Combating Sybil attacks in cooperative systems

by

Nguyen Tran

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

Courant Institute of Mathematical Sciences

New York University

Sep 2012

Professor Jinyang Li

To my parents and my lovely wife.

Acknowledgements

This thesis is the result of the collaboration with Jinyang Li, Lakshminarayanan Subramanian, Bonan Min and Sherman S.M. Chow. Bonan evaluated SumUp on Digg's voting trace. Sherman helped in improving the security analysis of Gatekeeper. I have discussed numerous times with Jinyang and Lakshmi to turn half-baked ideas into techniques, protocols and systems presented in this thesis. The research in this thesis is partly supported by the NSF CAREER award CNS-0747052. I am very grateful to Google for supporting the work with a Google PhD Fellowship in Distributed Systems.

I am very lucky to have Jinyang Li and Lakshminarayanan Subramanian as my advisors. Being their first student, I received a lot of their attention during my first two years, which was a critical period for an unexperienced PhD student like me. They have provided me enormous help and encouragement not only in research but also in life. Their intellect, rigor, enthusiasm, and diligence will always inspire me. I am also grateful to Marcos K. Aguilera and Frank Dabek who were my mentors at Microsoft Research and Google. They have provided me with skills that I could not get from graduate school.

It was fortunate for me to have Yair Sovran, Eric Hielscher and Russell Power around. They not only provided great comments on my thesis work but also are excellent collaborators in other research over the years.

Beyond those mentioned earlier, I would like to thank the following folks at NYU NEWS group, UC Berkeley and other institutes who made my six years at graduate school a fun journey: Jay Chen, Robert Soule, Arthur Meacham, Aditya Dhananjay, Aslesh Sharma, Michael Paik, Matt Tierney, Sunandan Chakraborty, Christopher Mitchell, Anh Le, Y.C. Tay, Mahesh Balakrisnan, Charalampos Papamanthou, Mohit Tiwari, Prashanth Mohan, Emil Stefanov, Dawn Song, Elaine Shi, and Prateek Mittal.

Abstract

Cooperative systems are ubiquitous nowadays. In a cooperative system, end users contribute resource to run the service instead of only receiving the service passively from the system. For example, users upload and comment pictures and videos on Flicker and YouTube, users submit and vote on news articles on Digg. As another example, users in BitTorrent contribute bandwidth and storage to help each other download content. As long as users behave as expected, these systems benefit immensely from user contribution. In fact, five out of the ten most popular websites are operating in this cooperative fashion (Facebook, YouTube, Blogger, Twitter, Wikipedia). BitTorrent is dominating the global Internet traffic.

A robust cooperative system cannot blindly trust that its users will truthfully participate in the system. Malicious users seek to exploit the systems for profit. Selfish users consume but avoid contributing resource. For example, adversaries have manipulated the voting system of Digg to promote their articles of dubious quality. Selfish users in public BitTorrent communities leave the system to avoid uploading files to others, resulting in drastic performance degradation for these content distribution systems. The ultimate way to disrupt security and incentive mechanisms of cooperative systems is using Sybil attacks, in which the adversary creates many Sybil identities (fake identities) and uses them to disrupt the systems' normal operation. No security and incentive mechanism works correctly if the systems do not have a robust identity management that can defend against Sybil attacks.

This thesis provides robust identity management schemes which are resilient to the Sybil attack, and uses them to secure and incentivize user contribution in several example cooperative systems. The main theme of this work is to leverage the social network among users in designing secure and incentive-compatible cooperative systems. First, we develop a distributed admission control protocol, called Gatekeeper, that leverages the social network to admit most honest user identities and only few Sybil identities into the systems. Gatekeeper can be used as a robust identity management for both centralized and decentralized cooperative systems. Second, we provide a vote aggregation system for content voting systems, called SumUp, that can prevent an adversary from casting many bogus votes for a piece of content using the Sybil attack. SumUp leverages unique properties of content voting systems to provide significantly better Sybil defense compared with applying a general admission control protocol such as Gatekeeper. Finally, we provide a robust reputation system, called Credo, that can be used to incentivize bandwidth contribution in peer-to-peer content distribution networks. Credo reputation can capture user contribution, and is resilient to both Sybil and collusion attacks.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Our society is entering an era in which many Internet services are designed to leverage the cooperative nature of their users. In a *cooperative system*, end users contribute resource in the form of contents, opinions, or physical resources instead of simply receiving the service from the system passively. For example, users share and comment on pictures (Flickr), or videos (YouTube). Users submit and vote on news articles on Digg, and answer each others' questions on Quora. In peer-to-peer networks such as BitTorrent, users are contributing bandwidth and storage to help each other download content. As long as users behave as expected, cooperative systems can receive immense benefit from users' contribution and scale inexpensively. Cooperative systems are ubiquitous today: five out of the ten most popular websites are operating in this cooperative fashion (Facebook, YouTube, Blogger, Twitter, Wikipedia). BitTorrent is dominating the global Internet traffic [69].

A robust cooperative system cannot blindly trust that its users will truthfully participate in the system. Malicious users seek to exploit the system for profit. Selfish users consume resource but avoid contribution. For example, adversaries have manipulated the voting system of Digg to promote their articles of dubious quality [73]. Selfish users in public BitTorrent communities leave the system as soon as they have finished downloading a file to avoid uploading the file to others, resulting in serious performance degradation for these content distribution systems [71]. A robust cooperative system must be able to limit the amount of damage inflicted by adversaries and to incentivize honest users to make adequate contribution.

The biggest threat that cooperative systems face is the Sybil attack [19], in which the adversary creates many Sybil identities (fake identities) and use them to disrupt the systems' normal operation. No security and incentive mechanism can work correctly if it lacks a *robust identity management* that can defend against Sybil attacks. For example, many systems replicate computation and storage among different nodes in order to protect data integrity (data loss) [16, 70, 8]. Others [25, 12, 26] divide computation and storage tasks and assign them to different nodes in order to protect data privacy. In both cases, these systems achieve the security and integrity guarantees only when independent tasks are performed by distinct users. Using the Sybil attack, the adversary can violate this security requirement by potentially controlling a majority of the identities in [78, 36, 19]. Likewise, Sybil attacks also thwart incentive mechanisms. A common approach to incentivize user contribution for the system is to provide higher quality of service (or other benefits) to users who have contributed more [54, 57, 31]. To keep track of user contribution, one can use a reputation system. However, the adversaries or selfish users can use Sybil identities to boost their reputation easily [54], thereby disrupting this incentive mechanism. The Sybil attack is not just a hypothetical threat investigated by the research community, but is actually happening in the real world: adversaries are using fake identities to launch spamming campaign on popular social networks like Facebook [67] and Renren [81], and promote articles with low quality on Digg [73]. Therefore, in order to provide security and incentive guarantees for cooperative systems, we must manage user identities in a Sybil-resilient fashion.

Defending against the Sybil attack is challenging. Existing systems restrict the creation of Sybil identities by limiting one identity per IP address or per solved CAPTCHA puzzle. Unfortunately, IP addresses and CAPTCHA solutions can be cheap to obtain. An ordinary human can solve a few thousands of CAPTCHA puzzles in one day in order to create thousands of Sybil identities. The adversary can also pay for an online service to solve CAPTCHA puzzles at a rate of $\$8$ for 1000 puzzles [4]. The adversary can obtain different IP addresses using public proxies or harvest IP addresses within his institution. The latter trick was used by MIT and CMU students to game an online poll for ”the best graduate school in computer science” [7].

This thesis provides robust identity management schemes which are resilient to the Sybil attack, and uses them to secure and incentivize user contribution in several example cooperative systems. The main theme of this work is to exploit the social relationship among users as a form of user identity. The insight is to associate the identity of each user with her social links with other users. Since social relationships take significant human effort to establish, the adversary is limited to a few links with honest users and thus can only have a few usable identities.

In this thesis, we first develop a distributed admission control protocol, called Gatekeeper, that leverages the social network to admit most honest user identities and only a few Sybil identities. Gatekeeper can be used as a foundational mechanism to admit users and nodes for both centralized and decentralized cooperative systems. Next, this thesis addresses the security and incentive challenges in two specific applications: an online content voting system and a peer-to-peer content distribution network (P2P CDN). A content voting system must prevent the adversary from casting a large number of bogus votes to boost the ranking of low quality content. We design a flow-based vote aggregation system called SumUp to collect most honest votes while limiting the number of bogus votes. The defense of both SumUp and Gatekeeper are currently the state-of-the-art in their respective problems as reported in a recent survey [82]. A P2P CDN faces serious performance problems as selfish nodes avoid uploading files after finishing downloading them. We provide a robust reputation system, called Credo, that is resilient to Sybil and collusion attacks, and can be used to incentivize bandwidth contribution in peer-to-peer content distribution networks. Credo relies on Gatekeeper to limit the number of Sybil identities an adversary can bring into the system, and a ”modeling good behavior” technique to defend against the collusion attack in which many adversaries collude to boost each other’s reputation.

In the rest of this chapter, we first explain the insight of social network based Sybil defense in Section 1.1. Next, we discuss the main findings of Gatekeeper, SumUp, and Credo before summarizing the contributions of this thesis in Section 1.5.

1.1 Social network based Sybil defense

In the real world, there exists a social network among people. An edge in this network represents a friendship between two people. Because creating and maintaining friendships require efforts in the real world, the adversary can only have a limited number of edges, which are called *attack edges*, to honest users. As a result, we can view these edges as a valuable resource for defending against the Sybil attack.

Although the adversary only possesses a limited number of attack edges, he can create a large number of Sybil identities and link them together arbitrarily to form any topology. Nevertheless, these Sybil identities are separated from the honest region of the

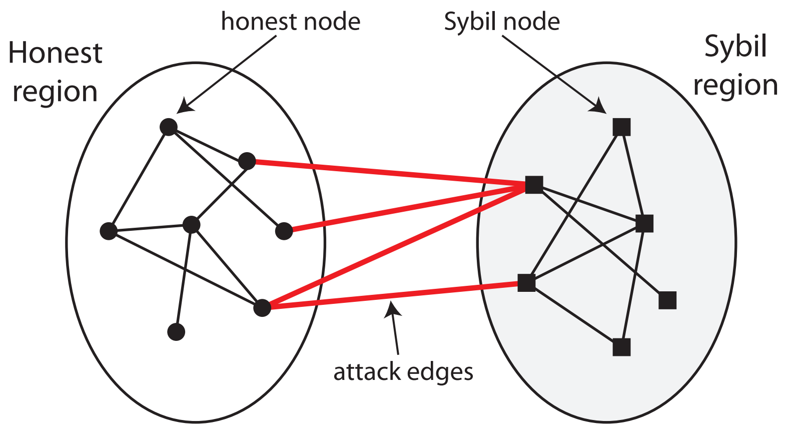


Figure 1.1: Sybil region and honest region are separated by a small cut formed by the attack edges as illustrated in SybilGuard [84].

social network by the cut which is formed by the attack edges, as showed in Figure 1.1. A large Sybil region creates an abnormal feature in the overall graph because we expect the social graph to be well-connected. For example, it is known that social graphs exhibit the fast mixing property [49, 84], in which a random walk starting from any node reaches its steady state quickly. To be more precise, the mixing time is bounded by $O(\log n)$, where n is the number of honest nodes in the graph. By contrast, it takes a long time for a random walk starting from the honest region in Figure 1.1 to arrive at a random node in the Sybil-region because the walk needs to pass through the small cut. A social network based Sybil defense mechanism exploits this abnormality to detect and eliminate Sybil identities from the system. As a result, the number of Sybil identities that the adversary can bring is limited by his ability to create attack edges.

SybilGuard [84] is the first work that exploits this insight to design a distributed *admission control protocol* in 2006. An admission control protocol is expected to admit most honest identities while limiting Sybil identities admitted into a system. SybilGuard guarantees that it only admits $O(\sqrt{n}\log n)$ Sybil identities per attack edge with high probability. While this bound presents an important first step in limiting the number of Sybil identities in the system, it is still big for practical systems because the number of honest users (n) can be on the order of millions for popular systems. In 2008, the authors of SybilGuard presented a new protocol, called SybilLimit [83], which improves the bound to admitting only $O(\log n)$ Sybil identities per attack edge. It has since remained an opened question if this bound can be further improved for an admission control protocol. We addressed this open challenge with the Gatekeeper protocol. We showed that Gatekeeper admits only $O(\log k)$ Sybil identities per attack edge, where k is the number of attack edges.

Apart from developing Sybil-resilient admission controls, some work also leverage the social network to secure specific applications. For example, Whānau [36] and X-Vine [48] provide integrity and privacy for distributed hash tables. SumUp [73] protects online content voting systems. Ostra [47] and SocialFilter [64] mitigate spam in email systems. Bazaar [59] alerts users about potential fraudulent transactions in online markets.

It is worth noting that in order for social network based Sybil defenses to be effective, the number of attack edges must be limited. However, obtaining social links in today's online social networks is relatively easy compared to that in the real world. In order to effectively defend against Sybil attack, these systems should derive more restricted online social networks from their existing ones. For example, the system can ask each user to select a constant number of trusted friends from their friend list in order to form the restricted network. A study [20] has shown that a normal human being cannot manage more than 150 friends efficiently. Another way is to derive the restricted network automatically from observing the interaction among users as suggested in [77].

1.2 Sybil-resilient admission control

The first contribution of this thesis improves the state-of-the-art defense for admission control protocols. Our new admission control protocol, called Gatekeeper, exploits the expander-like property of social graphs to achieve a better bound: admitting only $O(\log k)$ Sybil identities per attack edge, where k is the number of attack edges the adversary has. Compared to SybilLimit's bound of $O(\log n)$, this improvement is significant because the number of attack edges is negligible compared with the number of honest nodes (n) in the graph. When the number of attack edges is a constant, Gatekeeper admits only a constant number of Sybil identities, thereby achieving and the optimal result for social network based Sybil defense.

Gatekeeper introduces a technique called *distributed ticket distribution* to admit most honest nodes while limiting Sybil nodes. In distributed ticket distribution, a source node prepares $\Theta(n)$ number of ”admission” tickets and distributes them in a breath-first-search fashion. Upon receiving the tickets, each node keeps one ticket to itself and propagates the rest to its neighbors that are further away from the ticket source. When the number of attack edges (k) is small, Sybil nodes are far from a randomly chosen source with high probability because of the graph's expander-like property. As a result, the number of tickets which pass through an attack edge is a constant with high probability. In other words, the adversary only gets a constant number of admission tickets per attack edge, i.e. the system admits only a constant number of Sybil identity per attack edge. In order to admit most honest nodes, Gatekeeper combines the results of distributed ticket distributions at many ticket sources. Evaluation shows that Gatekeeper performs better than SybilLimit in both real world social graphs and synthetic social graphs.

1.3 Defending content voting systems against Sybil attack

Many websites rely on users' votes to rank user-submitted content. Example websites includes YouTube, Digg, and Reddit. A content voting system is susceptible to the Sybil attack because an adversary can outvote honest users using many Sybil identities. A Sybil-resilient vote aggregation system aims to collect most votes from honest users and as few votes from Sybil identities as possible for each piece of content.

One possible solution for vote aggregation is to first perform user admission control using Gatekeeper and then only collect votes from admitted users. The resulting vote aggregation system collects $O(k \log k)$ votes from Sybil identities in the face of k attack edges. We can improve this bound further by leveraging the fact that the number of honest users who vote for a specific content is very small compared with the total number of honest users. Therefore, a vote aggregation system needs to admit fewer number of users compared to an admission control system, which is a simpler task. Our vote aggregation system, SumUp, leverages this insight to collect only k votes from Sybil identities for each piece of content.

SumUp collects votes from users by computing a set of max-flow paths on the social graph from all voters to a known trusted identity called the vote collector. In order to perform max-flow computation, we need to set the flow capacity of each social link. One possible assignment is to set the capacity of every link to be one. Under such an assignment, the max-flow based vote collection can bound the attack capacity make between the set of trusted vote collectors and the adversary to be the number of attack edges (k), thus collecting at most k bogus votes from the adversary. However, this assignment also prevents most honest voters from having their votes collected, as the flow capacity between honest voters and trusted vote collectors is limited by the small number of immediate neighbors of the vote collectors. To address this limitation, we devised an assignment strategy which gives links close to vote collectors relatively higher capacities than links that are far away. Such an assignment enables max-flow based vote collection to gather most honest votes while still limiting the number of bogus votes to be the number of attack links with high probability. SumUp also incorporates users' feedback on the validity of collected votes to further improve its performance. In particular, if the adversary is found to have casted many bogus votes, SumUp eventually eliminates the adversary from the social network. SumUp offers immediate benefits to many popular websites that currently rely on users' votes to rank content.

We applied SumUp on the social graph and voting trace of Digg and found strong evidence of Sybil attacks. In particular, we identified hundreds of suspicious articles that have been promoted to the "popular" status on Digg by possible Sybil attacks.

1.4 Incentivizing bandwidth contribution in P2P CDN

Apart from containing the damage of malicious activities, cooperative systems must also incentivize contribution from honest but selfish users. Among today's cooperative systems, peer-to-peer content distribution networks (P2P CDNs) are particularly in need of incentive mechanisms to encourage nodes to contribute their upload bandwidth. The existing BitTorrent P2P CDN only provides incentives for nodes to upload to each other if they are actively downloading the same file. However, as soon as nodes finish downloading, they no longer have incentives to become a seeder and upload to others. Therefore, despite a large number of nodes that have previously downloaded a complete copy of the file, BitTorrent must rely on altruistic nodes provided by the content distributor to provide sufficient seeding capacity. Recently, there is a surge of private BitTorrent communities which enforce an exclusive user membership and minimal contribution requirements [71]. Our measurement shows that the average download bandwidth in private BitTorrent communities is 8 to 10 times higher than that in public BitTorrent communities because private BitTorrent communities have more seeders [71]. This performance disparity demonstrates the importance of providing seeding incentives.

User reputation systems have the potential to serve as a flexible incentive mechanism to incentivize bandwidth contribution in P2P CDNs. Users that have contributed more to the system obtain higher reputations which in turn entitle them to receive higher quality of service. e.g. higher download speed. The expectation that one's contribution is closely tied to his future service quality can be a strong incentive for contribution. Existing reputation systems suffer from two limitations. First, they are vulnerable to Sybil attacks and collusion [54]. In the Sybil attack, an adversary uses his Sybil identities to report fake downloads to increase his reputation. In collusion attack, many adversaries report fake downloads from each other in order to boost each other's reputation. Second, the reputation scores calculated by existing proposals do not accurately capture a user's past contribution.

To address both problems, we have designed the Credo reputation system [71] which encourages upload contribution by providing higher download speeds to nodes with higher reputation scores. In Credo, a node gives credits to those nodes that it has downloaded data from, and collects credits by uploading to others. Credo quantifies a node's net contribution based on the credits that the node has collected from others. By counting the diversity of those credits (i.e. the number of distinct issuers of credits) rather than quantity, Credo limits the maximum reputation score of an adversary to the number of his admitted Sybil identities, regardless of the number of credits issued by these Sybil identities. Since Credo uses Gatekeeper to manage user identities, an adversary can only have a few Sybil identities in the system. However, this technique alone cannot prevent multiple adversaries from colluding and sharing their Sybil identities with each other in order to increase the diversity. To address this collusion attack, Credo models the distribution of the amount of self-issued credits of honest users. This technique discourages Sybil identities from issuing abnormal amounts of credits compared with the modeled distribution. As a result, the Sybils issue similar amounts of credits compared with honest users. Therefore, each adversary in the collusion group receives a fixed amount of credits from the Sybils independent from the size of the collusion group.

We implemented Credo in the Azureus BitTorrent client. Experiments on PlanetLab have shown that Credo significantly improves the download times of most nodes by motivating nodes that have finished downloading a file to stay in the system and upload to others.

1.5 Contributions

To summarize, this thesis makes three contributions:

- The Gatekeeper Sybil-resilient admission control protocol for limiting the number of Sybil identities admitted into a cooperative systems. Gatekeeper's defense is optimal for the case of $O(1)$ attack edges and admits only $O(1)$ Sybil identities (with high probability). In the face of $O(k)$ attack edges (for any $k \in O(n/\log n)$), Gatekeeper admits $O(\log k)$ Sybils per attack edge.

- The SumUp Sybil-resilient vote aggregation system for limiting the number of bogus votes collected in an online content voting system. SumUp prevents the adversary from having more than k of his bogus votes collected if he has k attack edges. Using user feedback on votes, SumUp further restricts the voting power of adversaries who continuously misbehave to below the number of their attack edges.

- The Credo credit-based reputation system for incentivizing bandwidth contribution in P2P CDNs. Credo reputation can capture users' contribution precisely, and is resilient to Sybil and collusion attacks.

1.6 Thesis organization

The rest of this thesis starts with the description of Gatekeeper protocol and the comparison with SybilLimit in Chapter 2. We, then, describe the SumUp online content voting system and how to use SumUp to find real world Sybil attacks on Digg in Section 3. We explain the Credo reputation system in Chapter 4. The evaluation of each system is presented in its corresponding chapter. We summarize related work in Chapter 5 before concluding and discussing future work in Chapter 6.

Chapter 2

Optimal Sybil-resilient admission control

Open systems like Digg, Youtube, Facebook and BitTorrent allow any user on the Internet to join the system easily. Such lack of strong user identity makes these open systems vulnerable to Sybil attacks [19], where an attacker can use a large number of fake identities (Sybils) to pollute the system with bogus information and affect the correct functioning of the system. The only known promising defense against Sybil attacks is to use social networks to perform user admission control and limit the number of bogus identities admitted to the system [83, 84, 18, 73]. A link in the social network between two users represents a real-world trust relationship between the two users. It is reasonable to assume that an attacker usually has few links to honest users since establishing trust links requires significant human efforts. Therefore, **Sybil-resilient admission control** can be stated as follows: Consider a social network $\mathcal{G}$ consisting of n honest users and arbitrarily many Sybils connected to honest nodes via k attack edges (an *attack edge* is a link between an honest and a Sybil node). Given an honest node acting as the admission controller, determine the set of nodes to be admitted so that the vast majority of honest nodes in $\mathcal{G}$ are admitted and few Sybil nodes are admitted.

