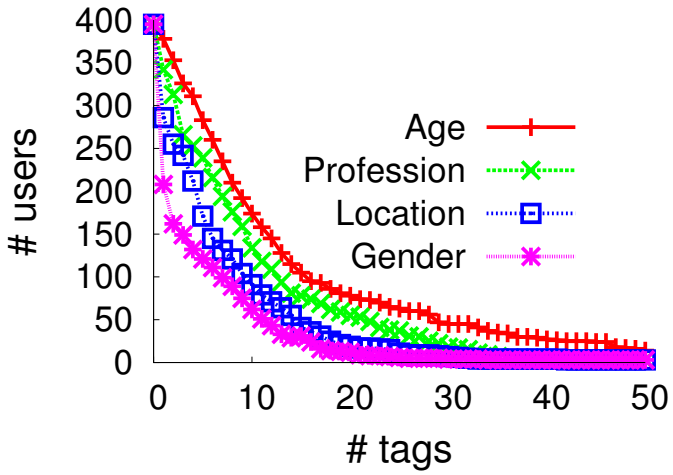

FIGURE 2.17*:* CCDF of the number of users as a function of the number of tags per user for each assertion type.

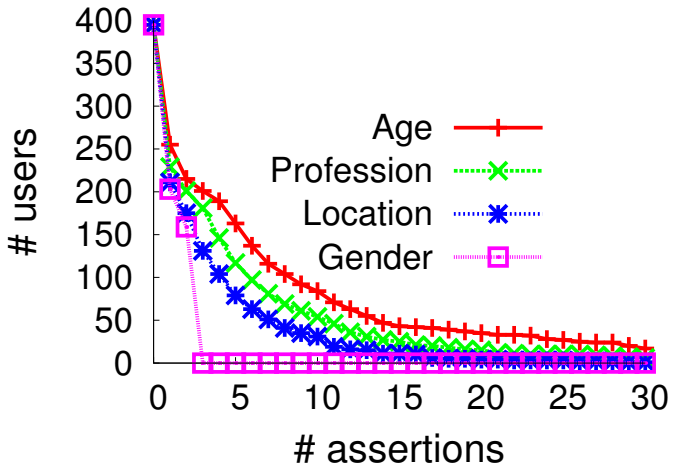

FIGURE 2.18*:* CCDF of the number of users as a function of the number of posted assertions per user for each assertion type.

collect a total of 50, 50, 50 and 20 age, profession, location and gender assertions, respectively. These include 14 false age assertions, 21 false profession assertions, 19 false profession assertions, and 10 false gender assertions. Each of these assertions were tagged ~ 6 times on average by distinct users.

Figure 2.6.2 shows the mean veracity per type of the true and false assertions with and without attackers in the system. Per each type, the first column depicts the mean assertion

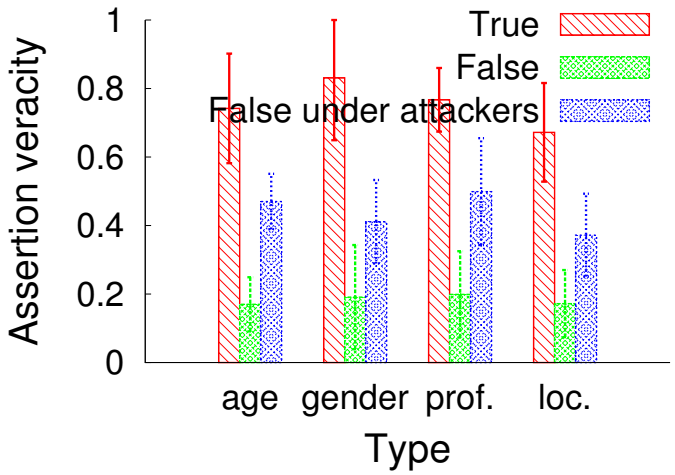

FIGURE 2.19*:* Veracity per type of true and false assertions in FaceTrust's real-world deployment with and without attackers. The error bars denote 95% confidence intervals.

veracity of true assertions. The second column depicts the mean assertion veracity of false assertions in the absence of attackers. The third column shows the mean veracity of false assertions when we inject 20 artificial dishonest users in AIR's social graph. The injected dishonest users do not represent real Facebook accounts. They are connected to the 10 real honest users, such that each of these real users is AIR-friends with two distinct dishonest users. The injected dishonest users tag the false assertions of the 10 real users as `true`. In an attempt to increase its similarity with honest users, an injected dishonest user launches the camouflage attack by tagging all the other assertions as `true`, if their prior veracity of the assertion was greater than 50% and `false` otherwise.

As can be seen in Figure 2.6.2, the computed veracity for true and false assertions in the absence of attackers correlates very well with the ground truth. This result indicates that users tend to tag correctly. We observe that users may make some mistakes in assessing each other's age, but when the truth for an assertion is straightforward, such as for gender, the veracity of the assertion is high.

As can be seen in the third column per type, the attack by the injected dishonest users has affected the computed veracity of false assertions. This is mainly because the AIR social graph is small, with each real honest user having less than 4 honest friends on

average. Two attackers per user have caused the false assertions' veracity to increase substantially. However, there is still a distinguishable gap between the average veracity of true and false assertions, suggesting that FaceTrust's assertion veracity scoring mechanism is resilient.

Deployment conclusions: Our Facebook deployment results indicate that users tag sufficiently frequently for the tagger trustworthiness measure to be effective. Importantly, our results also indicate that benign users tag mostly correctly. This demonstrates the efficacy of relying on users to certify each other's identity attributes.

2.6.3 Computational Efficiency

We have benchmarked the computational overhead to derive MaxTrust's max-flow-based tagger trustworthiness, using a 3.4GHz P4 machine with 2GB memory running Debian 2.6.25. We repeated the measurement 5 times. The mean computation time to obtain the tagger trustworthiness using our max-flow-based method for all $200K$ users and for $T_{max} = 100$ users is 629 sec. The computation time is almost independent of the number of trusted seeds. The required memory is $\sim$ 550MB. ($\sim$ 500MB for graph structure data, $\sim$ 50MB for computation.)

We observe that the computation cost of our heuristic for sub-million node graphs is not excessive. Using parallel computation techniques, *e.g.*, MapReduce [38] as used for the computation of PageRank in Google's datacenters, and by BotGraph [96], we expect that this computation could scale to multi-million user social networks.

On the other hand, solving the optimal max-flow using Edmonds-Karp algorithm is computationally prohibitive. For the same $200K$-user social graph, under the same machine configuration, Edmonds-Karp requires approximately 1 *million* sec.

3

SocialFilter: Introducing Social Trust to Collaborative Spam Mitigation

In this chapter, we describe how we use Online Social Networks to assess the trustworthiness of reporters (detectors) of spamming hosts in a collaborative spam mitigation system.

3.1 Introduction

The majority of the currently deployed spam email mitigation techniques rely on centralized infrastructures and place trust on a small number of security authorities. For instance, email systems and browsers rely heavily on a few centralized email sender reputation services (e.g., [17, 76, 15].)

Unfortunately, these services often maintain out-dated blacklists [72], offering a rather large window of opportunity to spammers. Moreover, the number of nodes that detect and report spam to them is limited. At the same time, attacks launched using large botnets are becoming increasingly surreptitious. That is, a single malicious host may attack multiple domains, each for a short period of time [70, 54], reducing the effectiveness of spam traffic detection with a small number of vantage points. Finally, several of these services

require paid subscription (e.g., CloudMark [3] and TrustedSource [15].)

Motivated by the shortcomings in terms of effectiveness and cost of these centralized email sender reputation services, researchers have proposed collaborative peer-to-peer spam filtering platforms [97, 98]. These systems assume compliant behavior from all participating reporters of spam. This is hardly true given the heterogeneity of the Internet and the fact that reporters may belong to distinct trust domains. Compromised hosts controlled by attackers may join the system and pollute the detection mechanisms. In addition, honest reporters may become compromised after they join the system. A more recent collaborative spam email sender detection system employs trust inference to weigh spammer reports [77], however it is still susceptible to Sybil attacks.

To this end, we propose SocialFilter, which is a Sybil-resilient collaborative spam filtering system that uses social trust embedded in Online Social Networks (OSN) to evaluate the trustworthiness of spam reporters. SocialFilter aims at aggregating the experiences of multiple security authorities, democratizing spam mitigation. It is a trust layer that exports the a measure of the system's belief that a host is spamming. Thus, it enables nodes with no spam detection capability to collect the experiences of nodes with such capability and use them to classify email connection requests from unknown email senders.

Each SocialFilter node submits *spammer reports* (Section 3.2.2) to a centralized repository. These reports are security alerts that classify spamming Internet hosts identified by their IP addresses. The goal of the system is to ensure that the reports reach other nodes prior to spamming hosts contacting those nodes, and that the spammer reports are sufficiently credible to warrant action by their receivers.

SocialFilter nodes are administered by human administrators (admins). Our insight is that nodes maintained by trusted admins are likely to generate trustworthy spammer reports, while nodes maintained by admins known to be less competent are likely to generate unreliable reports. The repository utilizes a trust inference method to assign to each node a *reporter trust* (Section 3.3.1) score. This score reflects the system's belief that the

spammer reports of a node are reliable.

The trust inference method exploits trust transitivity and operates on a trust graph that is formed at the repository as follows (Section 3.3.1.) Each vertex in the graph is a SocialFilter node, which is administered by a human admin. The edges in the graph are direct trust values between nodes administered by admins that are socially acquainted. The social relationships between admins are obtained from OSN providers, First, each admin explicitly assigns a direct trust value to nodes that are administered by his friends, based on his assessment of his friend's competency. Second, the direct trust values between nodes are updated to reflect the similarity between their spammer reports. This is based on the observation that trustworthy nodes are likely to report similarly on commonly encountered hosts. Third, the repository computes the maximum trust path from pre-trusted nodes to all other nodes in the trust graph using Dijkstra's algorithm.

However, traditional transitive trust schemes and email reputations systems are known to be vulnerable to the Sybil attack [39, 34, 35]. To mitigate this attack, we again use the social network to assess the belief that a node is a Sybil attacker, which we refer to as *identity uniqueness* (Section 3.3.2). Each node is associated with its administrator's identity. The identity uniqueness of a node is determined via the social network of administrators using a SybilLimit-based technique [92](Section 3.3.2).

The originator of a spammer report also assigns a confidence level to its report. The reporter trust of a node, its identity uniqueness, and its confidence level in a report determine how much weight the repository should place on each report. Subsequently, the repository combines the reports to compute a *spammer belief* score for each host IP reported to the system. This score can be explicitly interpreted as the belief that a host is spamming. This score is exported by the repository to other online systems for diverse purposes. For example, email servers can use them to automatically filter out email messages that originate from IPs that have been designated as spammers. IDS systems can use them to block SMTP packets that originate from suspicious IPs.

Ostra [64], a recent unwanted traffic mitigation system, combats unwanted communication by forcing it to traverse social links annotated by credit balances. The per-link credit balances rate-limit unwanted communication. Unlike Ostra, SocialFilter does not use social links to rate-limit unwanted traffic. Instead it utilizes social links to bootstrap trust between reporters, and to suppress Sybil attacks. We perform a comparative evaluation between Ostra's and SocialFilter's approach in leveraging social trust. Ostra uses the social network as a rate-limiting conduit for communication. SocialFilter on the other hand uses the social network as a trust layer from which information on the trustworthiness of spam detectors can be extracted.

We have evaluated our design (Section 3.4) using a $50K$-node sample of the Facebook social network. We demonstrate through simulation that collaborating SocialFilter nodes are able to suppress spam email traffic in a reliable and responsive manner. Our comparison with Ostra shows that our approach is slightly less effective in suppressing spam when the portion of spammers in the network exceeds 1% and when spammers employ more than 100 Sybils each. However, Ostra can result in a non-negligible percentage of legitimate emails being blocked (false positives), which is highly undesirable. This holds even when receivers do not falsely classify legitimate email as spam. In contrast, in this case SocialFilter yields no false positives. Given the severity of the problem of false positives, these results suggest that our system can be a better alternative under a multitude of deployment scenarios.