The knowledge of the social graph $\mathcal{G}$ may reside with a single party or be distributed across all users. Centralized node admission assumes complete knowledge of $\mathcal{G}$ (e.g. SybilInfer [18] and SumUp [73]) while distributed admission control only requires each user/node to be initially aware of only its immediate neighbors in $\mathcal{G}$ and seeks to discover all the other honest users/nodes in $\mathcal{G}$. This paper addresses the distributed node admission control problem.

We make a few important observations about the Sybil-resilient node admission problem. First, the problem is inherently probabilistic in its definition; hence, we seek to admit *most* honest nodes while limiting Sybil nodes. Finding a perfect algorithm that can detect all honest nodes and reject all Sybil nodes is fundamentally impossible. Second, the problem makes no assumption about n, the number of honest nodes in $\mathcal{G}$. As we show in our result, if the social network exhibits expander-graph properties, one does not require the knowledge of n to solve the problem. Third, any distributed admission control protocol can also be run in a centralized setting and hence is more general than centralized admission control.

The distributed admission control problem has been studied in prior work. SybilGuard [84] is the first work to show an admission protocol which limits the number of admitted Sybil identities to be $O(\sqrt{n}\log n)$ per attack edge, where n is the number of honest users in the social network. SybilLimit [83] significantly improves over SybilGuard and limits the number of Sybils admitted per attack edge to $O(\log n)$.

In this paper, we present a distributed Sybil-resilient admission control protocol called Gatekeeper with the following results:

Theorem: *Given a social network $\mathcal{G}$ which exhibits a random expander-graph prop-*

erty, Gatekeeper achieves the following properties with high probability:

1. *In the face of k attack edges with k up to $O(n/\log n)$, Gatekeeper limits the number of admitted Sybil identities to be $O(\log k)$ per attack edge. This implies that only $O(1)$ Sybil nodes are admitted per attack edge if the attacker has $O(1)$ attack edges.*

2. *Gatekeeper admits almost all honest users.*

To achieve these results, Gatekeeper uses an improved version of the *ticket distribution* algorithm in SumUp [73] to perform node admission control in a decentralized fashion. Gatekeeper executes the ticket distribution algorithm from multiple randomly chosen vantage points and combines the results to perform decentralized admission control. We prove the results under the assumption of random expander graphs, an assumption that holds for many existing social networks. Expander graphs are by nature fast-mixing, a common assumption made in SybilLimit and other related protocols [83, 84, 18, 36].

Our result establishes optimality and improves over SybilLimit by a factor $\log n$ in the face of $O(1)$ attack edges. Under constraints that attack edges are hard to establish and there is only a constant number of them, Gatekeeper is an optimal decentralized protocol for the Sybil-resilient admission control problem. The general result on admitting $O(\log k)$ Sybils per attack edge in the face of k attack edges for any $k \in O(n/\log n)$ establishes a continuum across the attack capacity spectrum. This provides a graceful degradation with increased number of attack edges. In the worst case when $k = O(n/\log n)$, Gatekeeper achieves the same level of resilience as SybilLimit where both Gatekeeper and SybilLimit admit $O(\log n)$ Sybils per attack edge with high probability.

We have tested our protocol experimentally on real-world social networks and synthetic topologies for varying number of attack edges. Our analysis shows that our protocol is able to drastically limit the number of admitted Sybil identities to a very small number while admitting almost all honest identities. Even when we significantly increase the number of attack edges to cover $\sim 2\%$ of the nodes, the number of admitted Sybil identities per attack edge remains very small.

2.1 System Model and Threat Model

We use a similar system model and threat model as those used in previous systems (e.g. SybilLimit [83], SybilGuard [84] and Whānau [36]). The system consists of n honest nodes belonging to n honest users. There exists an undirected social graph among all nodes in the system. A link between two honest users reflects the trust relationship between those users in the real-world. The knowledge of the social graph is distributed among all nodes. In particular, each honest node knows its immediate neighbors on the social graph and may not know the rest of the graph, including the value of n. Each node has a locally generated public/private key pair. A node knows the public-keys of its neighbors, however, there exists *no* public-key infrastructure that allows a node to correctly learn of all other nodes' public-keys.

The system also has one or more malicious users and each malicious user controls a number of malicious Sybil nodes. All Sybil nodes may collude with each other and hence are collectively referred to as the adversary or attacker. Honest nodes behave according to the protocol specification while Sybil nodes are assumed to behave in a Byzantine fashion. The attacker may know the entire social graph and is able to create arbitrary links among his Sybil nodes. We assume the attacker has k links with honest users (attack edges), where k can be up to $O(n/\log n)$.

Distributed admission control: A node acting as an admission controller determines which of the other nodes (suspect nodes) should be admitted into the system. The process can either be creating a list of admitted nodes, or deciding whether a particular suspect node can be admitted or not. In the centralized setting, one typically assumes the existence of a trusted controller that performs admission control on behalf of all nodes. By contrast, in the distributed setting, there exists no centralized source of trust and each node must act as its own controller. Each controller needs to consult other nodes to make its admission decisions. We note that a node acts as its own controller as well as a suspect for other controllers.

Sybil-resilient node admissions: The goal of Sybil-resilient admission is two-fold – it should accept most honest nodes and it should admit few Sybil nodes. The attacker aims to maximize the number of admitted Sybil nodes, and to minimize the number of admitted honest nodes.

It is worth emphasizing that the number of admitted Sybil nodes is ultimately dependent on k, the number of attack edges. Specifically, since attack edges are indistinguishable from honest edges, any protocol that admits most honest nodes would admit approximately one Sybil node per attack edge, resulting in k admitted Sybil nodes. The goal of a Sybil-resilient admission protocol is to approach this lower bound of one admitted Sybil node per attack edge. Separate mechanisms are required to ensure that k is likely to be small. Today's popular online social networks like Facebook do not promise small k. To minimize k, one can use techniques proposed in [3] and [77] to ensure that honest users only establish trust links with their close friends in the real-world so that the attacker is unlikely to possess many links to honest users, resulting in a small k.

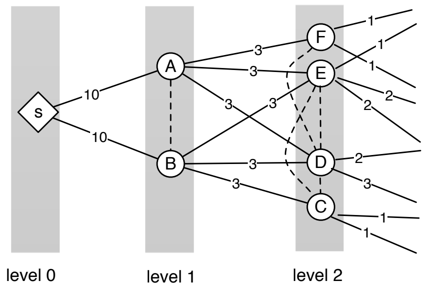


Figure 2.1: The ticket distribution process of a particular node S: The number on each link represents the number of tickets propagated via that link. The dotted lines are links between nodes at the same distant to the source.

2.2 Design Overview

In this section, we first describe the central component of Gatekeeper, the *ticket distribution process*. We proceed to discuss the challenges involved in using ticket distribution for node admission control and explain how Gatekeeper addresses these challenges.

2.2.1 Ticket Distribution

The principle building block of Gatekeeper is the ticket distribution process where each node acting as a ticket source disseminates t “tickets” throughout the social network until a significant portion of the honest nodes receive some tickets. We originally designed the distribution algorithm for SumUp [73], a *centralized* Sybil-resilient vote collection system. SumUp uses ticket distribution to assign link capacities which are needed for its centralized max-flow computation. As we will see later, Gatekeeper uses ticket distribution completely differently.

We illustrate the ticket distribution process using the example of Figure 2.1 where the ticket source (S) intends to disseminate $t = 20$ tickets. Tickets propagate in a breadth-first-search (BFS) manner: Each node is placed (conceptually) at a BFS-level according to its shortest-path distance from S. S divides the tickets evenly and sends them to its neighbors. Each node keeps one ticket to itself and distributes the rest evenly among its neighbors at the next level. In other words, a node does not send tickets back to neighbors that are at the same or smaller distance to the source. If a node does not have any outgoing links to the next level, it simply destroys all remaining tickets. The process continues until no tickets remain.

We use ticket distribution as a fundamental building block in Gatekeeper because of two considerations. First, since each node only needs knowledge of its immediate neighborhood to propagate tickets, the entire distribution process can be realized in a completely distributed manner. Second, as nodes propagate tickets in a BFS manner from the source, edges further away from the ticket source receive exponentially fewer tickets. Our intuition is that, since the attacker only controls a small number of attack edges, a randomly chosen ticket source is relatively "far away" from most attack edges, resulting in few tickets propagated along an attack edge. As a result, Gatekeeper may be able to directly use a received ticket as a token for a node's admission.

2.2.2 Our approach

The naïve strategy for applying ticket distribution to admission control works as follows: each node admission controller (S) disseminates n tickets and accepts a suspect node if and only if it has received some tickets from S. Such a strategy has two inherent limitations. First, it is infeasible to reach the vast majority (e.g. $> 99\%$) of honest nodes by distributing n tickets from a single ticket source. For example, the simulation experiments in [73] shows that only $\sim 60\%$ honest nodes receive some tickets. Second, in the presence of a single ticket source, an attacker may be able to strategically acquire some attack edge close to the source, resulting in a large amount of tickets being propagated to Sybil nodes via that attack edge.

The key idea of Gatekeeper is to perform distributed ticket distribution from *multiple* ticket sources. In Gatekeeper, an admission controller explicitly picks m random nodes (using the random walk technique in [83]) to act as ticket sources. Each randomly chosen ticket source distributes t tickets where t is chosen such that $\frac{n}{2}$ nodes receive some tickets. Later in Section 2.5, we will show that a source only needs to send out $t = \Theta(n)$ tickets. We say that a node is *reachable* from a ticket source if it has received a ticket disseminated by the source. The admission controller admits a suspect node if and only if the node is reachable from at least $f_{admit} \cdot m$ ticket sources, where f_{admit} is a small constant (our evaluations suggest using $f_{admit} = 0.2$).

Multi-source ticket distribution addresses both limitations associated with using a single ticket source. The first limitation is concerned with a single source not being able to reach the vast majority of honest nodes by sending only $t = \Theta(n)$ tickets. In Gatekeeper, an honest node not reachable from one source may be reached by other sources. Ultimately, an honest node is admitted as long as it is reachable by $f_{admit} \cdot m$ sources which is a high probability event. On the other hand, with a small number of attack edges, the attacker cannot appear close-by to many m randomly chosen sources, and thus is unlikely to receive a large number of tickets from as many as $f_{admit} \cdot m$ sources. Therefore, by admitting only nodes reachable by $f_{admit} \cdot m$ sources, Gatekeeper ensures that the number of admitted Sybil nodes per attack edge is small. The second limitation is concerned with an attacker strategically acquires some attack edge close to a known ticket source. Gatekeeper solves this problem because the admission controller explicitly picks m random ticket sources as opposed to acting as the ticket source itself. In Section 2.5, we present a detailed analysis of these intuitions.

2.3 Gatekeeper: The protocol

The Gatekeeper protocol consists of two phases: a *bootstrap phase* where each node acts as a ticket source to disseminate $\Theta(n)$ tickets throughout the network and an *admission phase* where a node acting as the admission controller selects m ticket sources and accepts another node if that node possesses tickets from $f_{admit} \cdot m$ of the m chosen sources. Below, we describe the details of these two phases:

2.3.1 Bootstrap: decentralized ticket distribution

To bootstrap the protocol, every node performs decentralized ticket distribution with the aim of reaching more than half of the honest nodes. Since ticket distribution proceeds in a BFS fashion, a forwarding node needs to know its neighbor's "level" (i.e. the neighbor's shortest path distance to the ticket source) in order to decide whether to forward that neighbor any tickets. In order to establish such shortest path knowledge, all nodes execute a secure path-vector based routing protocol. We adopt a known secure path-vector protocol [29] where a node explicitly advertises its shortest path to each ticket source using a signature chain signed by successive nodes along the path. As a result, Sybil nodes cannot disrupt the shortest path calculation among honest nodes.

The number of tickets a source should disseminate, t, is not a fixed parameter. Rather, each source adapts t iteratively by estimating whether a sufficiently large fraction of nodes receive some tickets under the current value of t. We first describe how a source S disseminates t_j tickets in the j-th iteration and discuss how S adapts t_j later. Each ticket from S consists of the current iteration number j, a sequence number $i \in [1..t_j]$, and a message authentication code (MAC) generated using the private key of S. The MAC is verifiable by the source and is necessary to prevent the forgery and tampering of tickets.

A node Q receiving r tickets consumes one of them and evenly divides the other $r-1$ tickets to those neighbors at the next BFS "level", i.e. neighbors that are further away from S than Q. Node Q can learn which neighbors are further by requesting and verifying its neighbors' shortest path signatures. If Q has no such neighbor, it simply discards its remaining tickets. When Q sends a ticket to its neighbor R, it explicitly transfers the ownership of that ticket by appending a tuple $\langle Q, R\rangle$ to the ticket and signing the ticket with Q's private key. If Q consumes a ticket, it appends itself $\langle Q*\rangle$ to denote the end of the transfer chain. The use of a signature chain allows a ticket source to detect a "double-spender", i.e. a malicious node that has sent the same ticket to different nodes. The signature chain scheme represents one of many solutions for detecting double-spenders. Alternative mechanism include secure transferable e-cash schemes [10]) which allow a source node to act as a "bank" issuing e-coins as tickets.

In order to help source S determine its reachable nodes, each node that has consumed a ticket from S forwards its ticket in the reverse direction of the ticket's signature chain. Suppose S receives a ticket consumed by Q, S must verify the validity of the signature chain associated with that ticket. In particular, S checks that the chain is not "broken", e.g., $\langle S, A\rangle, \langle A, B\rangle, \langle C, Q\rangle$ is not valid because it misses the link $\langle B, C\rangle$. Additionally, S also checks in its database of received tickets to see if there is any double spending. For example, if S discovers two tickets $(\langle S, A\rangle, \langle A, B\rangle, \langle B, Q\rangle)$ and $(\langle S, A\rangle, \langle A, B'\rangle, \langle B', Q'\rangle)$, it will blacklist node A as a double-spender and ignore both tickets. If Q's ticket passes verification, S records Q in its database of reachable nodes.

Adjust the number of tickets distributed iteratively: After a pre-defined time period, the ticket source terminates the current j-th iteration of ticket distribution, and decides if it needs to proceed with the $(j+1)$-th iteration with increased number of tickets to be distributed. In particular, the ticket source samples a random subset (W) of nodes in the social network by performing a number of random walks. Let R be the set of reachable nodes in the source's database. If less than half of the sampled nodes are within the reachable set, i.e. $\frac{|R \cap W|}{|W|} < 1/2$, the source proceeds to the next iteration ($j + 1$) with twice the amount of tickets, $t_{j+1} = 2 \cdot t_j$.

Intuitively, when the attacker controls up to $O(n/\log n)$ attack edges, only a negligible fraction of nodes ($o(1)$) are Sybils in the sampled set (W) and the reachable set (R). As a result, if the majority of the sampled nodes (W) are not in R, it implies that the amount of tickets distributed in the current iteration is insufficient and the source should distribute more tickets in the next iteration. On the other hand, once the amount of tickets distributed reaches $\Theta(n)$, the majority of honest nodes become reachable, thereby terminating the iterative process.

Our adaptive ticket adjustment process is similar to the benchmarking technique used in SybilLimit [83]. In SybilLimit, each node performs $O(\sqrt{n})$ random walks and benchmarking is used to determine the number of random walks to perform without explicitly estimating n. Similarly, in Gatekeeper, each ticket source adaptively decides on the amount of tickets to distribute ($t = \Theta(n)$) without having to explicitly estimate n.

2.3.2 Node admission based on tickets

After all ticket sources have bootstrapped, each node can carry out its own admission control to decide upon a list of nodes to be admitted into the system.

To perform admission control, a controller first selects m random ticket sources by performing m random walks of length $O(\log n)$. In fast-mixing social networks, a random walk of length $O(\log n)$ reaches a destination node drawn from the nodestationary distribution. Because nodes have varying degrees, a forwarding node i picks neighbor j as the random walk's next hop with a probability weight of $\min(\frac{1}{d_i}, \frac{1}{d_j})$, where d_i and d_j are the degree of node i and j, respectively. This ensures that m random walks sample m nodes uniformly at random [18]. It is in the attacker's best interest to claim that Sybil nodes have degree 1 in order to attract random walks into the Sybil region. To protect an unlucky controller who is a friend or a friend-of-a-friend of some Sybil node, we make an exception for honest nodes to forward random walks to its neighbors with *equal* probability during the first two hops of a random walk. We use the same strategy in [83] to estimate the required random walk length without the knowledge of n.

The controller asks each of the m chosen ticket sources for its reachable node list. The controller admits a node if and only if that node has appeared in more than $f_{admit} \cdot m$ reachable lists returned by the m chosen ticket sources. The parameter f_{admit} is set to a fixed value 0.2 in our simulations and we will analyze how to set the appropriate value for f_{admit} in Section 2.5.

2.4 Protocol Message Overhead

We consider the asymptotic message overhead of Gatekeeper when every node acts as a controller and compare to that of SybilLimit. During the bootstrap phase, the number of bits that need to be transferred during the ticket distribution process of a single source is $\Theta(n \log n)$ because the source sends out $\Theta(n)$ tickets and each ticket travels a path of length $\Theta(\log n)$. Therefore, in a network of n ticket sources, the total message overhead is $\Theta(n^2 \log n)$. In the admission phase, each controller obtains m node lists each of size $\Theta(n)$ from m chosen ticket sources. When each node acts as a controller, the total number of bits transferred during the admission phase is $\Theta(n^2)$. Thus, the total message overhead incurred by Gatekeeper is $\Theta(n^2 \log n) + \Theta(n^2) = \Theta(n^2 \log n)$. This overhead is the same as that of SybilLimit if every honest node aims to admit every other honest node. However, we must point out that if each controller only intends to admit a small constant of honest nodes, SybilLimit incurs only $\Theta(n\sqrt{n}\log n)$ total overhead. By contrast, the total overhead in Gatekeeper is always $\Theta(n^2 \log n)$ regardless of the number of honest nodes each controller intends to admit.

In some circumstances, it may be desirable to run Gatekeeper in a centralized setting using a single admission controller. For example, the online content voting site, Digg.com, may run Gatekeeper on its social graph using a single controller to decide upon the list of nodes allowed to cast votes. In these cases, Gatekeeper's overall runtime is $\Theta(n \log n)$, which is much better than that of SybilLimit ($\Theta(n\sqrt{n}\log n)$).

2.5 Security Analysis

We show Gatekeeper's Sybil-resilience by proving that, if the attacker possesses $k = O(n/\log n)$ randomly injected attack edges, a controller admits at most $O(\log k)$ Sybil nodes per attack edge and that each controller admits almost all honest nodes. Our proof makes certain assumptions about the social graph formed by honest users, denoted by $\mathcal{G}$. Specifically, we assume that:

1. $\mathcal{G}$ is a fixed degree sequence random graph constructed by the pairing method in [5, 43] with maximum node degree d. It has been shown that the pairing method generates an expander graph with expansion factor α with high probability. In other words, for every set W of vertices with fewer than $n/2$ nodes, $|N(W)| \geq \alpha|W|$ where $N(W)$ denotes the set of vertices adjacent to W but do not belong to

W [2]. Compared to previous work which only assumes fast-mixing graphs [84, 83, 36], expanders represent a stronger assumption. Nevertheless, expander has been commonly used as reasonable model for large-scale social graphs.

2. $\mathcal{G}$ is reasonably balanced. Let $\Delta_{half}(v)$ be the distance such that v is less than $\Delta_{half}(v)$ distance away from more than half of the honest nodes. In other words, $\Delta_{half}(v)$ is the BFS-level when v reaches more than $\frac{n}{2}$ nodes. Let $dist(u, v)$ be the distance between u, v. Define $S(v) = \{u|u \in \mathcal{G}, dist(u, v) \leq \Delta_{half}(u)\}$. In other words, $S(v)$ represents the set of ticket sources that deem v as reachable. We say $\mathcal{G}$ is balanced if for almost all v, $\frac{|S(v)|}{n} > f_{th}$ for a constant threshold value $f_{th} < 0.5$. In probabilistic terms, $\Pr(\frac{|S(v)|}{n} < f_{th})$ for any randomly chosen v is $o(1)$ (a function asymptotically lower than a constant). Most real world social graphs satisfy this balance criterion.

2.5.1 Gatekeeper admits $O(\log k)$ Sybils per attack edge

For this proof, we proceed in two steps: first, we bound the number of tickets sent to the attacker (via k attack edges) by a randomly chosen ticket source to $O(k \log k)$. Second, we show at most $O(\log k)$ Sybil nodes can receive tickets from more than $f_{admit} \cdot m$ of the m ticket sources using the Chernoff bound.

The more tickets a source distributes, the more tickets that likely end up with the attacker. Therefore, in order to bound the number of tickets received by the attacker, we must bound the number of tickets distributed by a ticket source, as described formally by the following theorem:

Theorem 2.1. *Suppose the graph $\mathcal{G}$ is a fixed degree sequence random graph constructed by the pairing method. The expected number of tickets required by a given* *ticket source to reach more than $n/2$ honest nodes is $E[t] = \Theta(n)$. (see proof in Appendix A)*

Given a ticket source u, we order honest nodes from closest to farthest from u according to their BFS level. Let Δ_{small} be the level of the $\frac{\epsilon}{k} \cdot n$-th node, where ϵ is a small constant like 0.01. Let Δ_{big} be the level of the $(1 - \frac{\epsilon}{k}) \cdot n$-th node. In other words, $\Delta_{small}, \Delta_{big}$ are chosen so that the BFS levels of $1 - \frac{2\epsilon}{k}$ fraction of honest nodes fall between $(\Delta_{small}, \Delta_{big}]$. As a result, the probability that all k attack edges are at some distance within the range $(\Delta_{small}, \Delta_{big}]$ is $(1 - \frac{2\epsilon}{k})^k > 1 - 2\epsilon$, which is high because of small ϵ. Next, we will bound the number of tickets received by the attacker for the high probability event that all k attack edges lie within ticket distribution levels $(\Delta_{small}, \Delta_{big}]$.

Lemma 2.2. *For a given ticket source u, given that all k randomly injected attack edges are at some distance in the range $(\Delta_{small}, \Delta_{big}]$ from u, the expected number of tickets received by the attacker is $O(k \log k)$.*

Proof. Let A_i be the number of u's tickets that are sent from level-i to level-$(i + 1)$. $A_0 = t$ is the number of tickets distributed by the source u. Let L_i be the number of nodes at level-i. We can calculate the expected number of tickets that pass though a random node at level $(\Delta_{small}, \Delta_{big}]$ as:

$$\sum_{\Delta_{small}+1}^{\Delta_{big}} \frac{A_{i-1}}{L_{(\Delta_{small}+1)} + \cdots + L_{\Delta_{big}}} \tag{2.1}$$

$$\leq (\Delta_{big} - \Delta_{small}) \frac{A_0}{L_{(\Delta_{small}+1)} + \cdots + L_{\Delta_{big}}} \tag{2.2}$$

By the definition of Δ_{small} and Δ_{big}, we know that $(L_{(\Delta_{small}+1)} + \cdots + L_{\Delta_{big}})$ has greater than $(1 - \frac{2\epsilon}{k})$ fraction of honest nodes. Furthermore, $E(A_0) = E(t) = \Theta(n)$

according to Theorem 2.1. Hence, $\frac{A_0}{L_{\Delta_{small}+1}+\cdots+L_{\Delta_{big}}} = O(1)$.

To show that $(\Delta_{big} - \Delta_{small})$ is $O(\log k)$ we consider the two terms $(\Delta_{half} - \Delta_{small})$ and $(\Delta_{big} - \Delta_{half})$ where Δ_{half} is the level where we reach the $n/2$-th node in the BFS tree of u. Because $\mathcal{G}$ is an expander with expansion factor α across each level, we have $\frac{\epsilon}{k}n \cdot \alpha^{\Delta_{half}-\Delta_{small}} \leq n/2$. Hence $\Delta_{half} - \Delta_{small}$ is $O(\log k)$. Similarly, we can bound $\Delta_{big} - \Delta_{half}$ to $O(\log k)$ by expanding from graph from the $\frac{\epsilon}{k}n$ nodes farthest from u to the $\frac{n}{2}$-th node. Summing up the two results, we get $(\Delta_{half} - \Delta_{small})$ as $O(\log k)$. Hence, we can bound the expected number of tickets received by a random node within the level range $(\Delta_{small}, \Delta_{big}]$ to be $O(\log k)$. Since an attack edge is connected to a random node at level within the range $(\Delta_{small}, \Delta_{big}]$, the expected number of tickets received by an attack edge is bounded by $O(\log k)$. Hence, with k attack edges all within this range, the expected number of tickets received by the attacker is $O(k \log k)$.

$\square$

Based on Lemma 2.2, a ticket source gives $O(k \log k)$ tickets to the attacker with k attack edges. However, the $O(k \log k)$ bound is only in expectation and some ticket sources may give much more than the expected number of tickets to the attacker. By requiring each admitted node to receive tickets from at least $f_{admit} \cdot m$ of m randomly chosen sources, we can prove the following theorem:

Theorem 2.3. *Gatekeeper admits $O(\log k)$ Sybils per attack edge with high probability.*

Proof. Let T_1, $T_2, \cdots, T_m$ be the random variables representing the total number of tickets received by the attacker via k attack edges from each of the m ticket sources. Since $E(T_i) = O(\log k)$, according to Markov's inequality, there exist constants, $\beta > 1$ and $\tau < \frac{f_{admit}}{2}$, such that $\Pr(T_i > \beta k \log k) \leq \tau$. In other words, the probability that any ticket source reaches more than $\beta k \log k$ Sybil nodes is bounded by τ.

We define a new random variable, Z_i, as follows:

$$Z_i = \begin{cases} 1 & \text{if } T_i \geq \beta k \log k \\ 0 & \text{if } T_i < \beta k \log k \end{cases}$$

Let $z = Z_1 + Z_2 + \cdots + Z_m$. Since $\Pr(Z_i = 1) < \tau$, using Chernoff bound, we can show that

$$\Pr(z \geq \frac{m f_{admit}}{2}) \leq e^{-m \cdot D(\tau, \frac{f_{admit}}{2})}$$

where $D(\tau, \frac{f_{admit}}{2})$ is the Kullback-Leibler divergence function that decreases exponentially with m. Hence, with high probability, $z \leq \frac{m f_{admit}}{2}$. We refer to the i-th source as type-A if $Z_i = 1$ or as type-B if $Z_i = 0$. Among the m sources, there are z type-A sources and $m - z$ type-B sources.

Suppose $\bar{s}$ Sybil nodes are finally admitted. In order to be admitted, each of the $\bar{s}$ Sybils can present at most z tickets from type-A sources. Additionally, *all* $\bar{s}$ Sybils can use at most $(m - z)\beta k \log k$ tickets from type-B sources. Hence, the total number of tickets that can be used for the admission of $\bar{s}$ Sybils is at most $\bar{s}z + (m - z)\beta k \log k$. Since $\bar{s}$ Sybils need at least $\bar{s} f_{admit} \cdot m$ tickets for admission, we arrive at the following inequality:

$$\begin{aligned} \bar{s} \cdot f_{admit} \cdot m &\leq \bar{s} \cdot z + (m - z) \cdot \beta k \log k \\ \bar{s} &\leq \frac{(m - z)}{f_{admit} \cdot (m - z)} \beta k \log k \\ \bar{s} &\leq \frac{2 - f_{admit}}{f_{admit}} \beta k \log k \\ \frac{\bar{s}}{k} &= O(\log k) \quad \Box \end{aligned}$$

□

2.5.2 Gatekeeper admits most honest nodes

Theorem 2.4. *Gatekeeper admits any honest node with high probability.*

Proof. Recall our earlier definition of $S(v)$, which represents the set of potential ticket sources that deem v as reachable. Since $\mathcal{G}$ is balanced, the probability that a randomly chosen ticket source can reach v is at least f_{th}. Since the events that v is reachable from randomly chosen ticket sources are independent, we can apply the Chernoff bound to show that the probability v is reachable from less than $f_{admit} \cdot m$ ticket sources is bounded by $e^{-m \cdot D(f_{admit}, f_{th})}$ where $D(\cdot)$ is the Kullback-Leibler divergence function. Thus, when choosing f_{admit} such that $f_{admit} < f_{th}$, the probability that an honest node is not admitted decreases exponentially with m. Hence, Gatekeeper admits an honest node with high probability. □

Note that we have proved both Theorem 2.3 and Theorem 2.4 for the case when *all* m ticket sources are honest. A Sybil node may be chosen as a source if a random walk escapes to the Sybil region of the graph. Let f_{esc} be the fraction of m sources in the Sybil region. When the attacker controls up to $O(n/\log n)$ attack edges, with high probability, f_{esc} is asymptotically smaller than a constant, i.e. $f_{esc} = o(1)$. Our earlier proofs can be extended to handle $f_{esc} = o(1)$. Next, we analyze the worst case scenario when f_{esc} is non-negligible.

2.5.3 Worst Case Analysis

The worst case scenario applies to those few unlucky controllers that are extremely close to some attack edge, resulting in a non-negligible f_{esc}. Let m' be the number of honest sources, i.e. $m' = (1 - f_{esc}) \cdot m$. We adjust the proof for Theorem 2.3 to handle the case when only m' sources are honest. For each of the $\bar{s}$ Sybils to be admitted, it can use at most z tickets from type-A ticket sources and at most $(m' - z) \cdot \beta k \log k$ from type-B sources. Additionally, $\bar{s}$ Sybils can use $\bar{s} f_{esc} \cdot m$ from those ticket sources in the Sybil region. Recall that $z < \frac{f_{admit} m'}{2}$, we have:

$$\begin{aligned}
\bar{s} \cdot f_{admit} \cdot m &\leq \bar{s} f_{esc} m + \bar{s} z + (m' - z)\beta k \log k \\
\bar{s}((f_{admit} - f_{esc})m - z) &\leq (m' - z)\beta k \log k \\
\Rightarrow \frac{\bar{s}}{k} &\leq \frac{(1 - f_{esc}) \cdot m - z}{(f_{admit} - f_{esc})m - z} \beta \log k
\end{aligned}$$

Therefore, to admit at most $O(\log k)$ Sybils per attack edge (i.e. $\frac{\bar{s}}{k} = O(\log k)$), the escape probability f_{esc} must be small enough such that $(f_{admit} - f_{esc}) \cdot m - z > 0$. Since $z < \frac{f_{admit} m'}{2}$, we obtain that $f_{esc} < \frac{f_{admit}}{2 - f_{admit}}$.

We adjust the proof of Theorem 2.4 similarly. In order for an honest node to be admitted, it must possess tickets from $f_{admit} \cdot m$ nodes out of the m' honest sources. Therefore, we require $f_{admit} < (1 - f_{esc}) f_{th}$, i.e. $f_{esc} < 1 - \frac{f_{admit}}{f_{th}}$. In summary, to satisfy both Theorem 2.3 and 2.4, we require that $f_{admit} < \min(\frac{f_{admit}}{2 - f_{admit}}, \frac{1 - f_{admit}}{f_{th}})$.

We use $f_{admit} = 0.2$ in our evaluations. Therefore, a controller admits $O(\log k)$ Sybil nodes per attack edge as long as $f_{esc} < 0.11$. As a concrete example, let us consider a controller with degree d who is immediately adjacent to the attacker. In this case, $f_{esc} = 1/d$. Hence if d is bigger than 9, f_{esc} will be small enough to satisfy both Theorem 2.3 and 2.4. If d is smaller than 9, the controller must be more than 1-hop away from the attacker to ensure that f_{esc} is small enough.

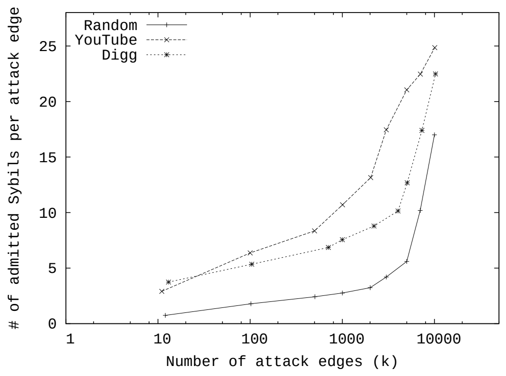


Figure 2.2: The number of Sybil nodes accepted per attack edge as a function of the number of attack edges (k).

Data set	Synthetic	YouTube [46]	Digg [73]
Nodes	varying	$446,181$	$539,242$
Undirected edges	varying	$1,728,948$	$4,035,247$
Average:median degree	$6:6$	$7.7:2$	$15:2$

Table 2.1: Social graph statistics

2.6 Evaluation

We evaluate the effectiveness of Gatekeeper in both synthetic graphs and real-world social network topologies. Specifically, we show that Gatekeeper admits most honest nodes ($> 90\%$ across different topologies) and significantly limits the number of Sybils admitted per attack edge to a small value even in the face of a large number of attack edges ($k \approx 0.02 \cdot n$).

2.6.1 Experimental Methodology

For real-world social topologies, we use the YouTube [46] and Digg [73] graph. For synthetic graphs, we generate random graphs with average node degree of 6. Table 2.1 summarizes the basic graph statistics. To model the Sybil attack, we randomly choose a fraction of nodes to collude with the attacker so that all the edges of these nodes as attack edges. The attacker optimally allocates tickets to Sybils to maximize the

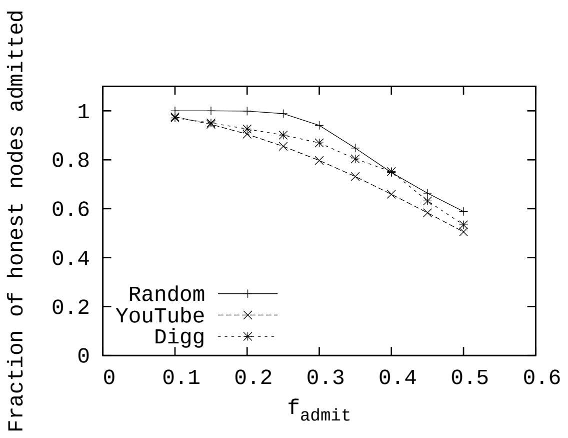


Figure 2.3: Fraction of honest nodes admitted under varying f_{admit}

number of Sybils admitted. In each simulation run, we randomly select a controller to perform admission control and measure the number of Sybils admitted per attack edge and the number of honest nodes admitted. We repeat each experiment for 2000 runs and compute the average and the deviation. Unless otherwise mentioned, a controller uses $m = 100$ ticket sources and admits another node if it has received tickets from at least $f_{admit} = 0.2$ fraction of the m sources.

2.6.2 Number of Sybils admitted

We first measure the number of Sybil nodes admitted per attack edge as a function of the number of attack edges (k). Figure 2.2 shows the number of admitted Sybil nodes as a function of k for a random graph with $500,000$ nodes, the YouTube graph and the Digg graph. Our theoretical result shows that Gatekeeper admits $O(\log k)$ Sybils per attack edge. Figure 2.2 confirms our analysis showing that the number of Sybils admitted per attack edge increases very slowly with k; even when k reaches 2% of the network size (i.e. $k = 10,000$), the number of Sybils nodes accepted per attack edge remains smaller than 25.

SybilLimit			
Dataset	Synthetic ($n = 500,000$)	YouTube	Digg
Parameter w	12	15	14
Parameter r	3200	3400	5100
Sybils admitted per attack edge	40.3	49.1	45.1
Gatekeeper			
f_{admit}	0.2	0.15	0.15
Sybils admitted per attack edge	1.5	4.9	7.1

Table 2.2: Comparison with SybilLimit

Unlike SybilLimit, Gatekeeper's bound on Sybils admitted per attack edge ($O(\log k)$) is independent of the network size n for a given k. We have verified this property by running Gatekeeper on random graphs with different network sizes.

Comparison with SybilLimit: We compare the performance of Gatekeeper and SybilLimit under both synthetic and real graph topologies with $k = 60$ attack edges. In separate experiments, we find the parameter values so that both Gatekeeper and SybilLimit admit $> 95\%$ honest nodes and use these values in our comparison.

Table 2.2 summarizes the parameter values used in each protocol and the number of Sybils admitted per attack edge. As we can see, SybilLimit admits $40 - 50$ Sybils per attack edge across all the three topologies, while Gatekeeper admits only $1 - 7$ Sybils nodes per attack edge. Therefore, Gatekeeper represents a significant improvement over SybilLimit in practical settings.

Compared to the random graph case, Gatekeeper accepts more Sybil nodes on the YouTube and Digg graphs because real-world graphs can exhibit certain asymmetries that are not present in a random graph. Because of this asymmetry, more tickets are dropped at some node with no neighbors at the next BFS-level. Having more ticket drops in turn causes a ticket source to send more tickets in order to reach more than half of honest nodes. As a result, attack edges also receive more tickets, thereby causing more Sybils to be admitted.

2.6.3 Admitting honest nodes

The parameters f_{admit} and m affect the fraction of honest nodes admitted by Gatekeeper. Choosing the appropriate f_{admit} is dependent on the balance properties of the graph. Figure 2.3 measures the fraction of honest nodes admitted for different values of f_{admit} under various topologies. We can see that larger f_{admit} results in fewer honest nodes being admitted. On the other hand, smaller f_{admit} will increase the number of Sybils admitted by a constant factor. Since synthetically generated random graphs are more balanced than YouTube and Digg graphs, Gatekeeper admits higher fraction of honest nodes in the random graph than in YouTube and Digg graph for the same value of f_{admit}. When $f_{admit} = 0.2$, Gatekeeper can admit more than 90% honest nodes in all three graphs. Hence, we use 0.2 as the default value for f_{admit}. We have also experimented with varying m and found that $m = 100$ was sufficient to admit most honest nodes across different topologies. Setting m to be bigger than 100 yields diminishing returns.

2.6.4 Worst case scenario with a close-by attacker edge

The worst case scenario happens for controllers that are extremely close to some attack edge such that a significant fraction of the m random walks escape into the Sybil region, causing the controller to use many Sybil nodes as ticket sources. To evaluate such worst case scenario, we ran Gatekeeper from different controllers with varying distances to some attack edge and recorded the fraction of the chosen ticket sources that turn out to be Sybil nodes, f_{esc}.

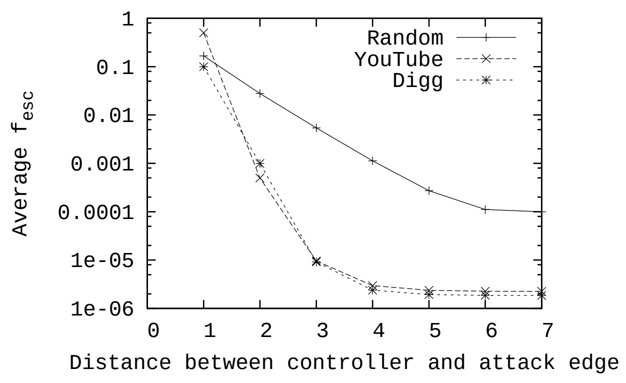


Figure 2.4: The average random walk escape probability, f_{esc}, as a function of the distance between the controller and the closest attack edge

Figure 2.4 shows f_{esc} as a function of the distance between the controller and the closest attack edge under various graph topologies. We can see that f_{esc} drops off quickly to a negligible value as long as the controller is more than 2 hops away from the attacker. The worst case comes when the controller is the immediate neighbor of some attack edges. We first note that if $f_{esc} > f_{admit}$, the controller may accept arbitrarily many Sybil nodes because the $f_{esc} \cdot m$ sources can give infinitely many tickets to Sybils. As we have discussed in Section 2.5.3, our theoretical bound only holds when $f_{esc} < \frac{f_{admit}}{2-f_{admit}}$. Specifically, with a default value of $f_{admit} = 0.2$, f_{esc} must be smaller than 0.11. When the controller is immediately adjacent to some Sybil node, the escape probability is $1/d$ where d is the controller's node degree. Hence, only those controllers with more than 9 neighbors can afford to be-friend the attacker while still satisfying $f_{esc} < 0.11$ and achieving our proven bound.

Chapter 3

Sybil-resilient online content voting

The Web 2.0 revolution has fueled a massive proliferation of user-generated content. While allowing users to publish information has led to democratization of Web content and promoted diversity, it has also made the Web increasingly vulnerable to content pollution from spammers, advertisers and adversarial users misusing the system. Therefore, the ability to rank content accurately is key to the survival and the popularity of many user-content hosting sites. Similarly, content rating is also indispensable in peer-to-peer file sharing systems to help users avoid mislabeled or low quality content [42, 22, 76].

People have long realized the importance of incorporating user opinion in rating online content. Traditional ranking algorithms such as PageRank [6] and HITS [33] rely on implicit user opinions reflected in the link structures of hypertext documents. For arbitrary content types, user opinion can be obtained in the form of explicit votes. Many popular websites today rely on user votes to rank news (Digg, Reddit), videos (YouTube), documents (Scribd) and consumer reviews (Yelp, Amazon).

Content rating based on users' votes is prone to vote manipulation by malicious users. Defending against vote manipulation is difficult due to the *Sybil attack* where the attacker can out-vote real users by creating many Sybil identities. The popularity of content-hosting sites has made such attacks very profitable as malicious entities can promote low-quality content to a wide audience. Successful Sybil attacks have been observed in the wild. For example, online polling on the best computer science school motivated students to deploy automatic scripts to vote for their schools repeatedly [28]. There are even commercial services that help paying clients promote their content to the top spot on popular sites such as YouTube by voting from a large number of Sybil accounts [63].

In this paper, we present SumUp, a Sybil-resilient online content voting system that prevents adversaries from arbitrarily distorting voting results. SumUp leverages the trust relationships that already exist among users (e.g. in the form of social relationships). Since it takes human efforts to establish a trust link, the attacker is unlikely to possess many attack edges (links from honest users to an adversarial identity). Nevertheless, he may create many links among Sybil identities themselves.

SumUp addresses the *vote aggregation problem* which can be stated as follows: *Given m votes on a given object, of which an arbitrary fraction may be from Sybil identities created by an attacker, how do we collect votes in a Sybil resilient manner?* A Sybil-resilient vote aggregation solution should satisfy three properties. First, the solution should collect a significant fraction of votes from honest users. Second, if the attacker has e_A attack edges, the maximum number of bogus votes should be bounded by e_A, independent of the attacker's ability to create many Sybil identities behind him. Third, if the attacker repeatedly casts bogus votes, his ability to vote in the future should be diminished. SumUp achieves all three properties with high probability in the face of Sybil attacks. The key idea in SumUp is the *adaptive vote flow* technique that appropriately assigns and adjusts link capacities in the trust graph to collect the net vote for an object.

Previous works have also exploited the use of trust networks to limit Sybil attacks [9, 37, 87, 84, 83, 47], but none directly addresses the vote aggregation problem. SybilLimit [83] performs admission control so that at most $O(\log n)$ Sybil identities are accepted per attack edge among n honest identities. As SybilLimit results in 10∼30 bogus votes per attack edge in a million-user system [83], SumUp provides notable improvement by limiting bogus votes to one per attack edge. Additionally, SumUp leverages user feedback to further diminish the voting power of adversaries that repeatedly vote maliciously.

In SumUp, each vote collector assigns capacities to links in the trust graph and computes a set of approximate max-flow paths from itself to all voters. Because only votes on paths with non-zero flows are counted, the number of bogus votes collected is limited by the total capacity of attack edges instead of links among Sybil identities. Typically, the number of voters on a given object is much smaller than the total user population (n). Based on this insight, SumUp assigns C_{max} units of capacity in total, thereby limiting the number of votes that can be collected to be C_{max}. SumUp adjusts C_{max} automatically according to the number of honest voters for each object so that it can aggregate a large fraction of votes from honest users. As C_{max} is far less than n, the number of bogus votes collected on a single object (i.e. the attack capacity) is no more than the number of attack edges (e_A). SumUp's security guarantee on bogus votes is probabilistic. If a vote collector happens to be close to an attack edge (a low probability event), the attack capacity could be much higher than e_A. By re-assigning link capacities using feedback, SumUp can restrict the attack capacity to be below e_A even if the vote collector happens to be close to some attack edges.