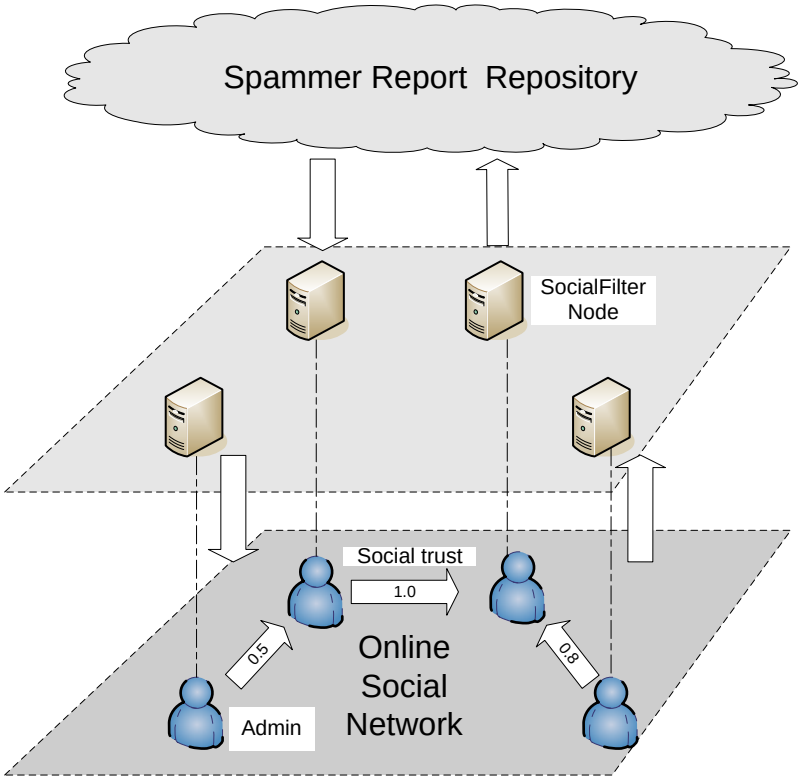

FIGURE 3.1: SocialFilter architecture.

3.2 Overview

In this section, we provide a high-level description of SocialFilter and the security challenges it addresses.

3.2.1 SocialFilter Components

Figure 3.1 depicts SocialFilter's architecture. At a high-level, the SocialFilter system comprises the following components: 1) human users that administer networked devices/networks (*admins*) and join a social network and maintain a unique account; 2) *SocialFilter nodes* (or reporters) that are administered by specific admins and participate in monitoring and reporting the behavior of email senders; 3) *spammer reports* submitted by SocialFilter nodes concerning email senders they observe; and 4) a centralized repository that receives and stores spammer reports, and computes trust values for SocialFilter nodes and their

reports.

The same admin that administers a SocialFilter node also administers a group of applications that interface with the node to report spamming behavior. Interfacing applications can be SMTP servers or IDS systems [68] that register with the SocialFilter repository. SMTP servers can classify spam by using their email characterization functionality (host reputation services such as TrustedSource [15], CloudMark [3] and DShield [95], or content-based filters.) The interfacing application can also be one driven by a human user who reports an email (and consequently its originating email server) as spam.

3.2.2 Spammer Reports

An email characterization application uses the `ReportSpammer(`h`, confidence)` call of the SocialFilter node RPC API to feedback its observed behavior for an email sender h to the node. The first argument identifies the email sender, i.e., an IP address. The second argument is the confidence with which the application is reporting that the specified host is a spammer. The latter takes values from 0% to 100% and reflects the fact that in many occasions traffic classification has a level of uncertainty. For example, a mail server that sends both spam and legitimate email may or may not be a spamming host. For instance, the confidence may be equal to the portion of emails received by host h that are spam [76].

In turn, the SocialFilter node submits a corresponding spammer report to the repository to share its experience with its peers. For example, if a node i's spam analysis indicates that half of the emails received from host with IP h are spam, i reports:

$$[\text{spammer report}]\ h,\ 50\%$$

To prevent forgery of reports and maintain accountability, nodes authenticate with both the repository and the OSN provider using standard single-sign-on authentication techniques, *e.g.*, [41, 83] or Facebook Connect [7].

3.2.3 Determining whether a Host is Spamming

Our system relies on the fact that nodes comprising Internet systems, such as email servers, honeypots or IDS, are administered by human admins. These users maintain accounts in online social networks (OSN.) The SocialFilter centralized repository utilizes two dimensions of trust embedded in OSNs to determine the trustworthiness of the reports submitted by SocialFilter nodes:

- **Reporter trust.** The SocialFilter repository computes *reporter trust* values for all nodes by employing a transitive trust inference mechanism. This mechanism relies on comparing the reports of SocialFilter nodes that are socially acquainted to derive pairwise *direct trust* values (Section 3.3.1.) If two friend nodes i and j have submitted reports concerning the same hosts, the repository can compare their reports to determine the direct trust value d_{ij}. The repository initializes the direct trust d_{ij} to a trust value explicitly submitted by the admin of i. This value is i's assessment on his friend's j ability to correctly maintain its SocialFilter node. Since two friends may initialize their direct trust at distinct values, the direct trust values are not symmetric.

- **Identity uniqueness.** The SocialFilter repository defends against Sybil attacks [39] by exploiting the fact that OSNs can be used for resource testing [92, 64, 85]. The test in question is a Sybil attacker's ability to create and sustain acquaintances. Using a SybilLimit-based [92] technique (Section 3.3.2,) the OSN provider assigns an *identity uniqueness* value to each node. This value quantifies the system's belief in that node being a Sybil.

An application can use the `IsSpammer(h)` call of the SocialFilter node RPC API to obtain a value that quantifies the belief the host h is spamming. The node obtains this value by querying the repository, which derives this value by aggregating spammer reports regarding h. The repository weighs these reports by the reporter trust and identity

uniqueness of the nodes that submitted them.

3.2.4 Assumptions

In designing SocialFilter, we make the following assumptions:

Correctly configured and trustworthy SocialFilter reporters send trustworthy reports and report similarly: We assume that trustworthy SocialFilter admins have correctly configured their spam detection systems, so that their SocialFilter node sends mostly correct reports. We also assume that when they report the same spamming host, their reports mostly match, since a host is expected to send spam to most of the nodes it connects to [90]. In the rest of this paper, we call correctly configured and trustworthy SocialFilter reporters *honest*. We treat non-malicious but incorrectly configured nodes as malicious (Section 3.2.5.)

Trusted repository: We assume that the OSN provider and the SocialFilter repository reliably maintain the social graph, and the spammer reports. We trust the SocialFilter repository to correctly compute the required spammer belief values.

The administrators of the SocialFilter repository have an initial estimate of what portion of the social network is honest: We assume that SocialFilter administrators know approximately the portion of users that is honest. They use this estimate to initialize the trust inference methods employed by SocialFilter.

The administrators of SocialFilter repository have a priori knowledge of fully trusted users in the social network: These users are used to seed the SybilLimit-based identity uniqueness method. They are also used as pre-trusted nodes for the maximum-trust-path-based trust inference method.

Social connections are properly vetted. We assume that social connections between admins have been properly vetted, i.e., when two admins befriend each other in the OSN, this implies that they know each other's ability to correctly maintain their systems. Thus, we do not need to tackle the problem of promiscuous users that connect to other mem-

bers of the OSN indiscriminately, reducing the guarantees offered by the Sybil-detection algorithm.

3.2.5 Threat Model

SocialFilter is a collaborative platform aiming at suppressing malicious traffic. In addition, it is an open system, meaning that any admin with a social network account and a device can join. As such, it is reasonable to assume that SocialFilter itself will be targeted in order to disrupt its operation. Our system faces the following security challenges:

False spammer reports. Malicious SocialFilter nodes may issue false reports aiming at reducing the system's ability to detect spam or disrupting legitimate email traffic.

Direct trust manipulation. The transitive trust scheme used to determine a node's reporter trust is vulnerable to manipulation via false spammer reporting as follows. First, a malicious node may purposely send false spammer reports in order to increase the direct trust between himself and malicious friends that also send false reports. This can result in the malicious friends of the malicious node to have increased reporter trust. Second, a malicious node may purposely send true spammer reports in order to increase its direct trust with honest SocialFilter nodes that send true reports (similar to the camouflage attack Section 2.2.4.) This manipulation can result in the malicious node having increased reporter trust. Third, malicious nodes may purposely send false spammer reports to reduce the direct trust with honest nodes. This can result in the honest nodes having decreased reporter trust. Fourth, a malicious host may send legitimate email to an honest node x and spam email to x's honest friend node y, aiming at decreasing the direct trust between x and y. This manipulation can decrease the reporter trust of x or y.

Sybil attack. An adversary may attempt to create multiple SocialFilter identities aiming at increasing its ability to subvert the system using false spammer reports and direct trust updates. Defending against Sybil attacks without a trusted central authority that issues verified identities is hard. Many decentralized systems try to cope with Sybil attacks by

binding an identity to an IP address. However, malicious users can readily harvest IP addresses through BGP hijacking [70] or by commanding a large botnet.

3.3 SocialFilter Design

We now present SocialFilter's design in more detail.

3.3.1 Reporter Trust

Malicious nodes may issue false spammer reports to manipulate the perceived belief that a host is a spammer. In addition, misconfigured nodes may also issue erroneous spammer reports. SocialFilter can mitigate the negative impact of malicious or incorrect reports by assigning higher weights to reports obtained from more nodes with higher reporter trust.

The repository maintains a reporter trust value $0 \leq rt_i \leq 1$ for each node i managed by an admin in the social graph. This trust score corresponds to the repository's estimation of the belief that node j's reports are accurate. It is obtained from three sources: a) manual trust assignments between friends in the social networks; b) spammer report comparison; and c) transitive trust.

To derive trust values, the repository needs to maintains the social graph $\mathcal{S}(\mathcal{V},\mathcal{E})$ of the admins in the SocialFilter system. $\mathcal{V}$ denotes the set of the admins and $\mathcal{E}$ denotes the set of the friend connections between socially acquainted admins. The repository also maintains a reporter trust graph $T(\mathcal{V},E)$. The vertices of this graph is the set of all SocialFilter admins as is the case for graph $\mathcal{S}(\mathcal{V},\mathcal{E})$. The edges E are the edges in $\mathcal{E}$ annotated with *direct trust* values between acquainted SocialFilter nodes. Next, we describe how the direct trust values are derived and how the reporter trust values are computed using $T(\mathcal{V},E)$.

User-defined trust. First, to initialize the direct trust values, the repository relies on the fact that nodes are administered by human users. Competent and benign users are likely to maintain their nodes secure, and provide honest and truthful reports. Moreover, admins that are socially acquainted can assess each other's competence. An admin i tags his acquaintance admin j with a *user-defined trust* score $0 \leq ut_{ij} \leq 1$ based on his belief on

j's ability to correctly configure his node. The repository uses this value to initialize the direct trust value between friend nodes i and j: $d_{ij} = ut_{ij}$. Users frequently use the OSN to add friends and to communicate with each other, thus the requirement for administrators to rate each other should not induce a substantial usability burden.

Spammer reports comparison. Second, the repository dynamically updates the direct trust d_{ij} by comparing spammer reports submitted by two nodes i and j. The spammer reports of two friend nodes i and j can be compared if both nodes have reported on the same host h. Intuitively, if i and j share similar opinions on h, i should place high trust in j's reports. Let $0 \leq v_{ij}^k \leq 1$ be a measure of similarity between i and j's k_{th} report on a common host. The repository updates i's direct trust to j using an exponential moving average:

$$d_{ij}^{k+1} = \alpha * d_{ij}^k + (1-\alpha) * v_{ij}^{k+1} \tag{3.1}$$

As i and j submit more common reports, the direct trust d_{ij}^k gradually converges to the similarity of reports from i and j. α is a system parameter that affects the influence of history on direct trust assessment.

Transitive trust. Third, the repository incorporates direct trust and transitive trust [46, 47] to obtain the reporter trust value for i: rt_i. It does so by analyzing the reporter trust graph $T(\mathcal{V}, E)$ from the point of view of a small set of pre-trusted nodes in $\mathcal{V}$. These pre-trusted nodes are administered by competent admins that are fully trusted by the Social-Filter repository.

We use transitive trust for the following reasons: a) due to the large number of nodes, the admin of a pre-trusted node i cannot assign a user-defined trust ut_{ij} to every admin of a node j, as he may not know him; b) due to the large number of email-sending hosts, a pre-trusted node i may not have encountered the same hosts with another node j, thus the repository may be unable to directly verify j's reports; and c) even if a pre-trusted node i

has a direct trust value for another node j, the repository can improve the correctness of rt_j by learning the opinions of other SocialFilter nodes about j.

The overall reporter trust rt_j can be obtained as the maximum trust path between a pre-trusted node i and the node j in the trust graph $T(\mathcal{V},E)$. That is, for each path $p \in P$, where P is the set of all paths between nodes the pre-trusted node and j:

$$rt_j = max_{p \in P}(\Pi_{u \to v \in p} d_{uv}) \tag{3.2}$$

The above reporter trust value is computed from the point of view of a single pre-trusted node. We repeat the above process for every pre-trusted node. We then average the reporter trust values for all pre-trusted nodes to derive a final rt_j value. We use multiple pre-trusted nodes to ensure that there is a trust path from a pre-trusted node to most honest nodes j. We also use many pre-trusted nodes to limit the influence of attackers that manage to establish a high trust path with one of the pre-trusted nodes.