Using a detailed evaluation of several existing social networks (YouTube, Flickr), we show that SumUp successfully limits the number of bogus votes to the number of attack edges and is also able to collect $> 90\%$ of votes from honest voters. By applying SumUp to the voting trace and social network of Digg (an online news voting site), we have found hundreds of suspicious articles that have been marked “popular” by Digg. Based on manual sampling, we believe that at least 50% of suspicious articles exhibit strong evidence of Sybil attacks.

This chapter is organized as follows. In Section 3.1, we define the system model and the vote aggregation problem. Section 3.2 outlines the overall approach of SumUp and Sections 3.3 and 3.4 present the detailed design. In Section 2.6, we describe our evaluation results.

3.1 The Vote Aggregation Problem

In this section, we outline the system model and formalize the vote aggregation problem that SumUp addresses.

System model: We describe SumUp in a centralized setup where a trusted central authority maintains all the information in the system and performs vote aggregation using SumUp in order to rate content. This centralized mode of operation is suitable for web sites such as Digg, YouTube and Facebook, where all users’ votes and their trust relationships are collected and maintained by a single trusted entity.

SumUp leverages the trust network among users to defend against Sybil attacks [87, 9, 37, 84, 83]. Each trust link is directional. However, the creation of each link requires the consent of both users. Typically, user i creates a trust link to j if i has an offline social relationship to j. Similar to previous work [47, 83], SumUp requires that links are difficult to establish. As a result, an attacker only possesses a small number of attack edges (e_A) from honest users to colluding adversarial identities. Even though e_A is small, the attacker can create many Sybil identities and link them to adversarial entities. We refer to votes from colluding adversaries and their Sybil identities as bogus votes.

SumUp aggregates votes from one or more trusted *vote collectors*. A trusted collector is required in order to break the symmetry between honest nodes and Sybil nodes [9]. SumUp can operate in two modes depending on the choice of trusted vote collectors. In *personalized vote aggregation,* SumUp uses each user as his own vote collector to collect the votes of others. As each user collects a different number of votes on the same object, she also has a different (personalized) ranking of content. In *global vote aggregation,* SumUp uses one or more pre-selected vote collectors to collect votes on behalf of all users. Global vote aggregation has the advantage of allowing for a single global ranking of all objects; however, its performance relies on the proper selection of trusted collectors.

Vote Aggregation Problem: Any identity in the trust network including Sybils can cast a vote on any object to express his opinion on that object. In the simplest case, each vote is either positive or negative (+1 or -1). Alternatively, to make a vote more expressive, its value can vary within a range with higher values indicating more favorable opinions. A vote aggregation system collects votes on a given object. Based on collected votes and various other features, a separate ranking system determines the final ranking of an object. The design of the final ranking system is outside the scope of this paper. However, we note that many ranking algorithms utilize *both* the number of votes and the average value of votes to determine an object's rank [6, 33]. Therefore, to enable arbitrary ranking algorithms, a vote aggregation system should collect a significant fraction of votes from honest voters.

A voting system can also let the vote collector provide *negative* feedback on malicious votes. In personalized vote aggregation, each collector gives feedback according to his personal taste. In global vote aggregation, the vote collector(s) should only provide objective feedback, e.g. negative feedback for positive votes on corrupted files. Such feedback is available for a very small subset of objects.

We describe the desired properties of a vote aggregation system. Let $G = (V, E)$ be a trust network with vote collector $s \in V$. V is comprised of an unknown set of honest users $V_h \subset V$ (including s) and the attacker controls all vertices in $V \setminus V_h$, many of which represent Sybil identities. Let e_A represent the number of attack edges from honest users in V_h to $V \setminus V_h$. Given that nodes in G cast votes on a specific object, a vote aggregation mechanism should achieve three properties:

- Collect a large fraction of votes from honest users.
- Limit the number of bogus votes from the attacker by e_A independent of the number of Sybil identities in $V \setminus V_h$.
- Eventually ignore votes from nodes that repeatedly cast bogus votes using feedback.

3.2 Basic Approach

This section describes the intuition behind *adaptive vote flow* that SumUp uses to address the vote aggregation problem. The key idea of this approach is to appropriately assign link capacities to bound the attack capacity.

In order to limit the number of votes that Sybil identities can propagate for an object, SumUp computes a set of max-flow paths in the trust graph from the vote collector to all voters on a given object. Each vote flow consumes one unit of capacity along each

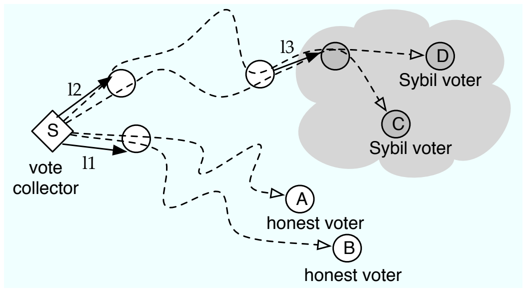


Figure 3.1: SumUp computes a set of approximate max-flow paths from the vote collector s to all voters (A,B,C,D). Straight lines denote trust links and curly dotted lines represent the vote flow paths along multiple links. Vote flow paths to honest voters are “congested” at links close to the collector while paths to Sybil voters are also congested at far-away attack edges.

link traversed. Figure 3.1 gives an example of the resulting flows from the collector s to voters A,B,C,D. When all links are assigned unit capacity, the attack capacity using the max-flow based approach is bounded by e_A.

The concept of max-flow has been applied in several reputation systems based on trust networks [9, 37]. When applied in the context of vote aggregation, the challenge is that links close to the vote collector tend to become “congested” (as shown in Figure 3.1), thereby limiting the total number of votes collected to be no more than the collector’s node degree. Since practical trust networks are sparse with small median node degrees, only a few honest votes can be collected. We cannot simply enhance the capacity of each link to increase the number of votes collected since doing so also increases the attack capacity. Hence, a flow-based vote aggregation system faces the tradeoff between the maximum number of honest votes it can collect and the number of potentially bogus votes collected.

The *adaptive vote flow* technique addresses this tradeoff by exploiting two basic observations. First, the number of honest users voting for an object, even a popular one, is significantly smaller than the total number of users. For example, 99% of popular arti-

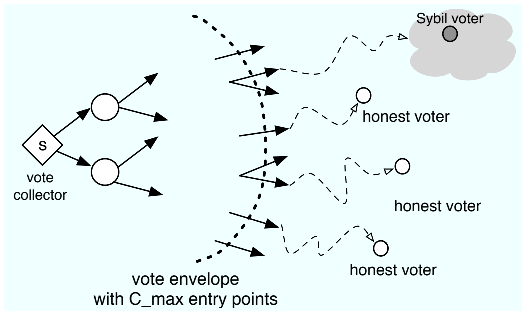


Figure 3.2: Through ticket distribution, SumUp creates a vote envelope around the collector. The capacities of links beyond the envelope are assigned to be one, limiting the attack capacity to be at most one per attack edge for adversaries outside this envelope. There is enough capacity within the envelope, such that nodes inside act like entry points for outside voters.

cles on Digg have fewer than 4000 votes which represents 1% of active users. Second, vote flow paths to honest voters tend to be only "congested" at links close to the vote collector while paths to Sybil voters are also congested at a few attack edges. When e_A is small, attack edges tend to be far away from the vote collector. As shown in Figure 3.1, vote flow paths to honest voters A and B are congested at the link l_1 while paths to Sybil identities C and D are congested at both l_2 and attack edge l_3.

The adaptive vote flow computation uses three key ideas. First, the algorithm restricts the maximum number of votes collected on an object to a value C_{max}. As C_{max} is used to assign the overall capacity in the trust graph, a small C_{max} results in less capacity for the attacker. SumUp can adaptively adjust C_{max} to collect a large fraction of honest votes on any given object. When the number of honest voters is $O(n^\alpha)$ where $\alpha < 1$, the expected number of bogus votes is limited to $1 + o(1)$ per attack edge (Section 3.3.4).

The second important aspect of SumUp relates to *capacity assignment*, i.e. how to assign capacities to each trust link in order to collect a large fraction of honest votes and only a few bogus ones? In SumUp, the vote collector distributes C_{max} *tickets* downstream in a breadth-first search manner within the trust network. The capacity assigned to a link is the number of tickets distributed along the link plus one. As Figure 3.2 illustrates, the ticket distribution process introduces a *vote envelope* around the vote collector s; beyond the envelope all links have capacity 1. The vote envelope contains C_{max} nodes that can be viewed as entry points. There is enough capacity within the envelope to collect C_{max} votes from entry points. On the other hand, an attack edge beyond the envelope can propagate at most 1 vote regardless of the number of Sybil identities behind that edge. SumUp re-distributes tickets based on feedback to deal with attack edges within the envelope.

The final key idea in SumUp is to leverage user feedback to penalize attack edges that continuously propagate bogus votes. One cannot penalize individual identities since the attacker may always propagate bogus votes using new Sybil identities. Since an attack edge is always present in the path from the vote collector to a malicious voter [47], SumUp re-adjusts capacity assignment across links to reduce the capacity of penalized attack edges.

3.3 SumUp Design

In this section, we present the basic capacity assignment algorithm that achieves two of the three desired properties discussed in Section 3.1: (a) Collect a large fraction of votes from honest users; (b) Restrict the number of bogus votes to one per attack edge with high probability. Later in Section 3.4, we show how to adjust capacity based on feedback to deal with repeatedly misbehaved adversarial nodes.

We describe how link capacities are assigned given a particular C_{max} in Section 3.3.1 and present a fast algorithm to calculate approximate max-flow paths in Section 3.3.2. In Section 3.3.3, we introduce an additional optimization strategy that prunes links in the trust network so as to reduce the number of attack edges. We formally analyze the security properties of SumUp in Section 3.3.4 and show how to adaptively set C_{max} in Section 3.3.5.

3.3.1 Capacity assignment

The goal of capacity assignment is twofold. On the one hand, the assignment should allow the vote collector to gather a large fraction of honest votes. On the other hand, the assignment should minimize the attack capacity such that $C_A \approx e_A$.

As Figure 3.2 illustrates, the basic idea of capacity assignment is to construct a vote envelope around the vote collector with at least C_{max} entry points. The goal is to minimize the chances of including an attack edge in the envelope and to ensure that there is enough capacity within the envelope so that all vote flows from C_{max} entry points can reach the collector.

We achieve this goal using a *ticket distribution* mechanism which results in decreasing capacities for links with increasing distance from the vote collector. The distribution mechanism is best described using a propagation model where the vote collector is to spread C_{max} tickets across all links in the trust graph. Each ticket corresponds to a capacity value of 1. We associate each node with a level according to its shortest path distance from the vote collector, s. Node s is at level 0. Tickets are distributed to nodes one level at a time. If a node at level l has received t_{in} tickets from nodes at level $l-1$, the node consumes one ticket and re-distributes the remaining tickets evenly across all its outgoing links to nodes at level $l+1$, i.e. $t_{out} = t_{in} - 1$. The capacity value of each link is set to be one plus the number of tickets distributed on that link. Tickets are not distributed to links connecting nodes at the same level or from a higher to lower level. The set of nodes with positive incoming tickets fall within the vote envelope and thus

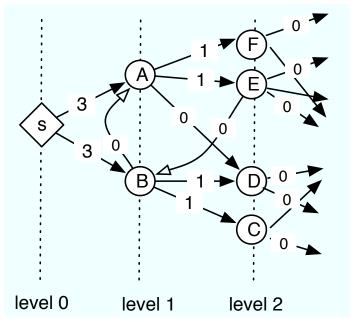


Figure 3.3: Each link shows the number of tickets distributed to that link from s (C_{max}=6). A node consumes one ticket and distributes the remaining evenly via its outgoing links to the next level. Tickets are not distributed to links pointing to the same level (B$\rightarrow$A), or to a lower level (E$\rightarrow$B). The capacity of each link is equal to one plus the number of tickets.

represent the entry points.

Ticket distribution ensures that all C_{max} entry points have positive vote flows to the vote collector. Therefore, if there exists an edge-independent path connecting one of the entry points to an outside voter, the corresponding vote can be collected. We show in Section 3.3.4 that such a path exists with good probability. When C_{max} is much smaller than the number of honest nodes (n), the vote envelope is very small. Therefore, all attack edges reside outside the envelope, resulting in $C_A \approx e_A$ with high probability.

Figure 3.3 illustrates an example of the ticket distribution process. The vote collector (s) is to distribute C_{max}=6 tickets among all links. Each node collects tickets from its lower level neighbors, keeps one to itself and re-distributes the rest evenly across all outgoing links to the next level. In Figure 3.3, s sends 3 tickets down each of its outgoing links. Since A has more outgoing links (3) than its remaining tickets (2), link A$\rightarrow$D receives no tickets. Tickets are not distributed to links between nodes at the same level (B$\rightarrow$A) or to links from a higher to lower level (E$\rightarrow$B). The final number of tickets distributed on each link is shown in Figure 3.3. Except for immediate outgoing edges from the vote collector, the capacity value of each link is equal to the amount of tickets it receives plus one.

3.3.2 Approximate Max-flow calculation

Once capacity assignment is done, the task remains to calculate the set of max-flow paths from the vote collector to all voters on a given object. It is possible to use existing max-flow algorithms such as Ford-Fulkerson and Preflow push [14] to compute vote flows. Unfortunately, these existing algorithms require $O(E)$ running time to find each vote flow, where E is the number of edges in the graph. Since vote aggregation only aims to collect a large fraction of honest votes, it is not necessary to compute exact max-flow paths. In particular, we can exploit the structure of capacity assignment to compute a set of approximate vote flows in $O(\Delta)$ time, where Δ is the diameter of the graph. For expander-like networks, $\Delta = O(\log n)$. For practical social networks with a few million users, $\Delta \approx 20$.

Our approximation algorithm works incrementally by finding one vote flow for a voter at a time. Unlike the classic Ford-Fulkerson algorithm, our approximation performs a greedy search from the voter to the collector in $O(\Delta)$ time instead of a breadth-first-search from the collector which takes $O(E)$ running time. Starting at a voter, the greedy search strategy attempts to explore a node at a lower level if there exists an incoming link with positive capacity. Since it is not always possible to find such a candidate for exploration, the approximation algorithm allows a threshold (t) of non-greedy steps which explores nodes at the same or a higher level. Therefore, the number of nodes visited by the greedy search is bounded by $(\Delta + 2t)$. Greedy search works well in practice. For links within the vote envelope, there is more capacity for lower-level links and hence greedy search is more likely to find a non-zero capacity path by exploring lower-level nodes. For links outside the vote envelope, greedy search results in short paths to one of the vote entry points.

3.3.3 Optimization via link pruning

We introduce an optimization strategy that performs link pruning to reduce the number of attack edges, thereby reducing the attack capacity. Pruning is performed prior to link capacity assignment and its goal is to bound the in-degree of each node to a small value, d_{in_thres}. As a result, the number of attack edges is reduced if some adversarial nodes have more than d_{in_thres} incoming edges from honest nodes. We speculate that the more honest neighbors an adversarial node has, the easier for it to trick an honest node into trusting it. Therefore, the number of attack edges in the pruned network is likely to be smaller than those in the original network. On the other hand, pruning is unlikely to affect honest users since each honest node only attempts to cast one vote via one of its incoming links.

Since it is not possible to accurately discern honest identities from Sybil identities, we give all identities the chance to have their votes collected. In other words, pruning should never disconnect a node. The minimally connected network that satisfies this requirement is a tree rooted at the vote collector. A tree topology minimizes attack edges but is also overly restrictive for honest nodes because each node has exactly one path from the collector: if that path is saturated, a vote cannot be collected. A better tradeoff is to allow each node to have at most $d_{in\_thres} > 1$ incoming links in the pruned network so that honest nodes have a large set of diverse paths while limiting each adversarial node to only d_{in_thres} attack edges. We examine the specific parameter choice of d_{in_thres} in Section 3.5.

Pruning each node to have at most d_{in_thres} incoming links is done in several steps. First, we remove all links except those connecting nodes at a lower level (l) to neighbors at the next level ($l + 1$). Next, we remove a subset of incoming links at each node so that the remaining links do not exceed d_{in_thres}. In the third step, we add back links removed in step one for nodes with fewer than d_{in_thres} incoming links. Finally, we add one outgoing link back to nodes that have no outgoing links after step three, with priority given to links going to the next level. By preferentially preserving links from lower to higher levels, pruning does not interfere with SumUp's capacity assignment and flow computation.

3.3.4 Security Properties

This section provides a formal analysis of the security properties of SumUp assuming an expander graph. Various measurement studies have shown that social networks are indeed expander-like [34]. The link pruning optimization does not destroy a graph's expander property because it preserves the level of each node in the original graph.

Our analysis provides bounds on the expected attack capacity, C_A, and the expected fraction of votes collected if C_{max} honest users vote. The average-case analysis assumes that each attack edge is a random link in the graph. For personalized vote aggregation, the expectation is taken over all vote collectors which include all honest nodes. In the unfortunate but rare scenario where an adversarial node is close to the vote collector, we can use feedback to re-adjust link capacities (Section 3.4).

Theorem 3.1. *Given that the trust network G on n nodes is a bounded degree expander graph, the expected capacity per attack edge is $\frac{E(C_A)}{e_A} = 1 + O(\frac{C_{max}}{n} \log C_{max})$ which is $1 + o(1)$ if $C_{max} = O(n^{\alpha})$ for $\alpha < 1$. If $e_A \cdot C_{max} \ll n$, the capacity per attack edge is bounded by 1 with high probability.*

Proof. Let L_i represent the number of nodes at level i with $L_0 = 1$. Let E_i be the number of edges pointing from level $i-1$ to level i. Notice that $E_i \geq L_i$. Let T_i be the number of tickets propagated from level $i-1$ to i with $T_0 = C_{max}$. The number of tickets at each level is reduced by the number of nodes at the previous level (i.e. $T_i = T_{i-1} - L_{i-1}$). Therefore, the number of levels with non-zero tickets is at most $O(log(C_{max}))$ as L_i grows exponentially in an expander graph. For a randomly placed attack edge, the probability of its being at level i is at most L_i/n. Therefore, the expected capacity of a random attack edge can be calculated as $1 + \sum_i (\frac{L_i}{n} \cdot \frac{T_i}{E_i}) < 1 + \sum_i (\frac{L_i}{n} \cdot \frac{C_{max}}{L_i}) = 1 + O(\frac{C_{max}}{n} \log C_{max})$. Therefore, if $C_{max} = O(n^{\alpha})$ for $\alpha < 1$, the expected attack capacity per attack edge is $1 + o(1)$.

Since the number of nodes within the vote envelope is at most C_{max}, the probability of a random attack edge being located outside the envelope is $1 - \frac{C_{max}}{n}$. Therefore, the probability that any of the e_A attack edges lies within the vote envelope is $1 - (1 - \frac{C_{max}}{n})^{e_A} < \frac{e_A \cdot C_{max}}{n}$. Hence, if $e_A \cdot C_{max} = n^{\alpha}$ where $\alpha < 1$, the attack capacity is bounded by 1 with high probability.

□

Theorem 3.1 is for expected capacity per attack edge. In the worse case when the vote collector is adjacent to some adversarial nodes, the attack capacity can be a significant fraction of C_{max}. Such rare worst case scenarios are addressed in Section 3.4.

Theorem 3.2. *Given that the trust network G on n nodes is a d-regular expander graph, the expected number of votes that can be collected out of n_v honest voters is $F(C_{max}, n_v) = \frac{d-\lambda_2}{d} \frac{m(n-m)}{n}$ where λ_2 is the second largest eigenvalue of the adjacency matrix of G, and $m = min\{C_{max}, n_v\}$.*

Proof. SumUp creates a vote envelop consisting of C_{max} entry points via which votes are collected. To prove that there exists a large fraction of vote flows, we argue that the minimum cut of the graph between the set of C_{max} entry points and an arbitrary set of C_{max} honest voters is large.

Expanders are well-connected graphs. In particular, the Expander mixing lemma [50] states that for any set S and T in a d-regular expander graph, the expected number of edges between S and T is $(d - \lambda_2)|S| \cdot |T|/n$, where λ_2 is the second largest eigenvalue of the adjacency matrix of G. Let S be a set of nodes containing C_{max} entry points and T be a set of nodes containing n_v honest voters, thus $|S| + |T| = n$ and $|S| \geq m, |T| \geq m$. Therefore, the min-cut value between S and T is $= (d - \lambda_2)|S| \cdot |T|/n \geq (d - \lambda_2) \cdot m(n - m)/n$. The number of vote flows between S and T is at least $1/d$ of the min-cut value because each vote flow only uses one of an honest voter's d incoming links. Therefore, the votes that can be collected is at least $\frac{d-\lambda_2}{d}\frac{m(n-m)}{n}$. When $C_{max} \geq n_v$, i.e. $m = n_v$, the expected fraction of votes that can be collected out of n_v honest voters is $\frac{d-\lambda_2}{d}(1 - \frac{n_v}{n})$.For well-connected graphs like expanders, λ_2 is well separated from d, so that a significant fraction of votes can be collected.

$\square$

3.3.5 Setting C_{max} adaptively

When n_v honest users vote on an object, SumUp should ideally set C_{max} to be n_v in order to collect a large fraction of honest votes on that object. In practice, n_v/n is very small for any object, even a very popular one. Hence, $C_{max} = n_v \ll n$ and the expected capacity per attack edge is 1. We note that even if $n_v \approx n$, the attack capacity is still bounded by $O(\log n)$ per attack edge.

It is impossible to precisely calculate the number of honest votes (n_v). However, we can use the actual number of votes collected by SumUp as a lower bound estimate for n_v. Based on this intuition, SumUp adaptively sets C_{max} according to the number of votes collected for each object. The adaptation works as follows: For a given object, SumUp starts with a small initial value for C_{max}, e.g. $C_{max} = 100$. Subsequently, if the number of actual votes collected exceeds ρC_{max} where ρ is a constant less than 1, SumUp doubles the C_{max} in use and re-runs the capacity assignment and vote collection procedures. The doubling of C_{max} continues until the number of collected votes becomes less than ρC_{max}.

We show that this adaptive strategy is robust: 1) the maximum value of the resulting C_{max} will not dramatically exceed n_v regardless of the number of bogus votes cast by adversarial nodes , 2) resulting C_{max} is big enough to collect big fraction of votes, i.e. C_{max} won't stop early.

Resulting C_{max} is not too big: Since adversarial nodes attempt to cast enough bogus votes to saturate attack capacity, the number of votes collected is at most $n_v + C_A$ where $C_A = e_A(1 + \frac{C_{max}}{n} \log C_{max})$. The doubling of C_{max} stops when the number of collected votes is less than ρC_{max}. Therefore, the maximum value of C_{max} that stops the adaptation is one that satisfies the following inequality:

$$n_v + e_A(1 + \frac{C_{max}}{n} \log C_{max}) < \rho C_{max}$$

Since $\log C_{max} \leq \log n$, the adaptation terminates with $C'_{max} = (n_v + e_A)/(\rho - \frac{\log n}{n})$. As $\rho \gg \frac{\log n}{n}$, we derive $C'_{max} = \frac{1}{\rho}(n_v + e_A)$. The adaptive strategy doubles C_{max} every iteration, hence it overshoots by at most a factor of two. Therefore, the resulting C_{max} found is $C_{max} = \frac{2}{\rho}(n_v + e_A)$. As we can see, the attacker can only affect the C_{max} found by an additive factor of e_A. Since e_A is small, the attacker has negligible influence on the C_{max} found.

The previous analysis is done for the expected case with random attack edges. Even in a worst case scenario where some attack edges are very close to the vote collector, the adaptive strategy is still resilient against manipulation. In the worst case scenario, the attack capacity is proportional to C_{max}, i.e. $C_A = xC_{max}$. Since no vote aggregation scheme can defend against an attacker who controls a majority of immediate links from the vote collector, we are only interested in the case where $x < 0.5$. The adaptive strategy stops increasing C_{max} when $n_v + xC_{max} < \rho C_{max}$, thus resulting in $C_{max} \leq \frac{2n_v}{\rho - x}$. As we can see, ρ must be greater than x to prevent the attacker from causing SumUp to increase C_{max} to infinity. Therefore, we set $\rho = 0.5$ by default.

Resulting C_{max} is not too small: Suppose C_0 is a constant satisfying $\rho C_0 = F(C_0, n_v)$. We will show that the resulting C_{max} is at least C_0. Because $\frac{F(x,n_v)}{x}$ is a decreasing function, when $C_{max} < C_0$, we have:

$$\begin{aligned} \frac{F(C_0, n_v)}{C_0} &< \frac{F(C_{max}, n_v)}{C_{max}} \\ \rho &< \frac{F(C_{max}, n_v)}{C_{max}} \\ \rho C_{max} &< F(C_{max}, n_v) \end{aligned}$$

Because the number of votes collected is at least $F(C_{max}, n_v)$ by Theorem 3.2, the condition for doubling C_{max} holds when $C_{max} < C_0$. When we pick ρ small enough such that $\rho < \frac{d-\lambda_2}{d}\frac{n-n_v}{n}$, $C_0 = \frac{d-\lambda_2}{d}\frac{n-n_v}{n}\frac{n_v}{\rho} > n_v$. Hence, the resulting C_{max} is bigger than number of honest voters.

3.4 Leveraging user feedback

The basic design presented in Section 3.3 does not address the worst case scenario where C_A could be much higher than e_A. Furthermore, the basic design only bounds the number of bogus votes collected on a single object. As a result, adversaries can still cast up to e_A bogus votes on *every* object in the system. In this section, we utilize feedback to address both problems.

SumUp maintains a penalty value for each link and uses the penalty in two ways. First, we adjust each link's capacity assignment so that links with higher penalties have lower capacities. This helps reduce C_A when some attack edges happen to be close to the vote collector. Second, we eliminate links whose penalties have exceeded a certain threshold. Therefore, if adversaries continuously misbehave, the attack capacity will drop below e_A over time. We describe how SumUp calculates and uses penalty in the rest of the section.

3.4.1 Incorporating negative feedback

The vote collector can choose to associate negative feedback with voters if he believes their votes are malicious. Feedback may be performed for a very small set of objects-for example, when the collector finds out that an object is a bogus file or a virus.

SumUp keeps track of a penalty value, p_i, for each link i in the trust network. For each voter receiving negative feedback, SumUp increments the penalty values for all links along the path to that voter. Specifically, if the link being penalized has capacity c_i, SumUp increments the link's penalty by $1/c_i$. Scaling the increment by c_i is intuitive; links with high capacities are close to the vote collector and hence are more likely to propagate some bogus votes even if they are honest links. Therefore, SumUp imposes a lesser penalty on high capacity links.

It is necessary to penalize *all* links along the path instead of just the immediate link to the voter because that voter might be a Sybil identity created by some other attacker along the path. Punishing a link to a Sybil identity is useless as adversaries can easily create more such links. This way of incorporating negative feedback is inspired by Ostra [47]. Unlike Ostra, SumUp uses a customized flow network per vote collector and only allows the collector to incorporate feedback for its associated network in order to ensure that feedback is always trustworthy.

3.4.2 Capacity adjustment

The capacity assignment in Section 3.3.1 lets each node distribute incoming tickets evenly across all outgoing links. In the absence of feedback, it is reasonable to assume that all outgoing links are equally trustworthy and hence to assign them the same number of tickets. When negative feedback is available, a node should distribute fewer tickets to outgoing links with higher penalty values. Such adjustment is particularly useful in circumstances where adversaries are close to the vote collector and hence might receive a large number of tickets.

The goal of capacity adjustment is to compute a weight, $w(p_i)$, as a function of the link's penalty. The number of tickets a node distributes to its outgoing link i is proportional to the link's weight, i.e. $t_i = t_{out} * w(p_i) / \sum_{\forall i \in nbrs} w(p_i)$. The question then becomes how to compute $w(p_i)$. Clearly, a link with a high penalty value should have a smaller weight, i.e. $w(p_i) < w(p_j)$ if $p_i > p_i$. Another desirable property is that if the penalties on two links increase by the same amount, the ratio of their weights remains unchanged. In other words, the weight function should satisfy: $\forall p', p_i, p_j, \quad \frac{w(p_i)}{w(p_j)} = \frac{w(p_i+p')}{w(p_j+p')}$. This requirement matches our intuition that if two links have accumulated the same amount of additional penalties over a period of time, the relative capacities between them should remain the same. Since the exponential function satisfies both requirements, we use $w(p_i) = 0.2^{p_i}$ by default.

3.4.3 Eliminating links using feedback

Capacity adjustment cannot reduce the attack capacity to below e_A since each link is assigned a minimum capacity value of one. To further reduce e_A, we eliminate those links that received high amounts of negative feedback.

We use a heuristic for link elimination: we remove a link if its penalty exceeds a threshold value. We use a default threshold of five. Since we already prune the trust network (Section 3.3.3) before performing capacity assignment, we add back a previously pruned link if one exists after eliminating an incoming link. The reason why link elimination is useful can be explained intuitively: if adversaries continuously cast bogus votes on different objects over time, all attack edges will be eliminated eventually. On the other hand, although an honest user might have one of its incoming links eliminated because of a downstream attacker casting bad votes, he is unlikely to experience another elimination due to the same attacker since the attack edge connecting him to that attacker has also been eliminated. Despite this intuitive argument, there always exist pathological scenarios where link elimination affects some honest users, leaving them with no voting power. To address such potential drawbacks, we re-enact eliminated links at a slow rate over time. We evaluate the effect of link elimination in Section 2.6.

Network	**Nodes** $\times 1000$	**Edges** $\times 1000$	**Degree 50%(90%)**	**Directed?**
YouTube [47]	446	3,458	2 (12)	No
Flickr [45]	1,530	21,399	1 (15)	Yes
Synthetic [68]	3000	24,248	6 (15)	No

Table 3.1: Statistics of the social network traces or synthetic model used for evaluating SumUp. All statistics are for the strongly connected component (SCC).

3.5 Evaluation

In this section, we demonstrate SumUp's security property using real-world social networks and voting traces. Our key results are:

- For all networks under evaluation, SumUp bounds the average number of bogus votes collected to be no more than e_A while being able to collect >90% of honest votes when less than 1% of honest users vote.

- By incorporating feedback from the vote collector, SumUp dramatically cuts down the attack capacity for adversaries that continuously cast bogus votes.

- We apply SumUp to the voting trace and social network of Digg [1], a news aggregation site that uses votes to rank user-submitted news articles. SumUp has detected hundreds of suspicious articles that have been marked as "popular" by Digg. Based on manual sampling, we believe at least 50% of suspicious articles found by SumUp exhibit strong evidence of Sybil attacks.

3.5.1 Experimental Setup

For the evaluation, we use a number of network datasets from different online social networking sites [45] as well as a synthetic social network [68] as the underlying trust network. SumUp works for different types of trust networks as long as an attacker cannot obtain many attack edges easily in those networks. Table 3.1 gives the statistics of various datasets. For undirected networks, we treat each link as a pair of directed links. Unless explicitly mentioned, we use the YouTube network by default.

To evaluate the Sybil-resilience of SumUp, we inject $e_A = 100$ attack edges by adding 10 adversarial nodes each with links from 10 random honest nodes in the network. The attacker always casts the maximum bogus votes to saturate his capacity. Each experimental run involves a randomly chosen vote collector and a subset of nodes which serve as honest voters. SumUp adaptively adjusts C_{max} using an initial value of 100 and $\rho = 0.5$. By default, the threshold of allowed non-greedy steps is 20. We plot the average statistic across five experimental runs in all graphs. In Section 3.5.6, we apply SumUp on the real world voting trace of Digg to examine how SumUp can be used to resist Sybil attacks in the wild.

3.5.2 Sybil-resilience of the basic design

The main goal of SumUp is to limit attack capacity while allowing honest users to vote. Figure 3.4 shows that the average attack capacity per attack edge remains close to 1 even when the number of honest voters approaches 10%. Furthermore, as shown in Figure 3.5, SumUp manages to collect more than 90% of all honest votes in all networks. Link pruning is disabled in these experiments. The three networks under evaluation have very different sizes and degree distributions (see Table 3.1). The fact that all three networks exhibit similar performance suggests that SumUp is robust against the topological details. Since SumUp adaptively sets C_{max} in these experiments, the results also confirm that adaptation works well in finding a C_{max} that can collect most of the honest votes without significantly increasing attack capacity. We point out that the re-

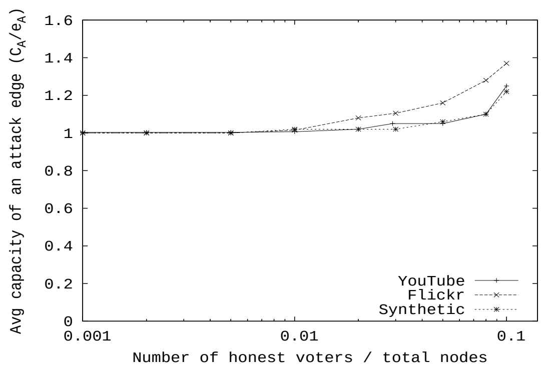


Figure 3.4: The average capacity per attack edge as a function of the fraction of honest nodes that vote. The average capacity per attack edge remains close to 1, even if $1/10$ of honest nodes vote.

sults in Figure 3.4 correspond to a random vote collector. For an unlucky vote collector close to an attack edge, he may experience a much larger than average attack capacity. In personalized vote collection, there are few unlucky collectors. These unlucky vote collectors need to use their own feedback on bogus votes to reduce attack capacity.

Benefits of pruning: The link pruning optimization, introduced in Section 3.3.3, further reduces the attack capacity by capping the number of attack edges an adversarial node can have. As Figure 3.6 shows, pruning does not affect the fraction of honest votes collected if the threshold d_{in_thres} is greater than 3. Figure 3.6 represents data from the YouTube network and the results for other networks are similar. SumUp uses the default threshold (d_{in_thres}) of 3. Figure 3.7 shows that the average attack capacity is greatly reduced when adversarial nodes have more than 3 attack edges. Since pruning attempts to restrict each node to at most 3 incoming links, additional attack edges are excluded from vote flow computation.

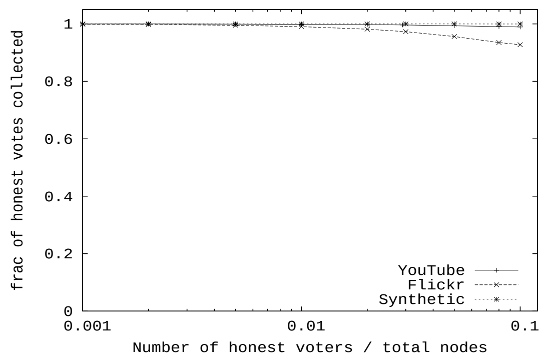


Figure 3.5: The fraction of votes collected as a function of fraction of honest nodes that vote. SumUp collects more than 80% votes, even $1/10$ honest nodes vote.

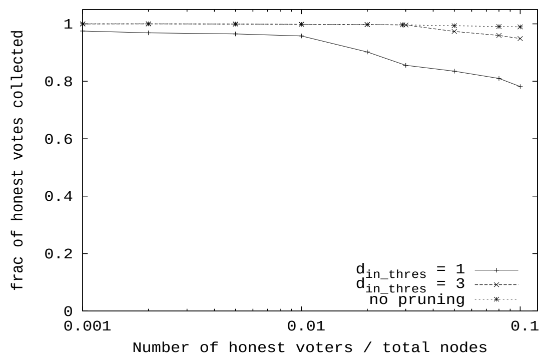


Figure 3.6: The fraction of votes collected for different d_{in_thres} (YouTube graph). More than 90% votes are collected when $d_{in\_thres} = 3$.

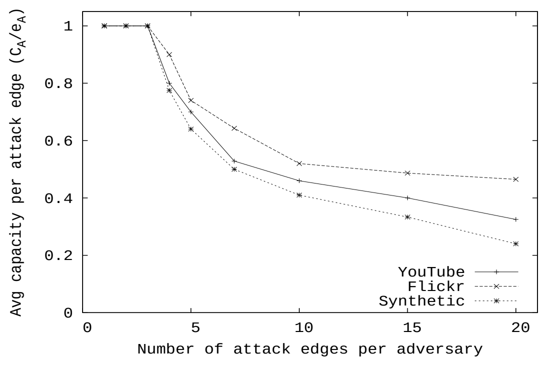


Figure 3.7: Average attack capacity per attack edge decreases as the number of attack edges per adversary increases.

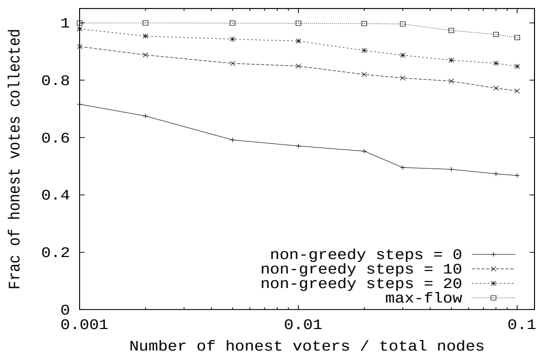


Figure 3.8: The fraction of votes collected for different threshold for non-greedy steps. More than 70% votes are collected even with a small threshold (10) for non-greedy steps.

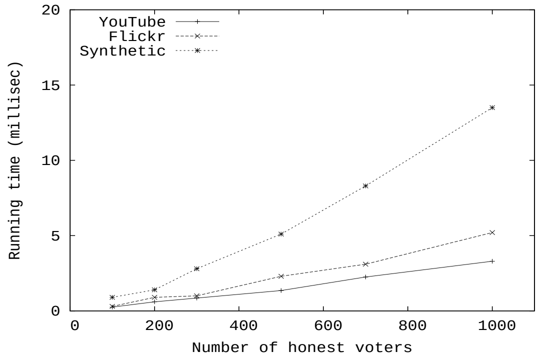


Figure 3.9: The running time of one vote collector gathering up to 1000 votes. The Ford-Fulkerson max-flow algorithm takes 50 seconds to collect 1000 votes for the YouTube graph.

3.5.3 Effectiveness of greedy search

SumUp uses a fast greedy algorithm to calculate approximate max vote flows to voters. Greedy search enables SumUp to collect a majority of votes while using a small threshold (t) of non-greedy steps. Figure 3.8 shows the fraction of honest votes collected for the pruned YouTube graph. As we can see, with a small threshold of 20, the fraction of votes collected is more than 80%. Even when disallowing non-greedy steps completely, SumUp manages to collect $> 40\%$ of votes.

Figure 3.9 shows the running time of greedy-search for different networks. The experiments are performed on a single machine with an AMD Opteron 2.5GHz CPU and 8GB memory. SumUp takes around 5ms to collect 1000 votes from a single vote collector on YouTube and Flickr. The synthetic network incurs more running time as its links are more congested than those in YouTube and Flickr. The average non-greedy steps taken in the synthetic network is 6.5 as opposed to 0.8 for the YouTube graph. Greedy-search dramatically reduces the flow computation time. As a comparison, the Ford-Fulkerson max-flow algorithm requires 50 seconds to collect 1000 votes for the YouTube graph.

3.5.4 Comparison with SybilLimit

SybilLimit is a node admission protocol that leverages the trust network to allow an honest node to accept other honest nodes with high probability. It bounds the number of Sybil nodes accepted to be $O(\log n)$. We can apply SybilLimit for vote aggregation by letting each vote collector compute a fixed set of accepted users based on the trust network. Subsequently, a vote is collected if and only if it comes from one of the accepted users. In contrast, SumUp does not calculate a fixed set of allowed users; rather,

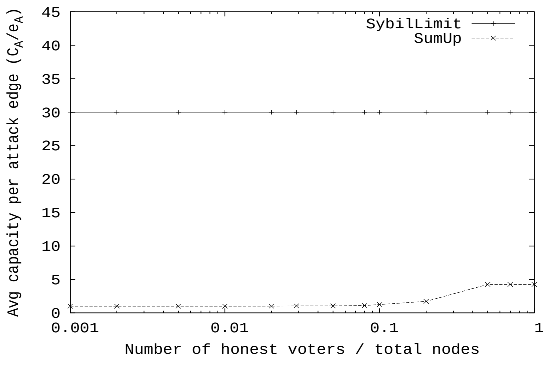


Figure 3.10: Average attack capacity per attack edge as a function of voters. SumUp is better than SybilLimit in the average case.

it dynamically determines the set of voters that count toward each object. Such dynamic calculation allows SumUp to settle on a small C_{max} while still collecting most of the honest votes. A small C_{max} allows SumUp to bound attack capacity by e_A.

Figure 3.10 compares the average attack capacity in SumUp to that of SybilLimit for the un-pruned YouTube network. The attack capacity in SybilLimit refers to the number of Sybil nodes that are accepted by the vote collector. Since SybilLimit aims to accept nodes instead of votes, its attack capacity remains $O(\log n)$ regardless of the number of actual honest voters. Our implementation of SybilLimit uses the optimal set of parameters ($w = 15$, $r = 3000$) we determined manually. As Figure 3.10 shows, while SybilLimit allows 30 bogus votes per attack edge, SumUp results in approximately 1 vote per attack edge when the fraction of honest voters is less than 10%. When all nodes vote, SumUp leads to much lower attack capacity than SybilLimit even though both have the same $O(\log n)$ asymptotic bound per attack edge. This is due to two reasons. First, SumUp's bound of $1 + \log n$ in Theorem 3.1 is a loose upper bound of the actual average capacity. Second, since links pointing to lower-level nodes are not eligible for ticket distribution, many incoming links of an adversarial nodes have zero

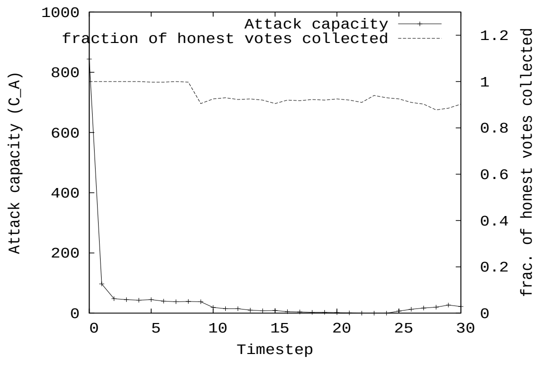


Figure 3.11: The change in attack capacity as adversaries continuously cast bogus votes (YouTube graph). Capacity adjustment and link elimination dramatically reduce C_A while still allowing SumUp to collect more than 80% of the honest votes.

tickets and thus are assigned capacity of one.

3.5.5 Benefits of incorporating feedback

We evaluate the benefits of capacity adjustment and link elimination when the vote collector provides feedback on the bogus votes collected. Figure 3.11 corresponds to the worst case scenario where one of the vote collector's four outgoing links is an attack edge. At every time step, there are 400 random honest users voting on an object and the attacker also votes with its maximum capacity. When collecting votes on the first object at time step 1, adaption results in $C_{max} = \frac{2n_v}{\rho - x} = 3200$ because $n_v = 400, \rho = 0.5, x = 1/4$. Therefore, the attacker manages to cast $\frac{1}{4}C_{max} = 800$ votes and outvote honest users. After incorporating the vote collector's feedback after the first time step, the adjacent attack edge incurs a penalty of 1 which results in drastically reduced C_A (97). If the vote collector continues to provide feedback on malicious votes, 90% of attack edges are eliminated after only 12 time steps. After another 10 time steps, all attack edges are eliminated, reducing C_A to zero. However, because of our decision to slowly add back eliminated links, the attack capacity doesn't remains at zero forever.