We use the maximum trust path because it can be efficiently computed with Dijkstra's shortest path algorithm in $O(|E|\log|\mathcal{V}|)$ time for a sparse $T(\mathcal{V},E)$. In addition, it yields larger trust values than the minimum or average trust path, resulting in faster convergence to high confidence on whether a host is spamming. Finally, it mitigates the effect of malicious nodes that have low direct trust values towards honest nodes.

We choose to record and consider the direct trust values only for pairs of nodes for which the admins are friends. By doing so, we prevent malicious nodes from establishing high direct trust values with many honest nodes, aiming at increasing their reporter trust (the camouflage attack mentioned in Section 2.2.4.) Furthermore, by considering direct trust only over social edges we keep the trust graph sparse, thus the cost of the maximum trust path computation decreases.

We compute the maximum trust path from the pre-trusted nodes to all other nodes periodically to reflect changes in direct trust values.

Direct Trust Manipulation Attack

Our design is inherently resilient to the direct trust manipulation attack mentioned in Section 3.2.5. We discuss each manifestation of this attack in turn using the enumeration used in Section 3.2.5.

The reporter trust mechanism by itself does not defend against attack (a). To tackle this attack when the malicious friends are Sybils, we incorporate an additional trust mechanism as described in Section 3.3.2. By performing attack (b) a malicious node may increase its reporter trust, but in doing so it will have to submit truthful spammer reports. The attack (c) is effective only if the maximum trust path to the targeted honest node passes through the malicious node. If there is at least one alternative trust path that yields a higher trust value, then the direct trust value between the malicious and the honest node is ignored. The attack (d) can be effective only if the malicious host has a legitimate reason to send email to the targeted honest nodes.

Bayesian Interpretation of Reporter Trust

Note that the max trust path metric described above has a partially Bayesian interpretation. The direct trust edge d_{uv} may correspond to the probability that u assigns to v to correctly assign direct trust to other users or to correctly classify spamming hosts. Thus, the maximum trust path from the pre-trusted nodes to a node i (rt_i) quantifies the belief that user i will correctly classify spam, as perceived by the pre-trusted node. It is derived by multiplying the correct classification probabilities that each intermediate node along the maximum trust path assigns to the next node.

3.3.2 OSN Providers as Sybil Mitigating Authorities

For an open system such as SocialFilter to operate reliably, prevention of Sybil attacks is of the utmost importance. We propose to leverage existing OSN repositories as inexpensive Sybil-mitigating authorities. OSNs are ideally positioned to perform such a function:

using SybilLimit-like [92] techniques (Section 3.3.2), OSNs can approximate the belief that a node's identity is not a Sybil. We refer to this belief as *identity uniqueness*.

Each node that participates in SocialFilter is administered by human users that have accounts with OSN providers. The system needs to ensure that each user's social network identity is closely coupled with its SocialFilter node. To this end, SocialFilter employs single sign-on authentication mechanisms, such as Facebook Connect [7], to associate the OSN account with the spammer report and direct trust update repository account.

Identity Uniqueness

When malicious users create numerous fake online personas, SocialFilter's spammer belief measure can be subverted. Specifically, a malicious user a with high reporter trust may create Sybils and assign high direct trust to them. As a result, all the Sybils of the attacker would gain high reporter trust. The Sybils can then submit reports that greatly affect the spammer belief values.

Fortunately, existing algorithms such as SybilGuard and SybilLimit [93, 92] can detect Sybil attackers on social graphs. These algorithms take advantage of the feature that most social network users have a one-to-one correspondence between their social network identities and their real-world identities. Malicious users can create many identities or connect to many other malicious users, but they can establish only a limited number of trust relationships with real users. Thus, clusters of Sybil attackers are likely to connect to the rest of the social network with a disproportionately small number of edges, forming small quotient cuts.

SocialFilter adapts the SybilLimit algorithm to determine an identity-uniqueness score $0 \leq id_i \leq 1$ for each node i. This score indicates the belief that the administrator of node i corresponds to a unique user in real life and thus is not part of a network of Sybils. To be Sybil-resistant, SocialFilter multiplies the identity-uniqueness score id_i by the reporter trust to obtain the trustworthiness of node i's spammer reports.

SybilLimit is designed to operate in a decentralized setting in which nodes are not aware of the complete social graph. We use a stripped-down centralized version of SybilLimit, because in our setting the OSN provider has complete knowledge of the social graph's topology. We now describe in detail how we compute id_i.

First, we provide a brief background on the theoretical justification of SybilLimit. It is known that randomly-grown topologies such as social networks and the web are fast mixing small-world topologies [88, 26]. Thus, in the social graph $\mathcal{S}(\mathcal{V}, \mathcal{E})$, the last edge (also referred to as the tail) traversed by a random walk of $\Theta(\log|\mathcal{V}|)$ steps is an independent sample edge approximately drawn from the stationary distribution of the graph. If we draw $\Theta(\sqrt{|\mathcal{E}|})$ $\Theta(\log|\mathcal{V}|)$-long random walks from a legitimate verifier node v and a legitimate suspect node s, it follows from the generalized Birthday Paradox that the sample tails intersect with high probability. The opposite holds if the suspect resides in a region of Sybil attackers. This is because the Sybil region is connected via a disproportionally small number of edges to the region of legitimate nodes. Consequently, the tails of random walks from the Sybil suspect are not samples from the same distribution as the tails of random walks from the verifier.

SybilLimit [92] replaces random walks with "random routes" and a verifier node v accepts the suspect s if random routes originating from both nodes intersect at the tail. In random routes, each node uses a pre-computed random permutation as a one-to-one mapping from incoming edges to outgoing edges. Each random permutation generates a unique routing table at each node. As a result, two random routes entering an honest node along the same edge will always exit along the same edge ("convergence property"). This property guarantees that random routes from a Sybil region that is connected to the honest region through a single edge will traverse only one distinct path, further reducing the probability that a Sybil's random routes will intersect with a verifier's random routes.

With SocialFilter's SybilLimit-based technique, the OSN provider computes an identity uniqueness score for each node s in the social graph $I(\mathcal{V}, \mathcal{E})$. At initialization time, the

OSN provider selects l pre-trusted verifier nodes. It also creates $2r$ independent instances of pre-computed random permutation as a one-to-one mapping from incoming edges to outgoing edges (routing table). The first $r = \Theta(\sqrt{|\mathcal{E}|})$ routing tables are used to draw random routes from suspect nodes s and the rest r routing tables are used to draw random routes from the verifier nodes v. For each s, the OSN provider runs the SybilLimit-like algorithm is as follows:

1. For each of the l verifiers v, it picks a random neighbor of v. It draws along the random neighbors r random routes of length $w = \Theta(\log|\mathcal{V}|)$, for each instance of the $r = \Theta(\sqrt{|\mathcal{E}|}$ routing tables. It stores the last edge (tail) of each verifier random route.

2. It picks a random neighbor of s and draws along it r random routes of length $w = \Theta(\log|\mathcal{V}|)$, for each instance of the nodes' routing tables. It stores the tail of each suspect random route. We refer to steps (1) and (2) of the algorithm as *random routing*.

3. For each verifier v, if one tail from s intersects one tail from v, that verifier v is considered to "accept" s. We refer to this step as *verification.*

4. It computes the ratio of the number of verifiers that accept s over the total number of verifiers l. That ratio is the computed identity uniqueness score id_s.

Nodes query the OSN provider for the identity uniqueness of their peers. The OSN provider performs the above computations periodically and off-line to accommodate for topology changes.

3.3.3 Spammer Belief

We now describe how we combine reporter trust, identity uniqueness and spammer reports to derive a measure of the belief that a host is spamming.

Definition 3.3.1. *We define* spammer belief *as a score in* 0% *to* 100% *that can be interpreted as the belief that a host is spamming: a host with* 0% *spammer belief is very unlikely to be a spammer, whereas a host with* 100% *spammer belief is very likely to be one.*

Spammer Reports

A node i may have email classification functionality through applications that interface with it using the i's `ReportSpammer()` API. In this case, i considers only the reports of those applications in calculating the belief that a host is spamming. When i receives spammer reports by more than one applications for the same h, i's confidence $c_i(h)$ that h is a spammer is the average (possibly weighted) of these applications' reports. Node i uses this average confidence to compute the similarity of its reports with the reports of members of its view, which is used to derive direct trust scores.

At initialization time, SocialFilter nodes consider all hosts to be legitimate (0% confidence in the hosts being spammers). As nodes receive emails from hosts, they update their confidence (3.2.2). For efficiency, nodes send spammer report to the repository only when the difference between the previous confidence in the node being a spammer and the new confidence exceeds a predetermined threshold δ.

When a node i receives a new spammer report for h, this new report preempts an older report, which is thereafter ignored. Consequently, SocialFilter nodes are able to revoke spammer reports by updating them. Each spammer report carries a timestamp. The time interval during which a spammer report is valid is a tunable system parameter. Reports that have expired are not considered in the calculation of the belief that a host is spamming. We assume loose synchronization between SocialFilter nodes.

Spammer Belief Equation

A node i that does not have email classification functionality may receive multiple spammer reports originating from multiple nodes $j \in V_i$ and concerning the same host h. Subse-

quently, i needs to aggregate the spammer reports to determine an overall belief `IsSpammer(h)` that h is a spammer. Node i derives the spammer belief by weighing the spammer reports' confidence with the reporter trust and identity uniqueness of their reporters:

$$IsSpammer(h) = \frac{\Sigma_{j \in V_i^h} \, rt_j \, id_j \, c_j(h)}{S} Logistic(S) \tag{3.3}$$

In the above equation, $V_i^h \subseteq V_i \setminus i$ is the set of members in i's view that have posted a spammer report for h. In addition, $S = \Sigma_{j \in V_i^h} \, rt_j \, id_j$.

The factor $0 \leq Logistic(S) \leq 1$ discounts the belief in a host h being spammer in case the reporter trust and identity uniqueness of the nodes that sent a spammer report for h is low. It is used to differentiate between the cases in which there are only a few reports from non-highly trustworthy nodes and the cases there are sufficiently many and trustworthy reports. When S is sufficiently large, we should consider the weighted average of the confidence in the reports to better approximate the belief that a host is spammer. But when S is small we cannot use the spammer reports to derive a reliable spammer belief value. Based on these observations, we define the function *Logistic* as the logistic (S-shaped) function of S:

$$Logistic(S) = \frac{1}{1 + e^{b(1-S)}} \tag{3.4}$$

b is a small constant set to 5 in our design. For $S \leq 0.4$, $Logistic(S)$ is very small. However, when S exceeds 0.6, $Logistic(S)$ increases drastically until it becomes 0.5 for $S = 1$. For $S = 2$, $Logistic(S)$ approximates 1.

3.3.4 SocialFilter Repository

Practice has shown that centralized email infrastructure such as web mail providers and email reputation services can scale to millions of clients. Thus, to simplify the design and provide better consistency and availability assurances we use a centralized repository.

This repository can in fact consist of a well-provisioned cluster of machines or even a data-center.

When a node queries the repository for the spammer belief of a host, the repository is interested on the reports for a single host. These reports are sent by multiple nodes, thus for efficiency it is reasonable to index(key) the reports based on the hash of the host's IP.

3.3.5 SocialFilter Operation Example

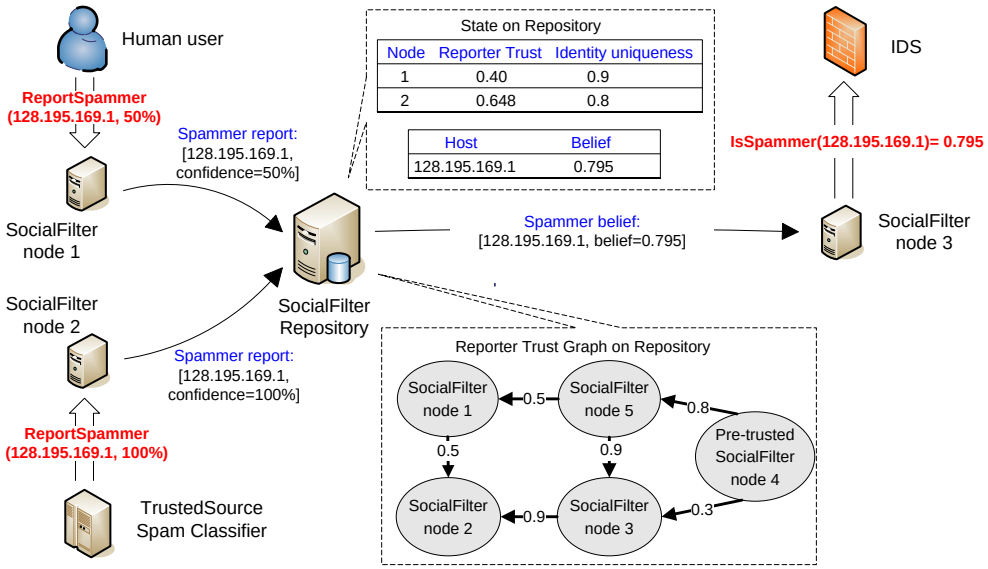

FIGURE 3.2*:* Example of the operation of a small SocialFilter network.