Number of Nodes	3,002,907
Number of Edges	5,063,244
Number of Nodes in SCC	466,326
Number of Edges in SCC	4,908,958
Out degree avg(50%, 90%)	10(1, 9)
In degree avg(50%, 90%)	10(2, 11)
Number of submitted (popular) articles 2004/12/01-2008/09/21	6,494,987 (137,480)
Diggs on all articles avg(50%, 90%)	24(2, 15)
Diggs on popular articles avg(50%, 90%)	862(650, 1810)
Hours since submission before a popular article is marked as popular. avg (50,%,90%)	16(13, 23)
Number of submitted (popular) articles with *bury* data available 2008/08/13-2008/09/15	38,033 (5,794)

Table 3.2: Basic statistics of the crawled Digg dataset. The strongly connected component (SCC) of Digg consists of 466,326 nodes.

Figure 3.11 also shows that link elimination has little effects on honest nodes as the fraction of honest votes collected always remains above 80%.

3.5.6 Defending Digg against Sybil attacks

In this section, we ask the following questions: Is there evidence of Sybil attacks in real world content voting systems? Can SumUp successfully limit bogus votes from Sybil identities? We apply SumUp to the voting trace and social network crawled from Digg to show the real world benefits of SumUp.

Digg [1] is a popular news aggregation site where any registered user can submit an article for others to vote on. A positive vote on an article is called a *digg*. A negative vote is called a *bury*. Digg marks a subset of submitted articles as “popular” articles and

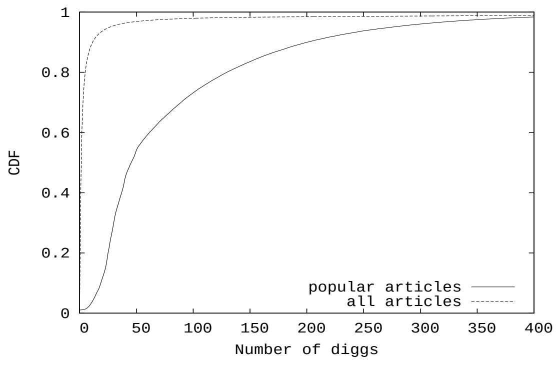


Figure 3.12: Distribution of diggs for all popular articles before being marked as popular and for all articles within 24 hours after submission.

displays them on its front page. In subsequent discussions, we use the terms *popular* or *popularity* only to refer to the popularity status of an article as marked by Digg. A Digg user can create a “follow” link to another user if he wants to browse all articles submitted by that user. We have crawled Digg to obtain the voting trace on all submitted articles since Digg’s launch (2004/12/01-2008/09/21) as well as the complete “follow” network between users. Unfortunately, unlike diggs, bury data is only available as a live stream. Furthermore, Digg does not reveal the user identity that cast a bury, preventing us from evaluating SumUp’s feedback mechanism. We have been streaming bury data since 2008/08/13. Table 3.2 shows the basic statistics of the Digg “follow” network and the two voting traces, one with bury data and one without. Although the strongly connected component (SCC) consists of only 15% of total nodes, 88% of votes come from nodes in the SCC.

There is enormous incentive for an attacker to get a submitted article marked as popular, thus promoting it to the front page of Digg which has several million page views per day. Our goal is to apply SumUp on the voting trace to reduce the number of successful attacks on the popularity marking mechanism of Digg. Unfortunately,

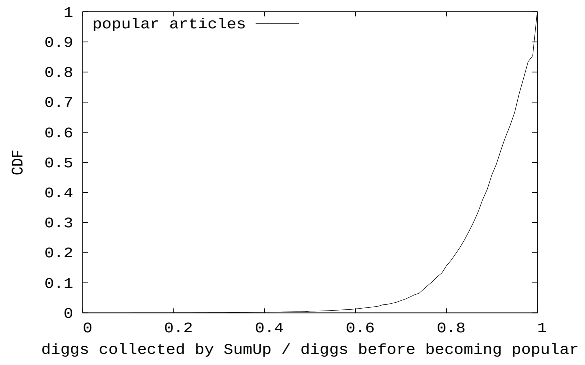


Figure 3.13: The distribution of the fraction of diggs collected by SumUp over all diggs before an article is marked as popular.

unlike experiments done in Section 3.5.2 and Section 3.5.5, there is no ground truth about which Digg users are adversaries. Instead, we have to use SumUp itself to find evidence of attacks and rely on manual sampling and other types of data to cross check the correctness of results.

Digg's popularity ranking algorithm is intentionally not revealed to the public in order to mitigate gaming of the system. Nevertheless, we speculate that the number of diggs is a top contributor to an article's popularity status. Figure 3.12 shows the distribution of the number of diggs an article received before it was marked as popular. Since more than 90% of popular articles are marked as such within 24 hours after submission, we also plot the number of diggs received within 24 hours of submission for all articles. The large difference between the two distributions indicates that the number of diggs plays an important role in determining an article's popularity status.

Instead of simply adding up the actual number of diggs, what if Digg uses SumUp to collect all votes on an article? We use the identity of Kevin Rose, the founder of Digg, as the vote collector to aggregate all diggs on an article before it is marked as popular. Figure 3.13 shows the distribution of the fraction of votes collected by SumUp over all diggs before an article is marked as popular. Our previous evaluation on various network topologies suggests that SumUp should be able to collect at least 90% of all votes. However, in Figure 3.13, there are a fair number of popular articles with much fewer than the expected fraction of diggs collected. For example, SumUp only manages to collect less than 50% of votes for 0.5% of popular articles. We hypothesize that the reason for collecting fewer than the expected votes is due to real world Sybil attacks.

Since there is no ground truth data to verify whether few collected diggs are indeed the result of attacks, we resort to manual inspection. We classify a popular article as suspicious if its fraction of diggs collected is less than a given threshold. Table 3.3 shows the result of manually inspecting 30 random articles out of all suspicious articles. The random samples for different thresholds are chosen independently. There are a number of obvious bogus articles such as advertisements, phishing articles and obscure political opinions. Of the remaining, we find many of them have an unusually large fraction ($>$30%) of new voters who registered on the same day as the article's submission time. Some articles also have very few total diggs since becoming popular, a rare event since an article typically receives hundreds of votes after being shown on the front page of Digg. We find no obvious evidence of attack for roughly half of the sampled articles. Interviews with Digg attackers [30] reveal that, although there is a fair amount of attack activities on Digg, attackers do not usually promote obviously bogus material. This is likely due to Digg being a highly monitored system with fewer than a hundred articles becoming popular every day. Instead, attackers try to help paid customers promote normal or even good content or to boost their profiles within the Digg community.

As further evidence that a lower than expected fraction of collected diggs signals a possible attack, we examine Digg's *bury* data for articles submitted after 2008/08/13, of which 5794 are marked as popular. Figure 3.14 plots the correlation between the

Threshold of the fraction of collected diggs	20%	30%	40%	50%
# of suspicious articles	41	131	300	800
Advertisement	5	4	2	1
Phishing	1	0	0	0
Obscure political articles	2	2	0	0
Many newly registered voters	11	7	8	10
Fewer than 50 total diggs	1	3	6	4
No obvious attack	10	14	14	15

Table 3.3: Manual classification of 30 randomly sampled suspicious articles. We use different thresholds of the fraction of collected diggs for marking suspicious articles. An article is labeled as having many new voters if $> 30\%$ of its votes are from users who registered on the same day as the article's submission date.

average number of bury votes on an article *after* it became popular vs. the fraction of the diggs SumUp collected before it was marked as popular. As Figure 3.14 reveals, the higher the fraction of diggs collected by SumUp, the fewer bury votes an article received after being marked as popular. Assuming most bury votes come from honest users that genuinely dislike the article, a large number of bury votes is a good indicator that the article is of dubious quality.

What are the voting patterns for suspicious articles? Since 88% diggs come from nodes within the SCC, we expect only 12% of diggs to originate from the rest of the network, which mostly consists of nodes with no incoming follow links. For most suspicious articles, the reason that SumUp collecting fewer than expected diggs is due to an unusually large fraction of votes coming from outside the SCC component. Since Digg's popularity marking algorithm is not known, attackers might not bother to connect their Sybil identities to the SCC or to each other. Interestingly, we found 5 suspicious articles with sophisticated voting patterns where one voter is linked to many identities (~ 30) that also vote on the same article. We believe the many identities behind that single

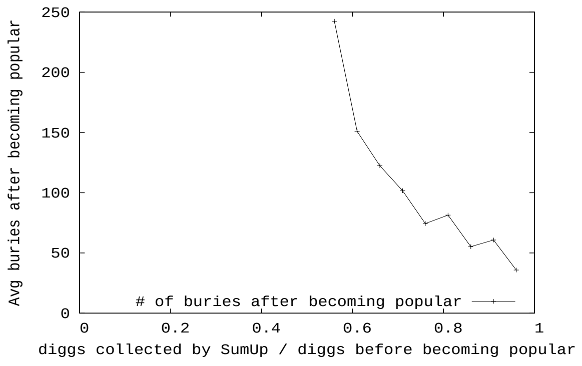


Figure 3.14: The average number of buries an article received *after* it was marked as popular as a function of the fraction of diggs collected by SumUp *before* it is marked as popular. The Figure covers $5,794$ popular articles with bury data available.

voter are likely Sybil identities because those identities were all created on the same day as the article's submission. Additionally, those identities all have similar usernames.

Chapter 4

Collusion-resilient credit-based reputation for peer-to-peer content distribution

With the recent growth in demand for high-quality multimedia content, capacity requirements for content distribution networks (CDNs) have increased proportionally. Peer-to-peer content distribution is a natural low cost option to scale distribution capacity. In a P2P CDN model, content providers serve content using a small number of "official" seeder nodes and rely on participating users to upload previously downloaded content to others. By aggregating the bandwidth of thousands or millions of participating users, P2P CDNs promise extremely high capacity at very low costs. However, in order to reach their full potential, P2P CDNs must address the long-standing challenge of incentivizing users to upload content to others.

The P2P incentive problem has been widely studied in the past. Unfortunately, popular solutions such as the tit-for-tat mechanism provided by BitTorrent [13] and variants [56, 38] are insufficient. BitTorrent only incentivizes those peers that are *actively* downloading the *same* file to upload to each other. Once a user completes a download, she has no incentive to act as a seeder and continue uploading. In practice, the distribution of content on popular torrent networks such as PirateBay is often heavy-tailed, with most files having only a few simultaneous downloaders. As a result, tit-for-tat is of little use as an incentive mechanism in these scenarios. An ideal incentive mechanism should motivate users to contribute to the P2P CDN even after completing their downloads. This is sometimes referred to as the *seeder promotion* problem [65]).

Existing studies [44] as well as our own measurement experiments suggest that P2P CDNs can achieve significant performance boost by addressing the seeder promotion problem (Section 4.1). However, the current solutions for solving the seeder promotion problem is unsatisfactory. PPLive and PPStream distribute proprietary software in that hope that the software cannot be modified to avoid uploading. However, proprietary software discourages third-party implementation and prevents external code auditing. Popular private BitTorrent communities such as TorrentLeech and What.CD maintain an invitation-only membership. These communities keep track of members' self-reported upload contribution, and kick out members that have failed to make the required amount of contribution. For example, in TorrentLeech, each peer must serve as a seeder for a file for 24 hours after downloading and upload at least 0.4 times of its downloaded amount. Unfortunately, selfish nodes can purposefully misreport their upload contributions, an attack that is becoming increasingly common in private BitTorrent communities [23]. Current defenses against such attacks are very limited and involves banning client software which can be modified to misreport information (e.g. Vuze, Deluge).

One promising direction for solving the seeder promotion problem is to design a robust reputation system. In such as system, a user's reputation score accurately reflects her upload contribution – to encourage contribution, P2P CDNs can give preferential treatment to users with high reputation; thus the more a peer contributes (in terms of its uploads) the better the service (in terms of download speed) it gets. The major challenge in designing such a reputation system is ensuring a user's reputation score accurately reflects their upload contribution; malicious users should not be able to acquire excessively high reputation scores via collusion or Sybil attacks. In this paper, we propose Credo, a *credit-based reputation* system that addresses both challenges and show how it can be applied to P2P CDNs to solve the seeder promotion problem.

To track a node's contribution in Credo, a seeder collects a signed upload receipt whenever it uploads a chunk of data to another node. Credo employs a central server to periodically aggregate upload receipts from nodes and compute a reputation score for each node based on these receipts.

Attacks via collusion and Sybil identities are particularly challenging for reputation systems. Specifically, a number of colluding adversarial nodes may generate a large number of upload receipts for each other using their respective Sybil identities without performing any actual uploads. Credo's reputation algorithm employs two techniques to defend against such attacks.

First, Credo bounds the reputation gain of attackers by measuring upload contribution using the concept of credit transfers. Credo keeps track of a credit set ($\mathcal{C}$) for each node and transfers a credit from node A's set to B if node B has uploaded a data chunk to A, as indicated by the corresponding upload receipt. A node with many uploads will have a large number of credits while a node with many downloads has large number of debits (i.e. there are many credits of that node in others' credit sets). Credo measures a node's reputation score by its credit sets *diversity* which counts no more than a few credits from the same node. Credit diversity allows Credo to bound the maximum reputation score of colluders: for k colluding adversaries each with s Sybil identities, their maximum reputation is only $\lambda \cdot k \cdot s$.

Second, Credo limits sustained collusion attacks where Sybil identities continuously generate upload receipts for colluding adversarial nodes. Sustained attacks are characterized by credit sets that contain disproportionally many credits from identities with large debits. Credo models the distribution of node debits and filters a node's credit set according to the measured distribution. The filtering step in Credo is important as it ensures that an adversarial node's reputation score will eventually decrease after it has downloaded a bounded amount of data from honest nodes.

We make three contributions in this paper.

- We propose the design of Credo. To the best of our knowledge, Credo is the first reputation scheme that accurately assign nodes reputation scores which reflect their contributions while providing quantifiable guarantees for resistance against Sybil and collusion attacks.

- We present an analysis of Credo's security guarantees in the face of k colluding adversarial nodes each controlling s Sybil identities. Each adversary's reputation score is upper bounded by $\lambda \cdot k \cdot s$. Furthermore, an adversarial node can download at most $O(s \cdot \bar{d})$ data chunks where $\bar{d}$ is the average debits of honest nodes before its reputation score is diminished.

- We show our implementation of Credo scales to handle a large number of peer nodes. Simulation results based on the transfer log of an existing P2P CDN show that Credo can accurately capture node contribution and defend against attacks in practical workloads.

4.1 Seeder promotion problem

The importance of seeder promotion. BitTorrent's tit-for-tat mechanism only motivates nodes that are currently downloading the same file to upload to each other. Once a node finishes downloading, there's no incentive for it to stay online and upload to others. Despite such lack of incentive, BitTorrent works fairly well in practice when there are a few altruistic high-capacity nodes available [57]. Therefore it is worth investgating how much practical performance gain is achievable if nodes are incentivized to stay online and become seeders after completing downloads.

In existing P2P distribution systems, most streams do not have a large number of simultaneous downloading nodes (leechers). In fact, the majority of data transfers ($> 80\%$) in various BitTorrent communities are between seeders and leechers [44]. For most streams, large numbers of simultaneous leechers only occur during the first few days when a very popular file initially appears. As a result, because seeders perform most of the uploads, the more seeders there are in the system, the better the download speed of leechers. Seeder promotion motivates nodes that have completed downloads to stay online and act as a seeder – this leads to more seeders and better performance. One way to quantify the impact of seeder promotion is to compare the performance of public BitTorrents to that of private BitTorrents. While there is no incentive for seeding in a public BitTorrent, a private BitTorrent demands a certain level of upload contribution from each node in order to maintain the node's membership.

We measured the download speed in a public BitTorrent system (PirateBay) as well as three private BitTorrent communities (Demonoid, What.CD and TorrentLeech). Of the three private BitTorrent communities, What.CD and TorrentLeech demand a certain minimum level of upload contribution (or sharing ratio) from each member in order to

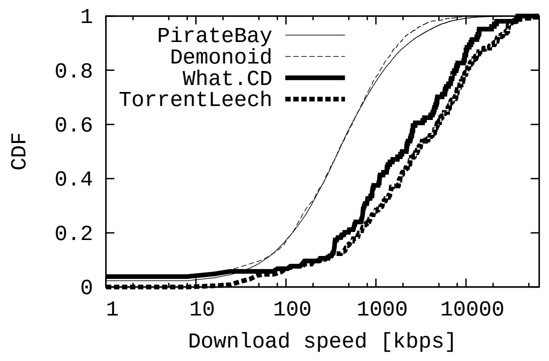


Figure 4.1: CDF of achieved download speeds in various BitTorrent communities. The download speeds in communities that incentivized seeding (TorrentLeech and What.CD) are significantly better than that of the public BitTorrent (PirateBay) and the private community (Demonoid) with no seeding incentives.

stay in the community while Demonoid has no such requirement. For each BitTorrent community, we joined 100 recently active swarms in that community and measured the download speeds of nodes in the swarm by periodically connecting to a node to obtain its download progress.

Figure 4.1 shows the cumulative distribution of observed download speed in all four BitTorrent communities. As we can see, private communities which incentivizes upload contribution (i.e. TorrentLeech and What.CD) achieve 6-7$\times$ the median download speed of PirateBay (a public BitTorrent) and Demonoid (a private community with no upload incentive). We also found that the ratio of seeders to leechers in TorrentLeech and What.CD is more then 10$\times$ those in PirateBay and Demonoid; this difference in available seeders is a large component of the performance gap between the private and public systems. Our observations are similar to another recent measurement study on a different set of private BitTorrent communities [44]. Based on these results, we hypothesize that P2P CDNs can achieve significant performance gain by addressing the seeder promotion problem.

Fairness incentivizes contribution. How to motivate selfish nodes to act as seeders? We assume that the utility of each peer is characterized by its average download speed and the goal of each peer is to employ a strategy that maximizes its download speed. It is worth pointing out that a selfish peer is *not* necessarily interested in minimizing its upload cost: each user has a different threshold for acceptable upload cost. This model of selfish peers is similar to that proposed in [38]. A P2P CDN is considered as *fair* if the more a peer contributes to the system (i.e. uploads) relative to its consumption (i.e. downloads), the better average download speed it experiences when competing with other downloaders. We hypothesize that, in a fair P2P CDN, nodes are motivated to act as seeders to achieve better download speeds in return.

Achieving fairness is more flexible than enforcing a specific sharing ratio as done in private BitTorrent communities such as TorrentLeech. With a specific sharing ratio, a peer has no incentives to upload more than its required sharing ratio. Worse yet, a peer unable to meet the sharing ratio requirement for various non-selfish reasons (e.g. it is seeding unpopular files or has small upload capacity compared to its download capacity) risks getting expelled from the system.

Our fairness notion maps naturally to a reputation system where each peer's reputation score reflects its net contribution (i.e. its uploads minus its downloads) and each peer allocates its upload capacity to active downloaders according to their reputation scores. However, this reputation based approach faces two practical challenges: (a) how to capture a peer's net contribution? (b) how to defend against attacks on the reputation system itself? The goal of our work is design a reputation system that addresses both these challenges.

4.2 Approach

At a high-level, Credo keeps track of each node's upload contribution using signed "receipts": in exchange for downloading a data chunk from seeder B, node A generates a signed receipt ($A \rightarrow B$) and gives it to B. Credo employs a central server to periodically aggregate upload receipts collected by all nodes and compute each node's reputation score based on these receipts. To promote contribution, each seeder preferentially selects nodes with higher reputation scores among competing download requests to upload data to.

Credo performs Sybil-resilient member admission control to prevent an attacker from joining the system with an arbitarily large number of Sybil identities. However, each attacker can still participate in the system with a few (s) Sybil identities to manipulate Credo's reputation mechanism. The main contribution of Credo is a centralized reputation algorithm that calculates a reputation score for each node based on the collection of upload receipts. The algorithm achieves quantifiable security guarantees for its defense against both Sybil and collusion attacks. In a Sybil attack, an adversarial node uses the few Sybil identities under its control to boost its own reputation score without performing actual uploads. Furthermore, several attackers can collude together with their respective Sybil identities to boost each others reputation score. Next, we describe the two key ideas in Credo's reputation algorithm.

1. Measuring contribution using credit diversity. The set of receipts from all nodes form the upload graph. Figure 4.2 shows two example graphs where the link $C \rightarrow B$ with weight 2 indicates that B has uploaded 2 data chunks to C. The naive method of measuring a node's *net* contribution is to calculate the difference between a node's weighted incoming links and its outgoing links. For example, in Figure 4.2(a), A's net

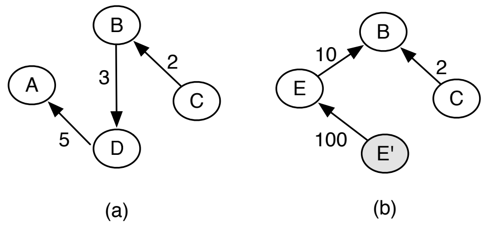


Figure 4.2: The upload graph formed by receipts, e.g. the link C→B of weight 2 denotes that B has uploaded 2 data chunk to C. The naive method of measuring each node's net contribution by the difference between a node's weighted incoming links and outgoing links is vulnerable to the Sybil attack. For example, in (b), A instructs its Sybil identity A' to generate a large number of upload receipts for A.

contribution is calculated as 5. The naive method accurately captures the net contribution of *honest* nodes but is extremely vulnerable to Sybil and collusion attacks. For example, in Figure 4.2(b), attacker E instructs its Sybil identity E' to generate 100 upload receipts for E, thereby increasing the net contribution of E to 90 without having to perform any actual uploads. How can we measure a node's net contribution while remaining resilient to this type of attack?