Figure 3.2 depicts an example of the operation of a small SocialFilter network. The network includes an IDS node tasked with checking incoming TCP connections for whether they originate from spamming hosts, SocialFilter node 3. That node has no inherent email classification functionality, thus it relies on the other two nodes, 1 and 2, for early warning about spam bots. Node 1 relies on human users to classify emails as spam. In this example, the human user has classified half of the emails originating from host with IP=128.195.169.1 as spamming, therefore it reports that this host is spamming with confidence $c_1(IP) = 50\%$. Node 2 is an email server that has subscription to the proprietary

TrustedSource email reputation service. In this example, Node 2 receives a connection request from the host with IP=128.195.169.1, it queries the email reputation service and gets a response that this host is spamming. Thereby, node 2 reports with confidence $c_2(IP) = 100\%$ confidence that the host is a spam bot.

The repository maintains the depicted reporter trust graph. The trust graph includes the pre-trusted node 4. It also includes nodes 1 and 2, which send the depicted spammer reports. Furthermore it includes node 3 and node 5, which do not send any reports in this example. The weighted directed edges in the graph correspond to the direct trust between the nodes in the trust graph. From the reporter trust graph and Equation 3.2, the maximum trust path between nodes 4 and 1 traverses nodes 5 and 1 yielding reporter trust $rt_1 = 0.4$. The maximum trust path between 4 and 2 traverses nodes 5, 3 and 2 and yields reporter trust $rt_2 = 0.648$. The identity uniqueness of nodes 1 and 2 has been computed by the OSN provider to be $id_1 = 0.9$ and $id_2 = 0.8$, respectively. The IDS can now call `IsSpammer(128.195.169.1)` on node 3 to determine the belief that the host is a spammer. To compute this belief, we use Equation 3.3:

$$\frac{rt_1 id_1 c_1(IP) + rt_2 id_2 c_2(IP)}{rt_1 id_1 + rt_2 id_2} = 79.5\%$$

3.4 Evaluation

We evaluate SocialFilter's ability to block spam traffic and compare it to Ostra [64]. Ostra represents a different approach to spam mitigation using social links. The goal of our evaluation is two-fold: a) to illustrate the importance of our design choices, namely incorporating identity uniqueness and initializing direct trust with user-defined trust; and b) to shed light on the benefits and drawbacks of SocialFilter's and Ostra's approach in using social links to mitigate spam. We also evaluate the computation cost of SocialFilter on the centralized repository.

3.4.1 Ostra Primer

Before we proceed with the comparative evaluation, we briefly describe Ostra to provide insights on its operation. Ostra bounds the total amount of unwanted communication a user can send based on the number of social trust relationships the user has and the amount of communication that has been flagged as wanted by its receivers. Similar to SocialFilter, in Ostra an OSN repository maintains the social network. When a sender wishes to establish an email connection to a receiver, it first has to obtain a cryptographic token from the OSN repository. The OSN repository uses the social links connecting the admins of the sender and the receiver nodes to determine whether a token can be issued.

In particular, a node is assigned a credit balance, B, for each social link it's administrator is adjacent to. B has an initial value of 0. Ostra also maintains a per-link balance range $[L,U]$, with $L \leq 0 \leq U$, which limits the range of the users credit balance (i.e., always $L \leq B \leq U$). The balance and balance range for a user is denoted as B_L^U. For instance, the link's adjacent user's state 2_{-4}^{+5} denotes that the user's current credit balance is 2, and it can range between -4 and 5.

When a communication token is issued, Ostra requires that there is a path between the sender and the receiver in the social network. Subsequently, for each link along the social

path the first adjacent nodes credit limit L is increased by one, and the second adjacent nodes credit limit U is decreased by one. This process propagates recursively from the sender to the receiver along the social links. If this process results in any of the links in the path to have adjacent nodes of which the credit balances exceed the balance range, Ostra refuses to issue the token. When the email connection is classified by the receiver, the credit limits L and U are restored to their previous state. If the connection is marked as unwanted, one credit is transferred from the balance of the first node of the link to the balance of the second one.

As a consequence of this design, the social links that connect spammers to their receivers eventually have balance beyond the allowed range, and a spammer is prevented from sending further emails. In addition, Ostra is Sybil-resilient because the credit available to a sender is not dependent on the number of Sybils it has. It is only dependent on the sender's connectivity in the social network and on whether the sender's emails are classified as wanted.

3.4.2 Evaluation Settings

For our evaluation, we use a large connected component sampled from the Facebook social network as described in Section 2.6.1. Our sample consists of 50K users and 442,772 symmetric links. The average number of friends of each user in the graph is approximately 18. The diameter of this graph is 11. The clustering coefficient is 0.178. Each user in the social network is the admin of an email server, which we also refer to as a SocialFilter or Ostra node. Nodes can send and receive email connections.

We use the SimPy 1.9.1 [66] discrete-event simulation Python framework to simulate the operation of SocialFilter and Ostra. We do not simulate physical, network or transport layer events (e.g. congestion and packet loss). For efficiency, the identity uniqueness and reporter trust modules are coded in C++.

We have two types of nodes: honest and spammers. Honest nodes send 3 legitimate

emails per day. 80% and 13% of the legitimate emails are sent to sender's friends and sender's friends of friends respectively, and the destination of the rest 7% emails is randomly chosen by the sender. Spammers send 500 spam emails per 24h, each to random honest nodes in the network. We set Ostra's credit bounds as $L = -5$ and $U = 5$. The above settings are obtained from Ostra's evaluation [64]. Honest and spammer nodes correspond to users uniformly randomly distributed over the social network.

Several nodes can instantly classify spam connections. These instant classifiers correspond to systems that detect spam by subscribing to commercial blacklists or by employing content-based filters. On the other hand, normal nodes can classify an email only after receiving it and their users read the email. That is, the normal classification can be delayed based on the behavior of the users (how frequently they check their email). In our evaluation, 10% of honest SocialFilter nodes have the ability of instant classification and the average delay of the normal classification is 2 hours [64].

In SocialFilter, when a node classifies received email as spam, it issues a spammer report as $\{$[spammer report] h, c(h)$\}$, where h is the IP of the email sender and $c(h)$ is the node's confidence. The issued spammer reports are gathered and aggregated in the repository. When normal users with no capability of instant classification receive SMTP connection requests from previously unencountered hosts they query the repository. Subsequently, the repository returns to them a value that corresponds to the belief that a host is a spammer (Section 3.3.3.) In summary, classifier nodes share their experiences by issuing spammer reports, and normal nodes use the reports to block spam from senders they had not previously encountered.

The reporter trust assigned to nodes is computed using Dijkstra's algorithm based on the pairwise direct trust value between users that are connected in the 50K-node social graph (Section 3.3.1.) The pairwise direct trust values are derived using Equation 3.1. The direct trust between users that are friends is initialized to a random value in $[0,\ 1]$. The number of pre-trusted nodes used is 100 and we recompute the reporter trust every 24

simulated hours.

In this evaluation, we compute the similarity between reports using Equation 3.1 with $\alpha = 0.8$. Assume that node i receives the k^{th} spammer report from node j that involves a host h that both nodes have observed and to which node i and j have assigned confidence $c_i(h)$ and $c_j(h)$, respectively. The repository computes the similarity v_{ij}^k as follows:

$$v_{ij}^k = \frac{min(c_i(h), c_j(h))}{max(c_i(h), c_j(h))} \tag{3.5}$$

The identity uniqueness of each node is computed as described in Section 3.3.2 by processing the 50K-node social graph. The parameters of the computation are set as follows: $w = 15$, $r = 2000$ and $l = 100$. If the overall spammer belief computed by Equation 3.3 is over 0.5, a node blocks the SMTP connection.

3.4.3 Importance of User-defined Trust

In this portion of our evaluation, we demonstrate the importance of using the user-defined trust score (Section 3.3.1) to initialize the direct trust between nodes.

In Figure 3.4.4, we show the percentage of blocked spam and legitimate email connections for SocialFilter with varying lengths of simulated time. Figure 3.4.4 also shows the spam mitigation capability of SocialFilter when the initial user-defined (UD) trust assigned by friends in the SocialFilter admin social network is 0 ("SF-Spam-without UD trust"). As can be seen, SocialFilter with direct trust initialized with user-defined trust is effective in blocking 99% of spam emails after 85h. On the other hand, when the user-defined trust is 0, it takes a lot more time (up to 340h) for SocialFilter to start effectively blocking spam.

When the user-defined trust is not used to initialize direct trust, after 85h of simulated time a SocialFilter node has on average only 0.22 reporter trust. This is because early in the simulation, nodes have encountered a small number of common spamming hosts, thus the repository cannot derive meaningful direct trust values. Consequently the reporter

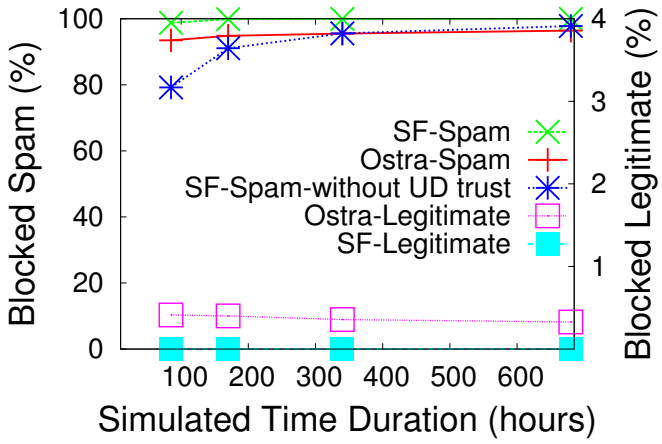

FIGURE 3.3*:* Percentage of blocked spam and legitimate emails connections for SocialFilter (SF) and Ostra as a function of simulated time length. The percentage of spammer nodes is 0.5%.

trust graph $T(\mathcal{V},E)$ is disconnected, resulting in low report trust scores. Consequently, the repository is unable to consider many valid spammer reports from the nodes. As time progresses, the repository can derive more meaningful direct trust values by comparing the reports of friend nodes. Therefore, we observe that after 170h SocialFilter is able to block almost 100% of spam.

This result validates our design choice to tap into the user-defined trust between acquainted SocialFilter admins. This source of trust is important because it enables the initial trust graph to be sufficiently connected. These trust values can be derived without prior spammer report comparisons. User-defined trust also contributes in trust values converging to correct ones faster (given that the admins have assigned appropriate values), even in case common spammer reports are infrequent.

3.4.4 Resilience to Spammers

In Figure 3.4.4, we depict SocialFilter's and Ostra's spam mitigation effectiveness with varying simulated time duration. We observe that SocialFilter manages to block 99% to 100% after 179h of simulated time. Once the repository has obtained sufficiently trustworthy spammer reports from nodes, it can inform all other nodes about the detected

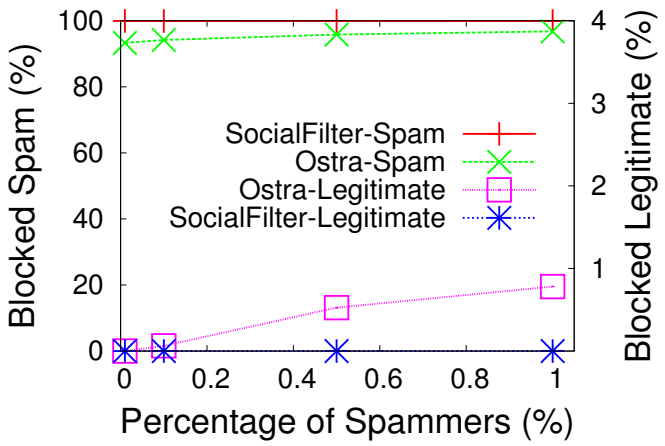

FIGURE 3.4*:* Percentage of blocked spam and legitimate email connections for SocialFilter (SF) and Ostra as a function of the percentage of spammer nodes. The simulated time duration is 340h.

spammers. In Ostra, after the percentage of blocked spam reaches 95% at 340h, it does not improve with the passage of time. We attribute this difference in effectiveness on the fact that in Ostra spam detection affects only a region of the social network: the one that is affected by the change in the credit balances of the links adjacent to the detector node. On the other hand, in SocialFilter once a sufficiently trustworthy node detects spam, its report can be used by all other nodes in the network to classify the spamming host. Importantly, we also observe that Ostra suffers from a non-negligible false positive rate (blocked legitimate emails), which is equal to $\sim 0.4\%$. In contrast, SocialFilter yields no false positives.

Figure 3.4.4 presents the spam mitigation effectiveness of SocialFilter and Ostra under a varying number of spammers. We make two observations. The first is that as in Figure 3.4.4, Ostra suffers from a substantial false positive rate when the percentage of spammers is greater than 0.1%. When the percentage of spammers is 1% (500 spammers), around 0.8% of legitimate emails are blocked. We can attribute Ostra's high false positive rate to the following. In SocialFilter, a node blocks an email sender only if it has been explicitly reported as spammer by a member of its view. On the other hand, Ostra blocks links (credit balance goes out of bounds) in the socials path used by a spammer, and some

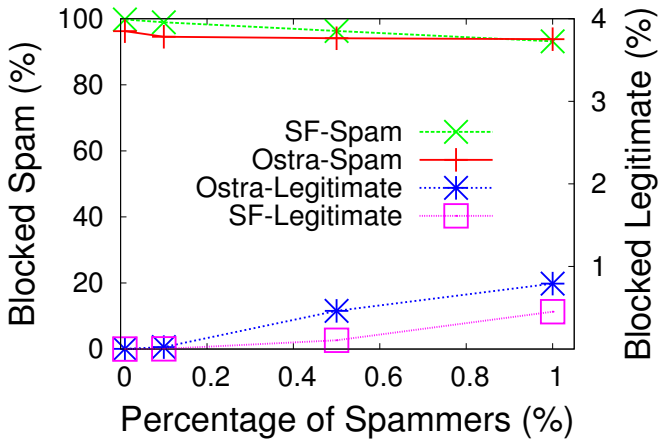

FIGURE 3.5*:* Percentage of blocked spam and legitimate email connections for SocialFilter (SF) and Ostra as a function of the portion of colluding spammers.

honest nodes cannot send email because those links are included in all the social paths used by those honest nodes.

The second observation is that SocialFilter always blocks $\sim 99\%$ of spam as the portion of spammers varies in 0.1% to 1%. Ostra always blocks 93% to 97% of spam connections. We observe that the spam detection rate increases substantially for Ostra and slightly for SocialFilter as the number of spammers increases. This is because the increased number of spam events induces nodes to share more information. As a result, the reporter trust graph becomes more connected in the case of SocialFilter, allowing the repository to consider reports from more nodes as trustworthy. In the case of Ostra, it reduces the balance on social links adjacent to spammers resulting in less spam passing through.

3.4.5 Resilience to Colluding Spammers and Sybils

We also consider attack scenarios under which spammers collude to evade SocialFilter and Ostra, as well as to disrupt email communication from legitimate hosts. We assume that spammers are aware of each other, which is reasonable if the spammers belong to the same botnet. In particular to attack SocialFilter, a spammer submits a report $\{[\text{spammer report}]\ s,\ 0\%\}$ for each of the other spammers in the network. Also, when a

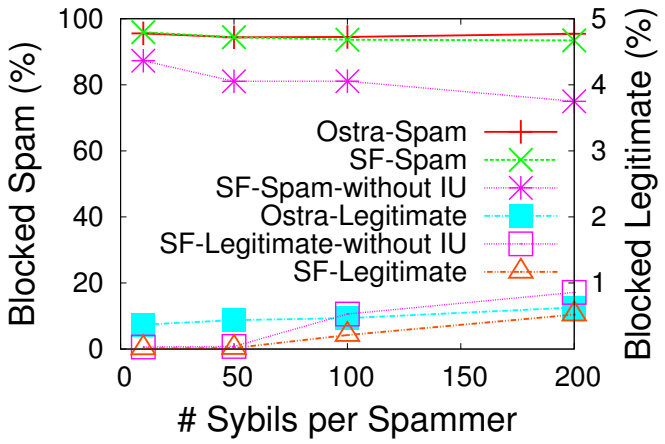

FIGURE 3.6*:* Percentage of blocked spam and legitimate email connections for SocialFilter (SF) and Ostra as a function of the number of Sybils created per spammer. The percentage of spammer nodes is 0.5%. Results for SocialFilter that does not employ identity uniqueness (IU) are also included. The simulated time is 340h.

spammer receives a connection from a legitimate host l, it submits $\{[\text{spammer report}]\ l,\ 100\%\}$ to induce SocialFilter to block h's emails. To attack Ostra, each spammer classifies a legitimate email and a spam email connection as unwanted and legitimate, respectively.

Figure 3.4.5 shows the percentage of blocked spam and legitimate email connections in SocialFilter and Ostra as a function of the percentage of nodes in the network that are colluding spammers. Ostra achieves almost the same effectiveness in blocking spam connections as in the absence of colluding spammers. However, the false positive rate (percentage of blocked legitimate email) increases substantially with the percentage of colluders. Since Ostra does not have any method to recognize false classification, it is more adversely affected by it.

As can be seen in Figure 3.4.5, SocialFilter is less effective in blocking spam email than in the absence of false classification. In fact, when the percentage of colluding spammers reaches 1%, SocialFilter becomes slightly less effective than Ostra. Moreover, the existence of false reporters that incriminate legitimate senders results in a non-zero false positive rate for SocialFilter, which however is substantially less than for Ostra. The rea-

son is that colluding spammers have very low direct trust to other honest users as their reports are different to those honest nodes. As a result, the reporter trust for spammers is lower, resulting in their reports to be mostly ignored by honest nodes.

Sybil Attack

We also consider the case in which colluding spammers create Sybil nodes. These Sybil nodes form a cluster that is directly connected only to their creator spammer node. The purpose of the Sybils is to decrease the reported by the repository belief in the spammer node being spammer, to increase the belief in an honest node being spammer and to send spam messages from many different sources. The latter increases substantially the number of spammers, rendering their classification more challenging.

At the start of the SocialFilter simulation, Sybils send positive spam reports for all other spammer nodes (including the Sybils). Honest nodes may send legitimate email to spammer nodes but not to their Sybils, whom they are not aware of. When a spammer node receives legitimate email from an honest node, the spammer reports the good user as a spammer and so do all the Sybils of the spammer. 10% of all Sybils act as spammers. sending spam messages at the same rate as their creator. In the simulation for Ostra, Sybil nodes classify a legitimate email and a spam email connection as unwanted and legitimate, respectively.

Figure 3.4.5 shows the percentage of blocked spam and legitimate email connections as a function of the number of Sybils per spammer in the network. In SocialFilter, Sybil users gets very low identity uniqueness, which becomes even lower as the number of Sybil users per spammer increases. As a result, we can see in Figure 3.4.5 that SocialFilter is resilient to this attack. In Ostra, Sybil spammers cannot send spam because the few social links that connect the creator of the Sybils with the rest of the network become blocked. We observe that when each spammer creates more than 100 Sybils, Ostra is able to block more spam than SocialFilter. However, Ostra still suffers from higher false positive rate.

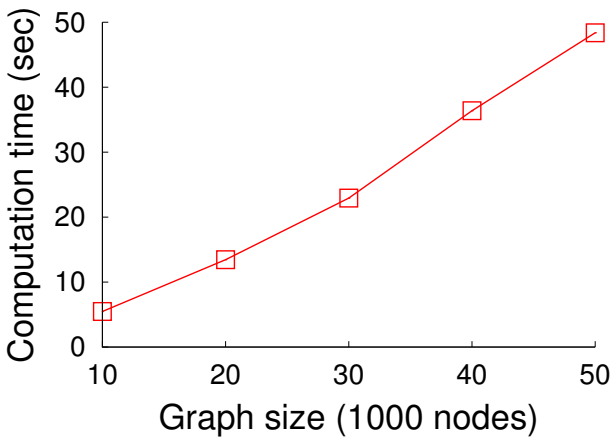

FIGURE 3.7*:* Computation cost of reporter trust as a function of social graph size for 100 pre-trusted nodes.

Importance of Identity Uniqueness

Figure 3.4.5 also shows the case in which SocialFilter does not employ identity uniqueness ("SF-Spam/Legitimate-without IU"). As can be seen, attackers are very effective in manipulating the system in this case. SocialFilter without identity uniqueness is unable to block a substantial percentage of spam, while it blocks a high percentage of legitimate email. This result profoundly illustrates the importance of integrating identity uniqueness in the spammer belief computation (Equation 3.3.)

3.4.6 Computation Cost of Trust Inference

Before comparing SocialFilter and Ostra, we investigate the computation cost of SocialFilter's trust inference mechanisms - reporter trust (Section 3.3.1) and identity Uniqueness (Section 3.3.2)- with respect to the size of the social and trust graphs being analyzed. For the measurement, we use an Intel Core Duo P8600, 2.4GHz CPU, 3MB L2 cache, 4GB RAM machine, and we use the trust computation algorithms implemented in C++.

The repository needs to periodically compute the identity uniqueness and the reporter trust of each node in the social graph. The reporter trust is computed each time the direct trust values in the trust graph change substantially (Section 3.3.1). It is computed by

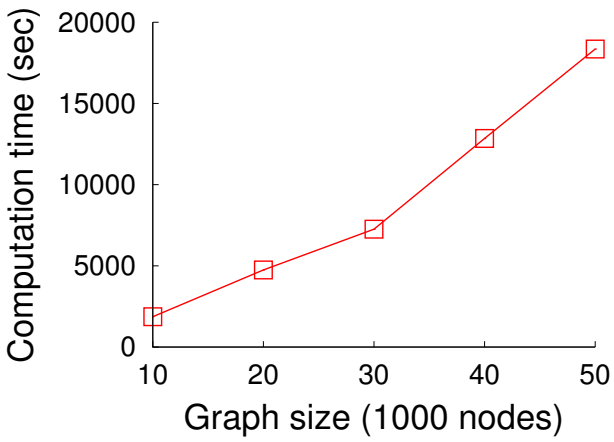

FIGURE 3.8*:* Computation cost of identity uniqueness as a function of social graph size for 100 verifiers.

using Dijkstra's algorithm and the complexity of this computation on the sparse trust graph graph $T(V,E)$ is $O(|E|\log|V|)$. In Figure 3.4.6, we show the computation time of the reporter trust with varying trust graph size. As the size of the network increases, the computation time increases almost linearly, which indicates this computation can scale to large trust graphs. We observe that the computation is relatively inexpensive. However, this is conditioned on the fact that this computation does not take place too often.

The identity uniqueness computation is the most demanding on the repository. This computation costs $O(|V|\sqrt{E}\log|V|)$. It is also linearly dependent on the number of verifiers. It is approximately 400 times (in the same order of magnitude as the $\sqrt{E}$ factor) more computationally expensive than the reporter trust computation. As can be seen in 3.4.6, the cost increases super linearly to the size of the network. Since the social graph does not change as often as the trust graph, this computation can be performed substantially less often that the reporter trust one, *e.g.*, once a day.

The above results indicate that these computations can be performed in reasonable amounts of time by computer clusters with adequate memory and processing power even for million-node email server networks.

4

Related Work

4.1 Overview

A variety of systems have employed trust in social networks to improve system security [69, 81, 71, 59, 91, 85, 37, 92, 84, 75, 69, 43, 99, 51]. To the best of our knowledge, FaceTrust is the first work that proposes to use OSNs to provide relaxed credentials for online personas. In addition, SocialFilter is the first collaborative spam mitigation system to employ Sybil-resilient trust inference to assess the trustworthiness of spamming host reporters.

We now discuss in detail prior work that is related to the proposed FaceTrust and SocialFilter systems.

4.2 Prior Work Pertinent to FaceTrust

FaceTrust improves upon a max-flow-based trust inference method [60] by making its computational cost independent of the number of trusted seeds. However, our contributions are not limited to the trust inference method. Instead, they primarily lie in the novel idea of using OSNs to provide lightweight, extensible, and relaxed identity credentials,

and the overall design and preliminary evaluation of FaceTrust. Table 4.2 summarizes the comparison of prior work with FaceTrust. Next, we classify related work based on its similarities with the problem FaceTrust is addressing and the techniques it uses.

Table 4.1: Comparison of Sybil-resilient trust inference systems and the problems they address.