Intuitively, node A's upload contribution in Figure 4.2(a) is more "diverse" than that of E in Figure 4.2(b) because the node (D) that A has uploaded to has also uploaded to other nodes while the node (E') that E has uploaded to has made no contribution. We capture this notion of diversity using the concept of credit transfers. Credo's reputation algorithm maintains two quantities for each node: (1) the node's credit set ($\mathcal{C}$), represented as a multi-set. (2) the node's debits (d), represented as a number. For example, $\mathcal{C}_A = \{C : 1, D : 2\}$ indicates that A's credit set consists of 1 credit issued by C and 2 credits issued by D. The algorithm processes every link in the upload graph by performing a "credit transfer". For example, to process $A \rightarrow B$, the algorithm removes one randomly chosen credit from A's credit set and adds it to B's set. The issuer of a credit does not change as it is transferred to another credit set. If A's credit set is empty, the algorithm increments A's debits (d_A) by 1 and adds a new credit issued by A to B's credit set.

The credit-based processing causes nodes with more "diverse" contribution to have credit sets with more distinct credits. For example, A's credit set in Figure 4.2(a) may become $\mathcal{C}_A = \{C : 1, B : 3, D : 1\}$ while E's credit set in Figure 4.2(b) is simply $\mathcal{C}_E = \{E' : 100\}$. We measure *credit diversity* by the number of distinct issuers in a credit set. However, since a node might repeatedly upload to the same node, we count $\lambda > 1$ credits from each distinct issuer. The default λ value is 3. For example, for $\mathcal{C}_A = \{C : 1, B : 3, D : 1\}$, $diversity(\mathcal{C}_A) = 5$. For $\mathcal{C}_E = \{E' : 100\}$, $diversity(\mathcal{C}_E) = 3$.

A node's reputation is calculated based on its credit diversity and its debits:

$$rep = diversity(\mathcal{C}) - d \tag{4.1}$$

Credit diversity limits the maximum reputation gain of Sybil attacks. If an adversarial node has only s Sybil identities, its maximum reputation score is only $\lambda \cdot s$ without performing any uploads. Moreover, even if a set of k adversaries each with s Sybil identities collude, the reputation score of each adversarial node is bounded by $\lambda \cdot k \cdot s$.

Credit diversity may under-estimate the contribution of an honest node if it repeatedly performs many uploads to another node who has done little contribution itself, a behavior indistinguishable from that of an adversarial node launching Sybil attack. Fortunately such a scenario is unlikely to occur – honest node preferentially upload to the downloaders with the highest reputation, and as high reputation nodes have a large and diverse credit set themselves, an honest seeder will able to increase its credit diversity by obtaining upload receipts from them, as our later evaluations demonstrate (Section 4.6).

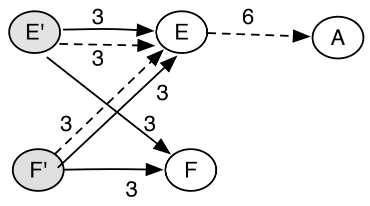


Figure 4.3: Two adversarial nodes (E,F) collude by obtaining from each of their respective Sybil identities (E',F') 3 upload receipts. As E downloads from honest node A (dotted link $E \to A$), E can replenish its diminished credit set by obtaining new upload receipts from Sybils (dotted links $E' \to E$, $F' \to F$).

Existing graph-based reputation schemes are based on Eigenvalue [6, 31] or max-flow computations [21, 9]. However, with Eigenvalue [6, 31] methods, a node's reputation score does *not* necessarily reflect its net contribution. In particular, a node can improve its reputation score more by uploading the same amount of data to those nodes with higher reputations. Credo employs a similar credit transfer mechanism as currency systems [55, 79, 11, 65], but its use of credits is very different. Currency systems give the same download service to all node who possess currency tokens and deny service to "bankrupted" nodes. By contrast, Credo uses each node's credit set to calculate its reputation score and gives preferential service to those nodes with higher reputations.

2. Credit filtering based on good behavior. Although using credit diversity limits the maximum reputation score, each colluding adversarial node can still maintain its maximum reputation score without any contribution no matter how much data it has downloaded from others. That is because every Sybil node can issue arbitrarily many upload receipts to replenish the credit set of an adversarial node. For example, in Figure 4.3, adversarial nodes E and F collude with their respective Sybils to achieve a credit set of diversity 6. If E downloads 6 units of data from honest node A (shown by the dotted link $E \to A$), E's credit set should ideally decrease by 6. However, E could easily request more upload receipts from the Sybils E' and F' (shown by dotted links $E' \to E$, $F' \to E$) to maintain a credit diversity of 2λ at all times no matter how much data it downloads from honest nodes.

To mitigate such sustained collusion attacks, Credo's reputation algorithm explicitly models the typical behavior of honest nodes. More concretely, the algorithm measures the distribution of debits of all nodes. The debit distribution as observed in the credit set of an honest node should not deviate too much from the measured distribution. We use such knowledge to filter a node's credit set to obtain a subset of credits, $\mathcal{C}' \subset \mathcal{C}$, such that the observed debit distribution in $\mathcal{C}'$ conforms to the measured distribution. We augment Equation 4.1 to use the filtered set ($\mathcal{C}'$) for calculating credit diversity. The filtering step significantly limits an adversarial node' ability to carry on sustained attacks.

Summary: The combination of ideas 1 and 2 defends against Sybil and collusion attacks. Specifically, credit diversity limits the maximum reputation score of an attacker. Credit filtering ensures that an adversarial node's reputation score goes down after it has downloaded a bounded amount of data from honest nodes.

4.3 Credo Design

In this section, we first describe the overall system architecture including how nodes generate upload receipts and how reputation scores are utilized. Next, we explain how Credo computes node reputations based on aggregated upload receipts.

4.3.1 System Architecture

The Credo system consists of a trusted central server as well as a large collection of peer nodes. The central server performs Sybil-resilient node admission control and is responsible for aggregating upload receipts and computing reputation for all peers. A peer node may act as a *seeder* who stores a complete file and uploads chunks of it to others. A peer node may also act as a *leecher* to download missing file chunks from other seeders or leechers. Credo focuses on the interaction between seeders and leechers which constitute the majority (¿80%) of data transfers [44] and lets the normal BitTorrent protocol handle data exchange among leechers.

Node admission: The central server admits a new node into the system by generating a public/private key pair for the admitted node. A node can prove its membership in the system to another node by presenting its public key certificate signed by the central server. Credo must prevent an attacker from joining the system with an arbitrarily large number of Sybil identities. The central server employs existing Sybil-resilient admission control methods such as schemes based on a fairly strong type of user identity (such as credit card numbers or cellphone numbers) or algorithms based on the social network among users [83, 84, 72]. Sybil-resilient admission control cannot prevent an adversarial node from joining the system but limit each adversarial node to a small number (s) of Sybil identities.

Obtain upload receipts for seeding: A seeder gets an upload receipt after uploading one data chunk (of size up to 1MB) to a leecher. For example, if seeder A has uploaded to B, A obtains the receipt of the form [$A \leftarrow B$, SHA1(data), ts] signed by B's private key. SHA1(data) is the hash of the data block being uploaded from A to B and ts is the current timestamp according to B.

A leecher might refuse to generate the required upload receipt after downloading data from a seeder. To deter such behavior, we borrow the fair-exchange protocol from BAR Gossip [40]. Specifically, to upload data to B, seeder A first transfers the encrypted data chunk to B and then gives B the decryption key only if A receives a valid upload receipt from B. The symmetric key used by A to encrypt data is uniquely determined by A's private key (Prv_A) and the content hash of the data chunk (SHA1(data)). In addition to the encrypted data, A also includes a signed tuple [SHA1(data),SHA1(encrypted data)] to bind its words that the encrypted data correspond to the data chunk being requested. B can only obtain the decryption key after sending the proper upload receipt to A. In the case that B does not receive any valid decryption key from A after it has sent the upload receipt, B sends the upload receipt to the central server. Since the central server knows all nodes' private key, it can re-generate the decryption key based on A's private key and SHA1(data) and give it to B. If seeder A uploads garbage data to B, B will report A's misbehavior to the central server with a verifiable proof including the encrypted data chunk and the signed [SHA1(data),SHA1(encrypted data)].

Aggregate upload receipts: Every peer node periodically transfers its newly received upload receipts to the central server. Since all honest nodes should verify every received upload receipt, the central server only samples a small fraction (e.g. 10%) of aggregated receipts to check their validity. Upon detecting an invalid receipt, the server can punish the corresponding seeder for presenting the bogus receipt by reducing its reputation score or suspending its membership.

The central server computes a reputation score for each node every few hours (default is four). Only upload receipts generated in the last τ time period are used for reputation computation. We pick τ to be two weeks, a period short enough so that a node is motivated to continuously contribute to the system and yet is also long enough for a node's past contribution to affect its current reputation score.

Each node can request a signed reputation certificate from the central server indicating its current reputation score. A leecher presents the reputation certificate as part of it download request to a seeder. Each seeder in Credo designates a small number of upload slots to serve a few leechers at a time. When there are more download requests than upload slots, the seeder picks those leechers with top reputation scores among all competing requests to serve.

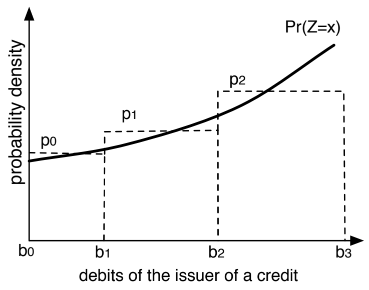
4.3.2 Credit-based reputation computation

The most interesting component of Credo is the algorithm used by the central server to compute node reputation based on the collection of aggregated upload receipts. As summarized earlier in Section 4.2, the algorithm processes the upload graph by performing credit transfers between nodes. The algorithm resists against Sybil and collusion attacks by filtering each node's credit set according to modeled honest behavior and quantifying a node's upload contribution by its credit diversity.

4.3.2.1 Credit transfer

The set of upload receipts form a directed graph with weighted links. A cycle in the graph with the same weight on each link represents a fair exchange among those nodes. Therefore, the algorithm first prunes the graph by removing such cycles. For example, if there exist link $A \rightarrow B$ with weight 5, link $B \rightarrow C$ with weight 2, and link $C \rightarrow A$ with weight 2, the pruned graph only contains link $A \rightarrow B$ with weight 3. Since all cycles are removed, the pruned graph becomes a direct acyclic graph (DAG).

The algorithm processes the graph in the topological sort order, i.e. a node is processed only if all of its predecessors were processed. For example, nodes in the graph of Figure 4.2(a) is processed in the order C, B, D, A and the ordering of processing for Figure 4.2(b) is {E',C}, E, B. All nodes initially have an empty credit set ($\mathcal{C} = \varnothing$) and zero debits ($d = 0$). Each node is processed by examining all links pointing to it. To process link $A \rightarrow B$ of weight w, the algorithm first checks if the number of credits in



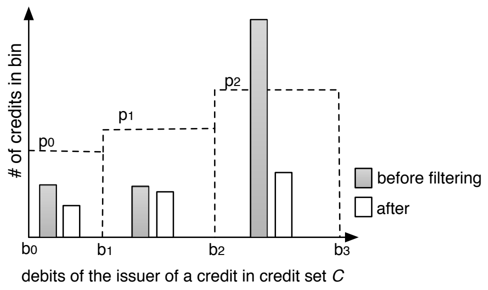


Figure 4.4: Credo models the behavior of honest nodes with Z-distribution (Z is the random variable corresponding to the number of self-issued credits by the issuer of a randomly chosen credit). Credo represents Z-distribution using m bins with exponentially increasing sizes.

Figure 4.5: Credo limits sustained collusion attack using the p_i test. In the example, the grey bars correspond to $\mathcal{C}$ of an adversarial node. The white bars correspond to the resulting $\mathcal{C}'$ that fits Z-distribution (dotted lines).

$\mathcal{C}_A$ is more than w. If it is the case, we subtract w randomly chosen credits from ($\mathcal{C}_A$) and adds them to $\mathcal{C}_B$. If that is not the case, then we make up for the difference x by adding x credits issued by A to $\mathcal{C}_B$ and incrementing A's debits by x. In Figure 4.2(a), C ends up with credit set $\mathcal{C}_C = \varnothing$ and $d_C = 2$ and B has credit set $\mathcal{C}_B = \varnothing$ and $d_B = 1$. In Figure 4.2(b), B ends up with credit set $\mathcal{C}_B = \{E' : 10, C : 2\}$ and $d_B = 0$. The algorithm continues until it has processed all links in the DAG.

After processing the graph, to calculate a node's reputation score, the algorithm first filters its credit set $\mathcal{C}$ to obtain a set of "good" credits, $\mathcal{C}' \subset \mathcal{C}$. It then computes the diversity of $\mathcal{C}'$ to bound the maximum reputation gain of collusion and Sybil attacks. When calculating $diversity(\mathcal{C}')$, we count no more than λ credits from the same node. The final reputation score is calculated as $diversity(\mathcal{C}') - d$. In the next section, we explain how the filtering step is performed to mitigate sustained collusion and Sybil attacks.

4.3.2.2 Credit filtering

The goal of credit filtering is to choose a subset of credits $\mathcal{C}'$ such that the distribution of the debits for the issuers of credits in $\mathcal{C}'$ approximates the overall debit distribution of honest nodes in the system. Since an adversarial node performing sustained Sybil and collusion attacks ends up with a credit set that may significantly deviate from the overall debit distribution, the filtering step will effectively remove many credits from the adversary's credit set, leading to a diminished reputation score.

Computing the overall debit distribution: Let X be the random variable of the debit of a node chosen randomly among those nodes with positive debits. Since a few Sybil identities may skew the distribution of X with extremely large debits, we use a truncated distribution of X that excludes a small fraction (δ) of nodes with the most debits. As a result, as long as the set of colluding Sybil identities do not exceed δ of all nodes, they cannot affect the measured distribution X.

Let Z be the random variable of the debit of the issuing node for a randomly chosen credit in the credit sets of all nodes. We model an honest node's credit set as a collection of randomly chosen credits in the system. Thus, we expect the debit distribution corresponding to collection of credits in an honest node's credit set to approximate the distribution of Z.

We can derive the distribution of Z from that of X as follows,

$$Pr(Z = x) = \frac{Pr(X = x)x}{E(X)} \tag{4.2}$$

The algorithm represents the Z-distribution using a set of probability density bounds that correspond to m bins, as shown in Figure 4.4. Let the range of the i-th bin be $[b_i, b_{i+1})$. The ranges of the bins are chosen to so that the size of successive bins increases exponentially, i.e. $\frac{b_{i+1}}{b_i} = \gamma$ where γ is a small constant bigger than 1. Our security bound for Credo's collusion resilience is dependent on the choice of γ (Section 4.4). The probability density of the i-th bin is calculated as $Pr(b_i \leq Z < b_{i+1})$. Let $p_i = \frac{Pr(b_i \leq X < b_{i+1}) \cdot b_i}{E(X)}$. We can see that p_i is the lower bound of $Pr(b_i \leq Z < b_{i+1})$, because:

$$Pr(b_i \leq Z < b_{i+1}) \quad = \quad \frac{\sum_{b_i \leq x < b_{i+1}} Pr(X = x) \cdot x}{E(X)}$$

The set of lower bounds p_i is used to ensure that the observed debit distribution for credits in the filtered credit set does not deviate too much from the overall Z-distribution. In particular, let Z' be the random variable of the debit of the issuing node for a random credit in a filtered credit set $\mathcal{C}'$, it must satisfy $Pr(b_i \leq Z' < b_{i+1}) \geq p_i$ for all i.

Filter credit set using Z-distribution: In order to extract a subset of credits $\mathcal{C}'$ whose debit distribution matches the overall Z-distribution, the algorithm chooses credits for $\mathcal{C}'$ so that the fraction of credits in the i-th bin exceeds p_i. A credit is classified to the i-th bin if its issuer has a debit value within $[b_i, b_{i+1})$.

When there is a sustained collusion attack, the credit set of an adversarial node consists of an "unusually" large number of credits in bins with large i's because their issuers (Sybils) have very large debits. This causes the fraction of credits in bins with small i's to be lower than the required lower bound. The filtering step will evict credits in the bin of large i's in order to increase fraction of credits in bins with smaller i's. Figure 4.5 gives an example. As can be seen, the original credit set $\mathcal{C}$ consists of many credits in bins corresponding to issuers with large debits. (the high grey bar at the rightmost side). After filtering, the number of credits accepted in that range is significantly reduced (the white bar at the rightmost side) according to the expected lower bound for each bin (the dotted bins).

The filtering process proceeds as follows. Let c_i be the number of credits classified to the i-th bin. When filtering the credit set to arrive at $\mathcal{C}'$, the algorithm ensures that:

$$\frac{c_i}{|\mathcal{C}'| \cdot p_i} \geq 1 \qquad \forall i \in [0, m) \tag{4.3}$$

We use a greedy heuristic for picking a set of credits that pass the test specified by Equation 4.3. In particular, we start with the original credit set $\mathcal{C}$ and check for the validity of the test. If the test fails with the i-th bin having the highest value of $\frac{c_i}{|\mathcal{C}'|p_i}$, we remove one credit from the i-th bin. We prefer to remove the credit which has the same issuer with at least λ other credits in the set. If there is no such credit, we remove a random one. We repeat this process until the test passes or no more credits are left. The remaining credits form the filtered set $\mathcal{C}'$.

4.4 Security properties

In this section, we present analysis results to quantify how Credo limits sustained collusion attacks. Colluding adversarial nodes exchange upload receipts issued by their Sybil identities (see example in Figure 4.3). We assume that colluders are self-interested individuals: they divide the Sybil-issued receipts among themselves so that each adversarial node is benefitted equally from the collusion. For the simplicity of discussion, we only show the analysis for the scenario where adversarial nodes do not contribute any uploads to the system.

Theorem 4.1. *Suppose there are k self-interested colluding adversarial nodes, each* *with s Sybil identities. Credo limits the maximum reputation of an adversarial node to be $\lambda \cdot k \cdot s$. More importantly, the average number of data chunks that an adversarial node can download with the maximum reputation score is at most $s \cdot \gamma \cdot \bar{d}$, where $\bar{d}$ is the average debits of an honest node.*

Proof. Since an adversarial node has no upload contribution, the credits in its credit set are belong to Sybil identities in the collusion group. Hence, the maximum credit diversity of k colluding adversaries is $\lambda \cdot k \cdot s$, resulting in maximum reputation score of $\lambda \cdot k \cdot s$.

Next, we prove the bound on the maximum downloads an adversarial node can perform with maximum reputation. Let X' be the random variable of the debit of a Sybil identity and let Z' be the random variable of the debit of the issuer (a Sybil identity) of a randomly chosen credit among the credits of adversarial nodes. We know that $Pr(Z' = x) = \frac{Pr(X'=x)x}{E(X')}$. Since adversarial nodes divide the upload receipts issued by Sybils among themselves, the debit distribution for each adversary's credit set can be approximated by the overall distribution Z'.

The filtering process ensures that the filtered credit set of an adversary passes the set of p_i tests, i.e.:

$$\begin{aligned} p_i &< Pr(b_i \leq Z' < b_{i+1}) \\ &< \frac{Pr(b_i \leq X' < b_{i+1}) \cdot b_{i+1}}{E(X')} \end{aligned} \tag{4.4}$$

Substituting $p_i = \frac{Pr(b_i \leq X < b_{i+1}) \cdot b_i}{E(X)}$ into Inequality 4.4 and re-arranging sides, we obtain:

$$\begin{aligned} E(X') &\leq \frac{b_{i+1}}{b_i} E(X) \frac{Pr(b_i \leq X' < b_{i+1})}{Pr(b_i \leq X < b_{i+1})} \\ &= \gamma \cdot E(X) \cdot \frac{Pr(b_i \leq X' < b_{i+1})}{Pr(b_i \leq X < b_{i+1})} \end{aligned} \tag{4.5}$$

In Inequality 4.5, γ is determined by the number of chosen bins (m) such that $\gamma = \frac{b_{i+1}}{b_i}$ for all i. Moreover, Inequality 4.5 holds true for all i. As the last step of simplification, we use the property that for any given positive numbers a, b_1, c_1, b_2, c_2, if $a \leq \frac{b_1}{c_1}$ *and* $a \leq \frac{b_2}{c_2}$, then $a \leq \frac{b_1+b_2}{c_1+c_2}$. Applying this observation to Inequality 4.5 for all i, we obtain:

$$E(X') \leq \gamma \cdot E(X) \tag{4.6}$$

Because the total number of credits from $k \cdot s$ Sybil identities is $k \cdot s \cdot E(X')$, each adversarial node has at most $s \cdot E(X') \leq s \cdot \gamma \cdot E(X)$ credits in its credit set. By definition, $E(X)$ is the expected debits of nodes after excluding those $\delta \cdot n$ nodes with the most debits, $E(X) \leq \bar{d}$ where $\bar{d}$ is the expected debits of an honest node with positive debit. Thus, we derive $E(X') \leq s \cdot \gamma \cdot \bar{d}$.

It is interesting to note that, for a given range $[b_0, b_m]$ of the distribution of X, the system parameter of the number of bins (m) uniquely determines γ. Specifically, $\gamma = \sqrt[m]{\frac{b_m}{b_0}}$. Therefore, γ decreases as we increase the number of bins (m), improving the bound on sustained collusion attacks. However, when m is too large, we also risk filtering out too many credits from a honest node's credit set unnecessarily. $\square$

4.5 Implementation

We have built the central server and the client node implementation using Java.

Central server: Our server implementation consists of $3000+$ lines of codes. The implementation logs upload receipts received from client nodes to disk. Every 4 hours, the server reads all collected receipts from disk and invokes the reputation algorithm to compute node reputation. The reputation calculation algorithm employs 12 concurrent threads to achieve speedup on multicore machines.

Credo client: Our client implementation is based on the open-source Azureus BitTorrent implementation. A Credo client can engage in the original BitTorrent protocol with other BitTorrent clients. When two Credo clients first meet, they exchange *CredoHandshake* messages (a new message type that we added to the existing BitTorrent protocol). A *CredoHandshake* consists the node public key certificate and its reputation score signed by the central server.

After finishing handshake, if both peers are leechers, they use the normal BitTorrent protocol to exchange data blocks among themselves. Otherwise, the seeder chooses the leecher with the highest reputation to serve. Specifically, we modified the two functions *calculateUnchokes* and *getImmediateUnchokes* in SeedingUnchoker.java to pick the leecher with the highest reputation to unchoke.