	Trust Inference	**Problem**	**Cost**	**Assumptions**
FaceTrust	MaxTrust	Veracity of Identity Assertion	$O(T_{max}\|E\|\log\|V\|)$	Known seed and # honest nodes
PGP Web of Trust	[61, 73]	Trustworthiness of Key/Identity binding	$O(T_{max}\|E\|)$ per node	Known seed and # honest nodes
SybilLimit	Birthday Paradox	Whether a node is Sybil	$O(\|V\|\sqrt{\|E\|}\log\|V\|)$	Known seed and fast-mixing
Advogato	Group Max flow	Trustworthiness of a user	$O(\|V\|\|E\|)$	Known seed and, # honest nodes
Sumup	Group Max flow	Whether a user can vote	$O(\|V\|\log\|V\|)$	Known collector and # honest nodes
Ostra	Pairwise Credit Balances	Whether a user can send email or vote	Unspecified	Known collector
SybilInfer	Bayesian Inference	Probability of a user being Sybil	$O(\|V\|^2\log\|V\|)$	known seed and fast-mixing
TrustRank	Eigenvector	Rank of a user in a network	$O(\|V\|\log\|V\|)$	known seed and fast-mixing

4.2.1 Similar Problem - Similar Techniques

Social Web of Trust

The PGP Web of Trust [99, 82, 16] uses the social network to determine how trustworthy is the binding between a public key certificate and an identity. The edges in the trust graph of PGP encode two distinct trust values that users ascribes to each other by signing their certificates: a) trustworthiness of a public key certificate, which reflects how much a user trusts that a public key certificate belongs to the identity designated on the certificate; b) trustworthiness of the introducer, which reflects how much a user trusts the owner of a public key certificate to be a competent assigner of trustworthiness. Several trust graph analysis techniques have been proposed for the Web of Trust, such as authentication by

a single chain of authorities [99], and scalar maximum-flow-based techniques [73, 61] (Section 4.2.3.)

Like PGP, FaceTrust aims to circumvent the expensive and often monopolized Certificate Authorities such as VeriSign to provide lightweight credentials. Unlike PGP, FaceTrust uses the intuitive OSN interface, and employs social tagging rather than key signing to derive trustworthiness. Furthermore, FaceTrust is easily extensible, and is not limited to certifying only public keys. Users can tag each other regarding multiple types of identity assertions, and the set of assertions can be extended by simply adding fields into a user's profile.

4.2.2 Similar Problem - Different Techniques

Birthday-paradox-based Trust Inference

Like MaxTrust, SybilLimit [94] is a trust inference scheme that can differentiate between honest and Sybil nodes in a social network. It exploits the fact that although Sybil attackers can create multiple identities, they are limited in their ability to create and sustain social acquaintances. SybilLimit performs special random walks of $O(\log|V|)$ length (called random routes) starting from trusted verifier nodes and suspect nodes. In a fast-mixing social graph, the last edge traversed by the random walk is drawn from the stationary distribution of the graph. Following from the generalized Birthday Paradox, the last edges of $\Theta(sqrt|E|)$ random walks from the verifier nodes and from the honest nodes intersect with high probability.

The opposite holds if the suspect resides in a region of Sybil attackers connected with a disproportionally small number of edges to the honest node region. In this case, the network has a higher mixing time and the last edges of random walks from the suspects are not drawn from the stationary distribution. We provide a more detailed description of SybilLimit in Section 3.3.2, where we describe its use in the SocialFilter system.

FaceTrust's Sybil defense also relies on the fact that malicious users can establish only

a limited number of trust relationships with real humans. FaceTrust could employ SybilLimit instead of MaxTrust as follows. For each level of trust $0 \leq w \leq T_{max}$ we can prune the tagging similarity graph such that it includes only edges that denote greater than or equal to w similarity. We subsequently run SybilLimit for each user in the graph and for each level of trust and use as verifiers the trusted seeds. The users (suspects) that are accepted for at most a trust level w are considered to have tagger trustworthiness w. The reason we do not employ SybilLimit is that its computation cost would be $O(\sqrt{|E|}T_{max}|V|\log|V|)$, which is approximately $\sqrt{|E|}$ times more expensive than MaxTrust's under our sparse social graph setting.

Eigenvector-based Trust Inference

Similar to MaxTrust, PageRank[32], EigenTrust [53], and TrustRank [49] are trust inference methods. In this class of trust inference methods, the node trust value vector is the left principal eigenvector e of the matrix c, where c_{ij} is the normalized pairwise trust between nodes i and j. Both EigenTrust and TrustRank seed the computation of the eigenvector at a few selected trusted nodes. This computation expresses how trust flows among users through weighted edges. For a fast-mixing social graph TrustRank can be computed in $O(|V|log|V|)$ time and this value approximates the stationary distribution of the graph for users that reside in the honest fast-mixing region of the graph.

The value e_i corresponds to the probability of a random walker of the similarity graph starting from a trusted seed to land at user i. At each step with probability g, the walker follows an edge with probability proportional to the edge's tagging similarity. With probability $1-g$ the walker stops and restarts the walk from a randomly selected seed. It holds that $0 \leq e_i \leq 1$ and that $\sum_{i \in V} e_i = 1$. The eigenvector-based trust inference that is seeded at select trusted peers satisfies the "bottleneck property" (Section 2.3.3.)

For a fast-mixing social graph, TrustRank can be computed in $O(|V|\log|V|)$ time and this value approximates the stationary distribution of the graph for users that reside in

the honest and fast-mixing region of the graph. However, Sybils make the graph non-fast-mixing as they introduce regions that are connected with disproportionately small number of edges to the rest of the network. As a result after only $\log(|V|)$ iterations (or random walks of $\log(|V|)$ steps) the TrustRank of Sybil users is substantially lower than the stationary distribution probability and substantially lower than the TrustRank of honest users. Although, for sparse and small-world social graphs TrustRank's computation cost is comparable to MaxTrust's, we do not employ it because Cheng et al. have shown that eigenvector-based trust inference is substantially manipulable under Sybil strategies [35].

4.2.3 Different Problem - Similar Techniques

Max-flow-based Trust Inference

Scalar max-flow-based trust inference computes the maximum flow over a trust graph from a trusted node (source) to a suspect node (sink) in order to determine whether the suspect is trustworthy. Levien et al. [61], Reiter et al. [73] and Cheng et al. [34] have formally proved the resilience of maximum-flow-based trust metrics to node and edge attacks. In addition, a suspect node cannot increase its trust by creating Sybils, and needs to establish social edges with multiple honest nodes in order to attain the same trustworthiness as honest nodes.

FaceTrust does not employ scalar trust inference because they do not prevent an attacker from creating Sybils that obtain the same trust value as their creator. Thus, an attacker can increase the sum of the trust of the users he controls simply by adding Sybils.

Advogato [60] and Sumup [85] use group max-flow-based trust inference to build a Sybil-resilient trust metric for posters in an online forum and a voter collection system, respectively. Group max-flow trust inference bounds the sum of the trust values of Sybils by the edge capacity of their creators.

Next, we justify our design choice to use MaxTrust instead of the newly proposed Sumup [85]. MaxTrust computes multiple source max-flow from multiple trusted seeds to

the supersink user using our heuristic. Sumup computes multiple-source max flow from the users/voters to a single trusted collector with a DFS-based heuristic to decide which users can vote at least once. This heuristic costs $O(|V|\Delta)$, where Δ is the diameter of the graph. Using Sumup to decide whether a user can vote T_{max} times, is equivalent to determining the trustworthiness in $[0, T_{max}]$ of a user in MaxTrust. Unlike Sumup, MaxTrust transforms the trust metric problem to a single source-single sink maximum flow problem which can be solved with our heuristic in $O(T_{max}|E||\Delta$ (Section 2.3.3). For sparse graphs, the computation cost of Sumup and MaxTrust is comparable.

The reason we use MaxTrust instead of Sumup is its effectiveness in identifying trustworthy users. In our $200K$-user graph, MaxTrust with a single seed and $T_{max} = 1$ and $C(supersource)$ initialized at $200K$ assigned tagger trustworthiness 1 to nearly $180K$ nodes. In contrast, when the same trusted seed is chosen as the trusted vote collector node in Sumup, and Sumup's parameter, C_{max} which corresponds to MaxTrust's $C(supersource)$, is set equal to $200K$, only $60K$ users are able to send 1 vote. Unlike MaxTrust, Sumup's DFS-based heuristic for the multiple-source/single-sink max flow problem, substantially underestimates the maximum flow in social graph topologies. For this reason, the authors of Sumup propose using it for voter aggregation where typically at most 1% of the nodes in the social graph vote on any single object.

Using OSN Applications to Thwart Impersonators

Baden et al. [29] recently proposed an OSN-application-based system for identifying friends or friends of friends via exclusive shared knowledge tests. We also employ a facebook application to solicit user feedback on their friends identity assertions. FaceTrust is a system that defends against impersonating users in general, while this proposal is aimed at vetting users that claim to be friends and should thus share knowledge. The two approaches are distinct, and can complement each other (Section 2.2.3.)

4.3 Prior Work Pertinent to SocialFilter

We now discuss prior work that is pertinent to SocialFilter's design.

4.3.1 Reputation Systems

SocialFilter is inspired by prior work on reputation and trust management systems [62, 50, 31]. Well-known trust and reputation management systems include the rating scheme used by the eBay on-line auction site, object reputation systems for P2P file sharing networks [87, 53] and PageRank [32]. In contrary to the above systems, our system incorporates social trust to mitigate false reporting and Sybil attacks. EigenTrust [53], PageRank [32] and TrustRank [49], provide trust values that enable a system to rank users based on their trustworthiness. However, this value cannot be explicitly interpreted as the likelihood of a node being honest. On the other hand, SocialFilter's reporter-trust- and identity-uniqueness-based trust values provide the probability that a node is trustworthy.

4.3.2 Collaborative Email Reputation Systems

Prior work also includes proposals for collaborative spam filtering [97, 98, 6, 28]. Kong et al. [56] also consider untrustworthy reporters, using Eigentrust to derive their reputation. These solutions only enable classifying the contents of emails and not the source of spam. This requires email servers to waste resources on email reception and filtering. SocialFilter can assign trust metrics to sources, thereby rejecting unwanted email traffic on the outset.

Similar to SocialFilter, RepuScore [77] is also a collaborative reputation management framework, which allows participating organizations to establish email sender accountability on the basis of senders past actions. It provides a global reputation value for IP addresses and email domains. RepuScore utilizes a distributed hierarchical architecture as well as algorithms to collect the local reputations for email senders calculated by each email server (reporter) in each domain.

Repuscore assigns the same global reputation value for both email senders and reporters of spam. To calculate a global reputation for a sender, RepuScore weighs the local sender reputation from the reporters by the global reputation of the reporters. It does not employ a rigorous sybil-resilient and transitive trust inference method, which results in the trust values being highly susceptible to manipulation. In particular, a email server's reported local reputations are considered reliable unless the email server itself sends spam and becomes detected by other reporters.

On the other hand, SocialFilter separates the reporter reputation from the reputation of the hosts. The confidence (local reputation) in each report is weighted by the reporter trust and identity uniqueness of each nodes. It updates the direct trust between email servers based on the similarity of the reported spam events and a manually set trust value that administrators input to the system. According to these direct trust values the reporter trust score of each reporter is calculated over the social graph using a transitive trust inference method. Unlike Repuscore, SocialFilter's Sybil detection mechanism can defeat Sybils before those Sybils send a substantial amount of spam. Thus, compared to Repuscore, SocialFilter provides increased attack-resilience in the face of malicious reporters and Sybils.

4.3.3 IP Blacklisting

SocialFilter is similar to IP blacklisting services such as SpamHaus [17], DShield [4] and TrustedSource [15] in that it employs a centralized repository. Currently, IP blacklisting relies on a relatively small number (in the order of a few hundreds or thousands) of reporters. Reporters submit their attack logs to the centralized repositories, and the repositories synthesize blacklists based on the attack history recorded in these logs. SocialFilter differs in that it automates the process of evaluating the reports and assigning reputations to reporters. Thus it does not incur the management overhead of traditional IP blacklisting services. It can therefore scale to millions of reporters. CloudMark [3] explicitly addresses the issue of trustworthiness of the collaborating spam reporters through a distributed re-

porter reputation management system based on history of past interactions. However, it does not leverage the social network to derive trust information.

Predictive blacklisting [95, 80] improves upon IP blacklisting by creating a customized blacklist for each reporter that is shorter and more likely to be relevant. However, it does not address adversarial behavior by reporters, i.e., false reporting combined with Sybil attacks. In addition, because it does not employ user-defined social trust, a node is able to obtain a customized ranking only if the node itself has classification functionality.

Highly Predictive Blacklisting [95] employs similarity between reports to rank how similarly the malicious traffic that two reporters encounter is. It uses a variation of the PageRank trust metric. This trust metric is computed from the point of view of each reporter. That is, the PageRank computation is seeded at the reporter, so that it can determine which nodes have encountered attacks that are more likely to be experienced by the reporter. The end-goal is to limit the size of blacklists shared between nodes by sending only the entries that are more relevant to each node. On the other hand, SocialFilter uses a global trust inference method, *i.e.*, computes trust values from the point of view of a few trusted nodes and these trust values are common in the system. This is because SocialFilter aims at determining the trustworthiness of nodes with respect to reporting ground truth events; typically a host is either a spammer or it is not regardless of who is the reporter that observes it.

4.3.4 Other Sybil defenses

Common defenses against Sybil attacks are based on resource testing of computing or storage capability. The underlying assumption is that a Sybil attacker does not possess enough resources to perform the additional tests imposed on each Sybil node. Some drawbacks with resource testing are listed in [39], such as the fact that attackers subvert this defense by tricking humans into solving CAPTCHAS [24] posted on their website or presented by malware on infected machines.

Similar to MaxTrust and SybilLimit, the schemes in [37, 64] take advantage of the fact that clusters of Sybils are connected to the honest regions of social networks with a disproportionally small number of edges. Danezis et al.'s [37] Bayesian Sybil detection method (SybilInfer) derives the probability of a suspect node being a Sybil, which is an explicitly actionable measure of trustworthiness. However, its computation cost ($O(|V|^2 \log |V|)$ is excessive for FaceTrust's setting, where we expect social graphs in the order of tens of millions of users. We considered SybilInfer as an alternative of SybilLimit in SocialFilter. Although SybilInfer provides a more formal probabilistic interpretation of identity uniqueness, it is more expensive than SybilLimit, which costs ($O(\sqrt{|E|}|V| \log |V|)$ in the sparse social graph setting of SocialFilter. Nevertheless, we believe SybilInfer is a plausible alternative to SybilLimit and we are currently experimenting with it.

Ostra [64] (§ 3.4.1) bears similarity to Sumup [85] in that it limits unwanted communication from Sybils. FaceTrust could employ a variation of Ostra by replacing its pairwise credit balance edge with the pairwise tagging-similarity-annotated edges. However, Ostra does not specify an efficient way to determine trust paths, as is the case with MaxTrust.

Whanau [59] forwards DHT queries over carefully constructed routes that consist of social links. It employs social trust in the sense that it considers trustworthy only forwarding links that correspond to social links. It borrows ideas from SybilLimit to construct its so-termed "layered identifiers', which improve its Sybil-resilience.

FaceTrust and SocialFilter can employ non-social-network-based Sybil defense techniques such as the ones proposed in [96, 94] to further limit the influence of Sybils in the system. BotGraph [96] detects botnet spamming attacks that target Web email providers. BotGraph detects botnets by constructing large user-user graphs for links between email-exchanging users and looking for tightly connected subgraph components. As stated in [96] this technique is applicable in social graphs.

5

Future Work

We now discuss ongoing and future work to improve FaceTrust and SocialFilter.

5.1 Incorporating External Sources of Trust

FaceTrust currently relies solely on explicit user feedback through tags to derive the veracity of an assertion. We are considering other external sources of trust information. These external sources can be used to both directly derive the veracity of the assertion and to derive the tagger trustworthiness of a user. By using additional sources of trust we can improve the accuracy of our trust values.

In particular, it is possible to assess the veracity of a professional identity statement, simply by analyzing the social network of the assertion poster. For example, it is more likely that a user is a CS professor if many of his acquaintances claim to be professors themselves or have PhD in CS. We intend to explore techniques that infer identity attributes of users by analyzing their social network [45, 14, 65].

FaceTrust can also be combined with stronger authentication mechanisms (such as manual verification) to acquire a bottom line on the identity of users. For example, users that are frequent review writers can present to FaceTrust proof of their profession in order

to increase their veracity score. A parent that wishes to prevent her teenage children from accessing age restricted sites can use out-of-band means to prove to FaceTrust her relation with her children. FaceTrust can then assign a lot more weight to her tags on her children's age assertions.

5.2 Attack-Resistance Analysis

We intend to extend FaceTrust's attack-resistance analysis with a more realistic model. We currently consider a tree social graph topology and we intend to extend it to more realistic topologies.

We will also define a formal utility model. That is, we will quantify the cost that an attacker has to incur to effectively increase the veracity of his and his colluders assertions. An attacker incurs cost to creates a new OSN account, to create friend connections and to vote similarly with honest users. The gains are the tagger trustworthiness he accumulates and the increase of veracity of his assertions.

5.3 Probabilistic Interpretation

Currently, our assertion veracity and spammer belief measures cannot be interpreted as the likelihood or probability of an assertion or host being trustworthy. We are investigating the use of a Bayesian models of trust [40, 55], such that we can derive veracity values that directly correspond to the probability of an assertion or host being trustworthy. Our goal is to design an efficient (less than $O(V^2)$) Bayesian trust algorithm.

5.4 Inferring Trust Between Friends in OSNs

In the physical world, declaring a person as a social acquaintance implies a high or moderate level of trust. To build trust with a person, one needs to devote effort, such as go to meetings, have conversations, and so on. Therefore, as stated in [93, 92], social connec-

tions can be used as a resource test to defend against Sybil [39] attacks.

However, in the realm of web-based Online Social Networks (OSN), users tend to be less careful with respect to whom they declare as friend. Our preliminary measurements have shown that it is relatively easy for fake Facebook users to establish friend connections with many real Facebook users. This experimental observation challenges the assumption that connections in the online social graph can be used as a resource test.

Our ongoing research aims to quantify with large scale measurements how easy it is for attackers to connect to users depending on the users' profile characteristics (age, profession, location, etc.) We are also working on efficient methods to detect "promiscuous" users who connect to others indiscriminately. Such detected users could then be ignored by social-graph-based Sybil detection algorithms.

We are also working in the direction of inferring real levels of trust between OSN friends via communication patterns and mutual privacy settings. The real trust levels can then be used to secure distributed systems as proposed in this thesis and [92, 64], or to appropriately determine privacy settings.

6

Conclusion

Distributed information systems leverage the Internet infrastructure to to solve challenging computational problems and to improve many aspects of every day human activities. For a distributed system to be effective in its tasks, it often has to span a multitude of devices across multiple administrative domains. This openness and scale results in a serious tension by introducing issues of trust: components of a distributed system need to be able to assess each other's trustworthiness.

The recent rise in popularity of Online Social Networking services as a popular communication medium has highlighted the importance of considering human relations in the design of our distributed infrastructure. Since OSNs encode relationships between humans, it is natural to consider them as a source of readily available trust information.

Hence, this work tackles the issue of improving distributed systems by means of social trust. We have focused on leveraging social networks to address two important security problems: a) how to assess the veracity of statements that OSN users make; and b) how to assess the trustworthiness of spam reporters in a collaborative spam detection system.

This thesis has demonstrated the plausibility of using Online Social Networks as a Sybil-resilient trust substrate to address these problems. Our design and implementa-

tion has demonstrated the practicality of using centralized OSN services to analyze trust-annotated social graphs to derive trust values for their users. We have learned that it is plausible to rely on those trust values to weigh the classification input (tags or spammer reports) of OSN users. Furthermore, our Facebook application deployment and user study has showed that it is plausible to use the OSN user interface to enable users to express their trust towards each other's classification ability and identity assertions.

6.1 Assessing the Veracity of Online Identity Assertions

We presented FaceTrust, a system that leverages online social networks to provide lightweight, extensible, relaxed and optionally anonymous credentials. These credentials assist users and services in assessing the veracity of assertions made by online users. With FaceTrust, OSN users post identity assertions such as "Am I really 18 years old?" on their social network profiles and their friends explicitly tag these assertions as `true` or `false`. An OSN provider analyzes the social graph and the user tags to assess how credible these assertions are, and issues credentials annotated by veracity scores. Our analysis, simulation-based evaluation, and real-world deployment suggest that FaceTrust is effective in obtaining credible and otherwise unavailable identity information for online personas.

We have derived the following lessons from the design and deployment of FaceTrust: a) it is plausible to obtain a relatively reliable measure of the veracity of identity assertions by relying on the friends of the user that made the assertion to classify them, and by employing social trust to determine the trustworthiness of the classifications; b) it is plausible to employ trust inference over the social graph to effectively mitigate Sybil attacks; c) users tend to mostly correctly classify their friends' identity assertions.

6.2 Introducing Social Trust to Collaborative Spam Mitigation

We have also presented SocialFilter, a large scale distributed system for collaborative spam mitigation. SocialFilter nodes consider each other's reports on encountered spam hosts and the social network of their human administrators to quantify the belief that a host is spamming. Applications can in turn use this belief measure to make informed decisions on how to handle traffic associated with that host.

Our simulation-based evaluation demonstrated our design's potential for the suppression of spam email. SocialFilter was able to identify 99% of spam connections with greater than 50% belief. Furthermore, in contrast to a competing social-network-based spam mitigation technique, Ostra [64], SocialFilter exhibited no false positives.

The design and evaluation of SocialFilter offered us the following lessons: a) we can improve the reliability and the attack-resilience of collaborative spam mitigation by introducing Sybil-resilient OSN-based trust inference mechanisms; b) using social links to obtain the trustworthiness of spammer reports can result in comparable spam-blocking effectiveness with approaches that use social links to rate-limit spam (e.g., Ostra [64]); c) unlike Ostra, SocialFilter yields no false positives. We believe that the design lessons from SocialFilter are applicable to other collaborative entity classification systems.

Bibliography

[1] "Am I Really?" Facebook Application. http://www.facebook.com/apps/application.php?id=82228014118.

[2] Belkin's Amazon Rep Paying For Fake Online Reviews. http://tinyurl.com/yzgp9co.

[3] Cloudmark. www.cloudmark.com/en/home.html.

[4] Cooperative Network Security Community. http://www.dshield.org/.

[5] Craigslist scams. www.craigslist.org/about/scams.

[6] Distributed Checksum Clearinghouses Reputations. www.rhyolite.com/dcc/reputations.html.

[7] Facebook connect. developers.facebook.com/connect.php.

[8] Facebook developers api. developers.facebook.com/.

[9] Facebook more popular than google? for one week, it hit first place. http://www.csmonitor.com/Innovation/Horizons/2010/0315/Facebook-more-popular-than-Google-For-one-week-it-hit-first-place.

[10] "Friend Facts" Facebook Application. http://www.facebook.com/apps/application.php?id=51254684277.

[11] Google Safe Browsing API. code.google.com/apis/safebrowsing.

[12] McAfee SiteAdvisor. www.siteadvisor.com.

[13] Nigerian Advance Fee Fraud. www.state.gov/www/regions/africa/naffpub.pdf.

[14] Project Gaydar. At MIT, an Experiment Identifies which Students are Gay, Raising New Questions about Online Privacy. http://www.boston.com/bostonglobe/ideas/articles/2009/09/20/project_gaydar_an_mit_experiment_raises_new_questions_about_online_privacy/?page=full.

[15] Secure Computing, TrustedSource. www.securecomputing.com/index.cfm?skey=1671.

[16] Thawte web of trust. www.thawte.com/secure-email/web-of-trust-wot/.

[17] The SpamHaus Project. www.spamhaus.org/.

[18] Unedited. Unfiltered. News. iReport.com. www.ireport.com.

[19] Verisign: Personal Digital Certificate Enrollment. https://personalid.verisign.com.au.

[20] Worth DOUBLE the money. http://tinyurl.com/y8pqgvl.

[21] Your Real Name ^{TM}Attribution. http://www.amazon.com/gp/help/customer/display.html?nodeId=14279641.

[22] Amazon glitch outs authors reviewing own books. www.ctv.ca/servlet/ArticleNews/story/CTVNews/1076990577460_35, 2004.

[23] Teen Accused of Sex assaults in Facebook Scam. www.msnbc.msn.com/id/29032437/, 2009.

[24] Luis Von Ahn, Manuel Blum, Nicholas J. Hopper, and John Langford. Captcha: Using Hard AI Problems for Security. In *Eurocrypt*, 2003.

[25] Yong-Yeol Ahn, Seungyeop Han, Haewoon Kwak, Sue Moon, and Hawoong Jeong. Analysis of Topological Characteristics of Huge Online Social Networking Services. In *WWW*, 2007.

[26] Reka Albert, Hawoong Jeong, and Albert-Làszló Barabási. Internet: Diameter of the World-Wide Web. In *Nature*, 1999.

[27] Reid Andersen, Christian Borgs, Jennifer Chayes, U. Uriel Feige, Abraham Flaxman, Adam Kalai, Vahab Mirrokni, and Moshe Tennenholtz. Trust-based Recommendation Systems: An Axiomatic Approach. In *WWW*, 2008.

[28] Josh Attenberg, Kilian Weinberger, Anirban Dasgupta, Alex Smola, and Martin Zinkevich. Collaborative Email-Spam Filtering with the Hashing Trick. In *CEAS*, 2009.

[29] Randy Baden, Neil Spring, and Bobby Bhattacharjee. Identifying Close Friends on the Internet. In *HotNets*, 2009.

[30] Albert-Làszló Barabási and Reka Albert. Emergence of Scaling in Random Networks. In *Science*, 1999.

[31] Matt Blaze, Joan Feigenbaum, and J Jack Lacy. Decentralized Trust Management. In *IEEE S&P*, 1996.

[32] Sergey Brin and Larry Page. The Anatomy of a Large-scale Hypertextual Web Search Engine. In *Computer Networks and ISDN Systems*, 1998.

[33] Jan Camenisch and Els Van Herreweghen. Design and Implementation of the idemix Anonymous Credential System. In *ACM CCS*, 2002.

[34] Alice Cheng and Eric Friedman. Sybilproof Reputation Mechanisms. In *P2PEcon*, 2005.

[35] Alice Cheng and Eric Friedman. Manipulability of PageRank under Sybil Strategies. In *NetEcon*, 2006.

[36] Bram Cohen. Incentives Build Robustness in BitTorrent. In *P2P Econ*, June 2003.

[37] George Danezis and Prateek Mittal. SybilInfer: Detecting Sybil Nodes using Social Networks. In *NDSS*, 2009.

[38] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *OSDI*, 2004.

[39] John R. Douceur. The Sybil Attack. In *IPTPS*, March 2002.

[40] Thomas DuBois, Jennifer Golbeck, , and Aravind Srinivasan. Rigorous Probabilistic trust-inference with Applications to Clustering. In *International Conference on Web Intelligence*, 2009.

[41] Marlena Erdos and Scott Cantor. Shibboleth Architecture DRAFT v05. *Internet2/MACE, May*, 2, 2002.

[42] Paul Erdös and Alfréd Rényi. On Random Graphs. In *I. Publicationes Mathematicae Debrecen*, volume 5:290–297, 1959.

[43] Scott Garriss, Michael Kaminsky, Michael Freedman, Brad Karp, David Mazieres, and Haifeng Yu. Re:Reliable Email. In *NSDI*, 2006.

[44] Minas Gjoka, Maciej Kurant, Carter T. Butts, and Athina Markopoulou. A Walk in Facebook: Uniform Sampling of Users in Online Social Networks. In *INFOCOM*, 2010.

[45] Jennifer Golbeck. Trust and nuanced profile similarity in online social networks. volume 3, pages 1–33, 2009.

[46] Elizabeth Gray, Jean-Mark Seigneur, Yong Chen, and Christian Jensen. Trust Propagation in Small Worlds. In *LNCS*, pages 239–254. Springer, 2003.

[47] Ratan Kuman Guha, Ravi Kumar, Prabhakar Raghavan, and Andrew Tomkins. Propagation of Trust and Distrust. In *WWW*, 2004.

[48] Krishna Gummadi, Ramakrishna Gummadi, Steven Gribble, Sylvia Ratnasamy, Scott Shenker, and Ion Stoica. The Impact of DHT Routing Geometry on Resilience and Proximity. In *SIGCOMM*, 2003.

[49] Zóltan Gyöngyi, Hector Garcia-Molina, and Jan Pedersen. Combating Web Spam with TrustRank. In *VLDB*, 2004.

[50] Kevin Hoffman, David Zage, and Cristina Nita-Rotaru. A Survey of Attack and Defense Techniques for Reputation Systems. In *ACM Computing Surveys*, 2008.

[51] Tad Hogg and Lada Adamic. Enhancing Reputation Mechanisms via Online Social networks. In *ACM Conference on Electronic Commerce*, 2004.

[52] Paul Jaccard. Etude Comparative de la Distribution Florale dans une Portion des Alpes et des Jura. In *Bulletin del la Socit Vaudoise des Sciences Naturelles 37, 547-579*, 1901.

[53] Sepandar D. Kamvar, Mario T. Schlosser, and Hector Garcia-Molina. The EigenTrust Algorithm for Reputation Management in P2P Networks. In *WWW*, 2003.

[54] Sachin Katti, Balachander Krishnamurthy, and Dina Katabi. Collaborating Against Common Enemies. In *ACM IMC*, 2005.

[55] Reto Kohlas and Ueli Maurer. Confidence valuation in a public-key infrastructure based on uncertain evidence. In *PKC*, 2000.

[56] Joseph S. Kong, B A. Rezaei, N Sarshar, V P. Roychowdhury, and P O. Boykin. Collaborative Spam Filtering Using e-mail Networks. In *IEEE Computer*, 2006.

[57] Lesslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine Generals Problem. In *ACM TOPLAS*, 1982.

[58] Jure Leskovec and Christos Faloutsos. Sampling from Large Graphs. In *SIGKDD*, 2006.

[59] Chris Lesniewski-Laas and M. Frans Kaashoek. Whanau: A Sybil-proof Distributed Hash Table. In *NSDI*, 2010.

[60] Raph Levien. Attack-resistant Trust Metrics. `www.levien.com/thesis/compact.pdf`, 2003.

[61] Raph Levien and Alexander Aiken. Attack-resistant Trust Metrics for Public Key Certification. In *Usenix Security*, 1997.

[62] Sergio Marti and Hector Garcia-Molina. Taxonomy of Trust: Categorizing P2P Reputation Systems. In *Computer Networks*, 2006.

[63] Alan Mislove, Massimiliano Marcon, Krishna P. Gummadi, Peter Druschel, and Samrat Bhattacharjee. Measurement and Analysis of Online Social Networks. In *IMC*, 2007.

[64] Alan Mislove, Ansley Post, Peter Druschel, and Krishna P. Gummadi. Ostra: Leveraging Social Networks to Thwart Unwanted Traffic. In *NSDI*, 2008.

[65] Alan Mislove, Bimal Viswanath, Krishna P. Gummadi, and Peter Druschel. You are who you Know: Inferring User Profiles in Online Social Networks. In *ACM WSDM*, 2010.

[66] Klaus Mller and Tony Vignaux. SimPy (Simulation in Python). `simpy.sourceforge.net/`.

[67] Tavi Nathanson, Ephrat Bitton, and Ken Goldberg. Eigentaste 5.0: Constant-time adaptability in a recommender system using item clustering.

[68] Vern Paxson. Bro: A System for Detecting Network Intruders in Real-Time. In *Computer Networks*, 1999.

[69] Josep M. Pujol and Ramon Sangesa Jordi Delgado. Extracting Reputation in Multi Agent Systems by Means of Social Network Topology. In *International Conference on Autonomous agents and Multiagent Systems*, 2002.

[70] Anirudh Ramachandran and Nick Feamster. Understanding the Network-level Behavior of Spammers. In *ACM SIGCOMM*, 2006.

[71] Anirudh Ramachandran and Nick Feamster. Authenticated Out-of-Band Communication Over Social Links. In *WOSN*, 2008.

[72] Anirudh Ramachandran, Nick Feamster, and Santosh Vempala. Filtering Spam with Behavioral Blacklisting. In *ACM CCS*, 2007.

[73] M. Reiter and S. Stubblebine. Authentication Metric Analysis and Design. In *ACM TISSEC*, 1999.

[74] Michael K. Reiter and Stuart G. Stubblebine. Toward Acceptable Metrics of Authentication. In *IEEE S&P*, 1997.

[75] Jordi Sabater and Carles Sierra. Reputation and Social Network Analysis in Multiagent Systems. In *International Conference on Autonomous agents and Multiagent Systems*, 2002.

[76] Gautam Singaraju and Brent ByungHoon Kang. RepuScore: Collaborative Reputation Management Framework for Email Infrastructure. In *USENIX LISA*, 2007.

[77] Gautam Singaraju, Jeff Moss, and Brent ByungHoon Kang. Tracking Email Reputation for Authenticated Sender Identities. In *CEAS*, 2008.

[78] Michael Sirivianos, Kyungbaek Kim, and Xiaowei Yang. FaceTrust: Assessing the Credibility of Online Personas via Social Networks. In *HotSec*, 2009.

[79] Michael Sirivianos, Xiaowei Yang, and Stanislaw Jarecki. Robust and Efficient Incentives for Cooperative Content Distribution. In *IEEE/ACM Transactions on Networking, to appear*, 2009.

[80] Fabio Soldo, Anh Le, and Athina Markopoulou. Predictive Blacklisting as an Implicit Recommendation System. In *INFOCOM*, 2010.

[81] Yair Sovran, Alana Libonati, and Jinyang L. Pass it on: Social Networks Stymie Censors. In *IPTPS*, 2008.

[82] William Stallings. Protect Your Privacy: A Guide for PGP Users. In *Prentice-Hall*, 1995.

[83] Jennifer G. Steiner, Clifford Neuman, and Jeffrey I. Schiller. Kerberos: An Authentication Service for Open Network Systems. In *USENIX Winter Conference*, 1988.

[84] Amin Tootoonchian, Stefan Saroiu, Yashar Ganjali, and Alec Wolman. Lockr: Better Privacy for Social Networks. In *CoNEXT*, 2009.

[85] Dinh Nguyen Tran, Bonan Min, Jinyang Li, and Lakshminarayanan Subramanian. Sybil-Resilient Online Content Rating. In *NSDI*, 2009.

[86] Twitter. Verified Account. `http://twitter.com/help/verified`.

[87] Kevin Walsh and Emin Gun Sirer. Experience with an Object Reputation System for Peer-to-Peer Filesharing. In *NSDI*, 2006.

[88] Duncan J. Watts and Steven H. Strogatz. Collective Dynamics of Small-World Networks. In *Nature*, 1998.

[89] Alma Whitten and Doug Tygar. Why Johnny can't Encrypt: A Usability Evaluation of PGP 5.0. In *USENIX Security*, 1999.

[90] Yinglian Xie, Fang Yu, Kannan Achan, Rina Panigrahy, Geoff Hulten, and Ivan Osipko. Spamming Botnets: Signatures and Characteristics. In *ACM SIGCOMM*, 2008.

[91] Sarita Yardi, Nick Feamster, and Amy Bruckman. Photo-Based Authentication Using Social Networks. In *WOSN*, 2008.

[92] Haifeng Yu, Phillip Gibbons, Michael Kaminsky, and Feng Xiao. A Near-Optimal Social Network Defense Against Sybil Attacks. In *IEEE S&P*, 2008.

[93] Haifeng Yu, Michael Kaminsky, Phillip B. Gibbons, and Abraham Flaxman. SybilGuard: Defending Against Sybil Attacks via Social Networks. In *ACM SIGCOMM*, 2006.

[94] Haifeng Yu, Chenwei Shi, Michael Kaminsky, Phillip B. Gibbons, and Feng Xiao. DSybil: Optimal Sybil-Resistance for Recommendation Systems. In *IEEE S&P*, 2009.

[95] Jian Zhang, Philip Porras, and Johannes Ullrich. Highly Predictive Blacklisting. In *USENIX Security*, 2008.

[96] Yao Zhao, Yinglian Xie, Fang Yu, Qifa Ke, Yuan Yu, Yan Chen, and Elliot Gillum. Botgraph: Large Scale Spamming Botnet Detection. In *NSDI*, 2009.

[97] Zhenyu Zhong, Lakshmish Ramaswamy, and Kang Li. ALPACAS: A Large-scale Privacy-Aware Collaborative Anti-spam System. In *IEEE INFOCOM*, 2008.

[98] Feng Zhou, Li Zhuang, Ben Y. Zhao, Ling Huang, Anthony D. Joseph, and John Kubiatowicz. Approximate Object Location and Spam Filtering on Peer-to-Peer Systems. In *ACM/IFIP/USENIX Middleware*, 2003.

[99] Philip R. Zimmerman. The Official PGP Users Guide. In *MIT Press*, 1995.

Biography

Michael Sirivianos was born in Athens, Greece on March 4th 1978. He defended his Phd Thesis at Duke University in April 2010. His research interests include introducing social trust in distributed system design, cooperative content distribution and human verifiable secure device pairing. He received a B.S in Electrical and Computer Engineering from the National Technical University of Athens in 2002, and an M.S. in Computer Science from the University of California, San Diego in 2004.

Last but not least, he has served as systems security officer on board the Federation Starfleet's USS Enterprise, under the command of Captain Jean-Luc Piccard.