Once a leecher is unchoked, it sends a *CredoRequest* message to request a set of data blocks. The total size of data blocks requested is less than 1MB (the default data size per upload receipt in Credo). The seeder uploads encrypted data to the leecher. After finishing downloading, the leecher sends a *CredoPay* message with the required upload receipt. Finally, the seeder sends back a *CredoKey* message that contains the decryption key for the data. Every hour, every seeder sends its collection of receipts received in the last hour to the central server.

4.6 Evaluation

This section evaluates whether Credo reputation's reputation scheme gives the right incentive for nodes to contribute in the face of collusion and Sybil attacks. Our key results are:

- Nodes with higher net contribution achieves faster downloads in Credo. Thus, Credo gives nodes an incentive to contribute as seeders.
- Colluders who do not upload any data to the system have bounded maximum reputation scores. Furthermore, as they download data from honest nodes, their reputation scores decrease.
- The implementation of Credo incurs reasonable traffic and computation overhead and can potentially scale to handle a large number of peer nodes.

We use a combination of simulations and experiments with our prototype implementations to demonstrate these results.

4.6.1 Simulations

4.6.1.1 Simulation setup

We simulate a network of 3000 nodes for a 1 month period. We set the upload speed limit of each node to be $200KB$ per second. A node divides its upload capacity among 4 upload slots of $50KB$ per second each. The download speed of a node is 5 times its upload speed, i.e. $1MB$ per second. We control the upload contribution of a node by the *willingness* parameter. Whenever a seeder has a free upload slot, it decides to upload to some leecher with a probability proportional to its willingness. We set the willingness of nodes in our simulation to follow the distribution of upload capacity as measured

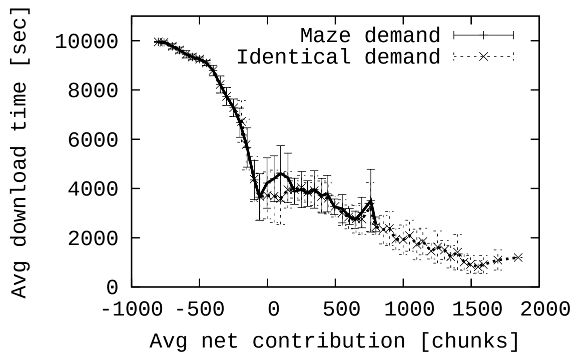


Figure 4.6: Average download time as a function of a node's net contribution. The average download time decreases as a node's net contribution increases

in [56]. Our simulations set the receipt expiration period (τ) to be 1 day. This is much smaller than our default τ value of 2 weeks because our simulated network is relatively small ($n = 3000$ nodes). Thus, we must use a smaller τ so that not many nodes can reach the maximum reputation score of $n \cdot \lambda$ during a τ time period

We inject new files of $200MB$ to the system sequentially: a new file is injected when all nodes that want a particular file have finished downloading it. Not every node wants every file: when we inject a new file, we randomly choose 300 nodes to download the file. The probability that a node is chosen to download the file is proportional to its demand. We model two types of demand: 1) all nodes have identical demands, 2) the demand of a node follows the demand distribution observed in the Maze file sharing system [80]. We also choose 10 random nodes as the initial seeders.

4.6.1.2 Credo incentivizes contribution

More contribution leads to faster download: Figure 4.6 plots the download time as a function of a node's net contribution. We measure a node's net contribution during the last τ time period as the number of its uploaded chunks minus the number of its down-

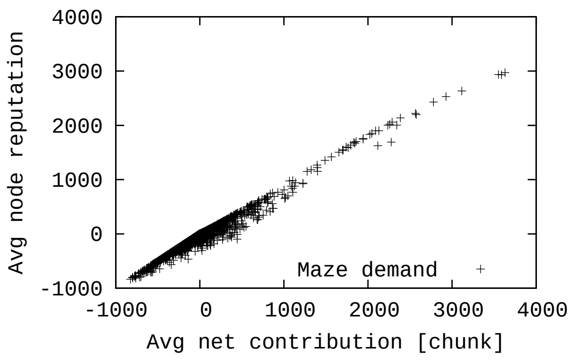


Figure 4.7: Average node reputation as a function of a node's net contribution. A node's reputation score increases linearly as its net contribution increases.

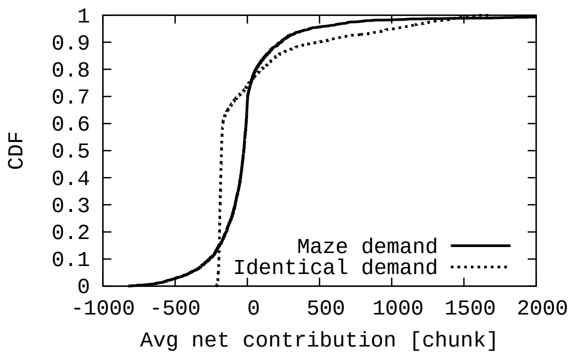


Figure 4.8: Cumulative distribution of nodes' average net contribution. A large fraction of nodes (0.7) have negative net contribution.

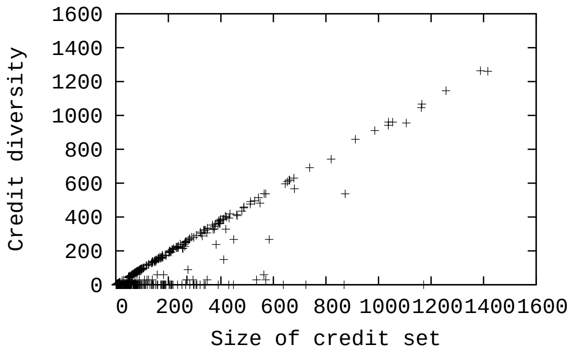


Figure 4.9: Credit diversity as a function of credit size. Credit diversity has strong correlation with large credit size (> 200), and weaker correlation for nodes with smaller credit size.

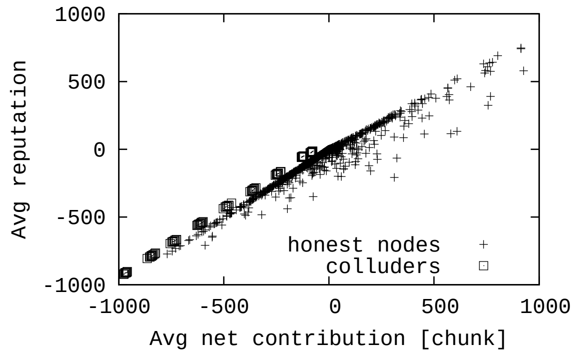


Figure 4.10: Reputation of colluding adversaries and honest nodes as a function of net contribution. Colluding adversaries vary demand in different simulations. Adversary nodes with high demand use up credits from Sybils, issue new credits, and get a negative reputation. As demand decreases, adversaries' reputations increase, but they are bounded by 3 time the number of Sybil nodes (60).

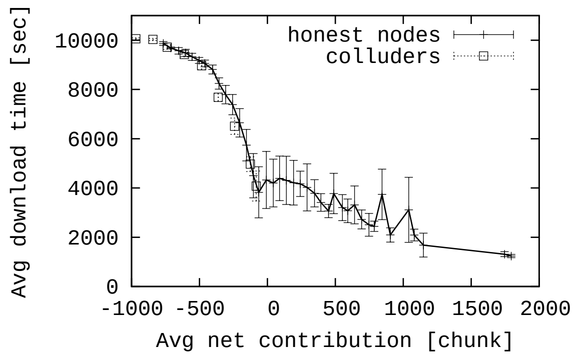


Figure 4.11: Download time of colluding adversaries and honest nodes as a function of net contribution; colluding adversaries vary demand at different simulations. As demand decrease, adversaries' download time decreases but it is higher than that of nodes with net contribution of 60.

loaded chunks. We record the net contribution and download time after a node finishes downloading a file. For the results shown in Figure 4.6, we grouped the net contribution of different nodes into bins of size 50, and computed the average and standard deviation of the download time in each bin.

Figure 4.6 shows that a node achieves faster download times when it has a higher net contribution, for both the Maze demand model and the identical demand model.

A node's reputation reflects its net contribution: A node achieves faster downloads when it has a higher reputation. Figure 4.7 shows a node's reputation as a function of its net contribution. We recorded a node's reputation score as well as its net contribution when it finishes downloading a file. We computed the average net contribution and average reputation for each node and plot the two quantities in Figure 4.7. We can see the reputation increases linearly as a node's net contribution increases.

To further quantify how Credo's reputation score accurately captures a node's net contribution, we calculate the standard A' metric, which is defined to be the probability that the reputation of a node A is greater than the reputation of node B given that A has higher net contribution for two random nodes A and B in the network. In Figure 4.7, $A' = 0.95$ which shows that Credo's reputation score accurately reflects a node's net contribution.

Figure 4.7 is divided into two regions: negative net contribution on the left, and positive net contribution on the right. We observe that the derivative of the curve on the left is approximately 1. That is because nodes with negative net contribution rarely earn credits, i.e. $|d(\mathcal{C}')| \approx 0$. The reputation of those nodes is essentially the negation of their debits. Figure 4.8 shows the cumulative distribution of nodes' net contribution. We can see that more than 70% of nodes have negative net contribution in simulation with both the Maze and identical demand models.

Nodes with positive net contribution are rarely associated with positive debits. Their reputation is mainly the diversity of their credit sets. The derivative of the curve in the positive contribution region of Figure 4.7 is only slightly smaller than 1. This shows that filtering credits and measuring credit diversity do not significantly under-estimate the net contribution of honest nodes.

We also take a closer examination on how the reputation scores for nodes with positive net contributions. Figure 4.9 shows credit diversity as the function of the size of credit set at end of the simulation. We observe that credit diversity is close to the size of the credit set when the size is greater than 200. This means that credit filtering has little negative effect on the credit sets of honest nodes with enough upload contribution. When the credit set size is smaller than 200, there are some sets whose credit diversity is much smaller than their actual size. This is because when the credit set is small, it is difficult to approximate the Z-distribution. As the result, many credits have been filtered.

4.6.1.3 Credo's defense against colluders is robust

We evaluate how Credo performs under the collusion attack by designating 10 nodes as adversarial nodes. Each adversarial node controls 2 Sybil nodes. They collude with each other to form a collusion size of 30, i.e. 1% of the system. In each τ interval, the Sybil nodes issue upload receipts to optimize the amount of credits that can pass the filtering step and achieve the bound in Observation 4.1 (Section 4.4). The credits are divided equally to adversarial nodes. Adversaries use those credits to download files. We also vary the the number of files that adversaries want to download in different simulations.

In Figure 4.10, we plot the reputation of the adversarial nodes as well as honest nodes as a function of their net contributions. Because the adversarial nodes never upload to other nodes, their net contribution are always negative. Their maximum reputation score is 60 because there are 20 colluding Sybil nodes. As their demand of downloading increases, adversarial nodes require Sybils to issuing more and more upload receipts which are eventually filtered. As a result, their reputation scores decrease.

We plot the download time as a function of the net contribution for adversarial nodes as well as honest nodes in the simulations in Figure 4.11. As expected, the download time increases as adversary nodes' demand increase, i.e. their net contribution decreases. Even when adversarial nodes have small demand, i.e. their net contribution is close to 0, their download times are still longer than that of honest nodes with net contribution 60. This is due to their reputation scores being bounded by 60.

4.6.1.4 Credo vs Eigenvector-based reputation

An alternative way to compute nodes' reputation at the central server is finding the eigenvector of the contribution graph induced from upload receipts. This eigenvector-based reputation such as PageRank was originally designed for web ranking [6] but has been used to compute node reputations in P2P systems [41]. However, the reputation score of the eigenvector-based scheme is not designed to capture the net contribution of a node. We show that when used in a P2P CDN, PageRank scores do not accurately reflect a node's net contribution and is also more vulnerable to collusion than Credo.

For real world workload, we use a Maze trace collected in December 2005 which records two weeks upload and download activities of nodes in the system. A node A that upload wMB to node B is represented by a directed edge from B to A in a graph G with a weight w on it. We compute eigenvector-based reputation using the standard PageRank method. We vary the probability of resetting a random walk ϵ, and find that the value of $= 0.15$ produces the best balance between defending against collusion and capture net contribution. We refer to this eigenvector-based reputation as PageRank reputation. To model attack, we choose 50 nodes that have uploaded at least 50MB and have the lowest net contribution as adversaries. Each adversary introduces 5 more Sybils to the graph, i.e. they form a collusion group of 300 nodes. They perform collusion by having the 50 adversaries create high weight ($1,000,000$) edges to each other. Each pair of an adversary and a sybil also create a pair of high weight directed edges. Figure 4.12 shows PageRank reputation of honest nodes and adversaries using the Maze workload. Each point in the graph represents a node in the system. As we can see, PageRank reputation does not reflex net contribution well ($A' = 0.71$). Moreover, it has very poor defense against collusion. The 50 adversaries we choose are among the bottom 0.2% in term of net contribution, but they can get in top 2% in term of reputation by colluding.

To compute Credo reputation, we feed the central server with receipts generated from the Maze trace. The same 50 adversaries collude by having the 250 sybils issue credits in an optimal way assume that they know the Z-distribution. Figure 4.13 shows the reputation of honest nodes and adversaries as a function of net contribution. As we can see, Credo reflects net contribution well ($A' = 0.96$). We notice that the reputation of nodes that have net contribution greater than $5,000$ are much smaller than the size of their credit sets. That is because these nodes upload huge amount of data (5GB) in 5 days, and they upload much more than 3MB to other nodes. The diversity technique penalizes these nodes' reputation because of their repeat interaction with other nodes. Nevertheless, these nodes still remain as top reputation nodes. This graph also shows that Credo's defense against collusion is better than PageRank. The 50 adversaries remain in the bottom 22% in term of reputation even after colluding.

We also do the same comparison on a synthetic workload from a 1000 nodes net-

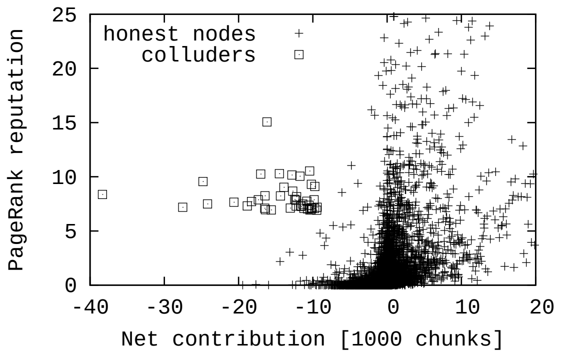


Figure 4.12: Eigenvector-based (PageRank) reputation of colluding adversaries and honest nodes as a function of net contribution on Maze workload. Adversaries who are in the bottom 0.2% of net contribution can get in the top 2% of reputation by colluding. Eigenvector-based reputation does not reflect net contribution well among honest nodes ($A' = 0.71$).

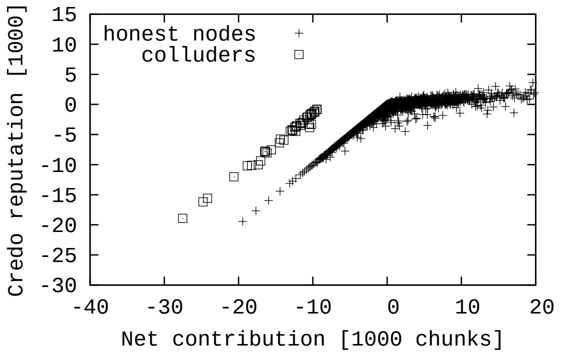


Figure 4.13: Credo reputation of colluding adversaries and honest nodes as a function of net contribution on Maze workload. Adversaries who are in the bottom 0.2% of net contribution can only get in the top $80-60\%$ of reputation by colluding. Credo reputation reflects net contribution well among honest nodes ($A' = 0.96$).

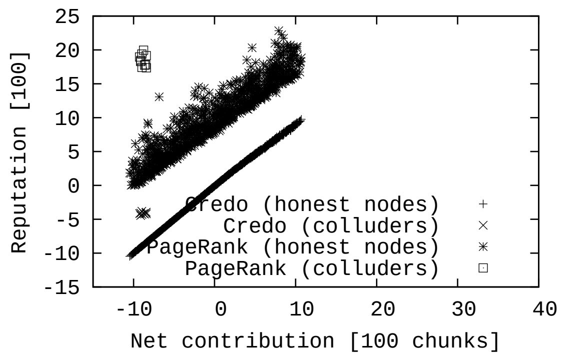


Figure 4.14: Reputation of colluding adversaries and honest nodes as a function of net contribution on synthetic workload for both Credo and eigenvector-based reputation. Credo out-performs eigenvector-based reputation in both reflecting net contribution and defending against collusion.

work. To generate workload, we create upload receipts in which the downloader is chosen randomly and the uploader is picked according to the willingness to contribute of nodes. We set the willingness to increase linearly with nodes' ID. We chose 10 adversaries who uploads at least 10 chunks among those which have the worst net contribution. Each adversary bring in 2 sybils. The attack strategy is the same as in the previous experiment. Figure 4.14 shows the reputation of honest nodes and adversaries in Credo and PageRank reputation. We scale the PageRank reputation so that it can fit to one graph with Credo reputation. Since the synthetic workload has less repeat interaction and are more uniform, both reputation reflect net contribution better than in the Maze trace. Still, Credo reputation is still better than eigenvector-based reputation ($A' = 0.998$ vs $A' = 0.898$). Credo also has better defense against collusion. The adversaries remains in bottom 30% in Credo reputation, while they can get to top 1% in PageRank reputation.

4.6.2 Experiments using Credo's prototype

4.6.2.1 Scalability of the central server

Credo central server needs to receive upload receipts from all nodes, verifies them, and compute nodes' reputation. We show that our implementation running on a 8-core machine (2.27GHz CPUs and 16GB memory) can easily handle the traffic of $260,000$ nodes in Maze network.

Bandwidth: We show that the bandwidth required for the central server to receive upload receipts is modest. The aggregate traffic among Maze nodes in 2 weeks is 280TB. A fraction of this data traffic is between leechers only which does not result in upload receipts nor receipt upload traffic to the central server. In typical public and private BitTorrent communities, $> 80\%$ the traffic are between seeders and leechers [44]. Even if all of the traffic are upload from seeders to leechers, the central server receives only 1.4GB of uploaded receipts per day (each receipt is 70-byte in size and captures a 1MB chunk transfer). This means the central server only needs a modest download capacity of 130Kbps to receive upload receipts.

Storage: After verifying an upload receipt, the central server stores it as a triple $\langle SHA1(data), uploaderID, downloaderID \rangle$ which requires 28 bytes on disk. Therefore, it requires 8GB disk space to store 2 weeks upload activities in order to compute reputation.

CPU: The central server needs to verify a fraction of upload receipts it receives. Our implementation can verify 400 receipts per second using a single thread, i.e. 35 millions receipts per day. This means that a single threaded verifier can handle data traffic that is 17.5 times bigger than observed in the Maze system.

The central server also needs to compute the reputation score for every node. Our current implementation takes 1 hour for the central server to process the upload receipts corresponding to 2 weeks traffic in the Maze system. Our default period for updating reputation is 4 hours. Within that amount of time, our central server can compute reputation for a network of $1,000,000$ nodes whose traffic demand is similar to that observed in the Maze system.

4.6.2.2 Deployment on Planet lab

To examine the real performance benefits when nodes are incentivized to contribute, we compare between two scenarios:

1. Every node runs the Credo client implementation Nodes are incentivized to stay online, and serve other nodes in order to gain reputation after downloading a file.

2. Every node runs the original Azureus client implementation. Since there is no incentive to stay online after downloading a file, nodes go offline immediately after finishing downloading the file.

We experiment with both scenarios on 210 PlanetLab nodes. We inject a file to one seeder at the beginning. We set the file size to 25MB to make the experiment finish in a reasonable time ($<$ 2 hours). Other nodes arrive to download the file once at a time every 15 second. We set the application limit throughput using the distribution in [56]. The download throughput limit is 5 times larger than upload throughput limit for every node, in order to capture the asymmetry of upload and download throughput in wide area network.

Figure 4.15 plots the cumulative distribution of complete download time for each node in both scenario. We observe that both the average as well as the median download time improve significantly when nodes are incentivized to stay online in scenario 1. The

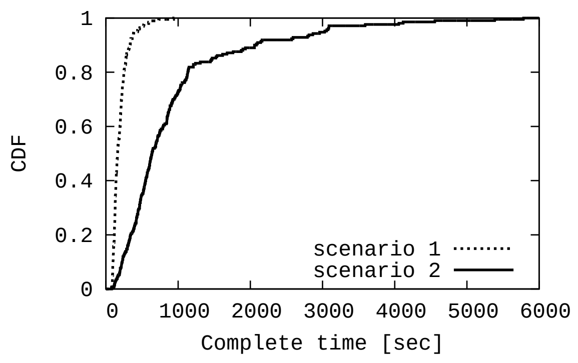


Figure 4.15: Cumulative distribution of download time two scenario: 1) nodes are incentivized by Credo protocol to stay online and continue upload to others after downloading a file, 2) nodes go offline immediately after getting a file. The average and median download time in scenario 1 are significantly smaller than that of scenario 2.

average download time drops from 935 seconds (scenario 2) to 347 seconds (scenario 1). The median download time also drops from 638 seconds to 172 seconds.

This result shows that the aggregate capacity of the system improves by a factor of 2.7 when nodes are incentivized to contribute. The reason is that because the download capacity of nodes is higher than upload capacity, download capacities are always under-utilized when there is not enough seeders. There is only 1 seeder at any instance in scenario 2. On the other hand, in scenario 2, after some nodes finish downloading the file, they contribute to the aggregate seeding capacity of the system. Other nodes can utilize their high download capacities to get the file faster.

Chapter 5

Related work

5.1 Sybil-resilient admission control

Traditionally, open systems rely on a central authority who employs CAPTCHA or computational puzzles to mitigate the Sybil attack [76, 55, 57]. Unfortunately, these solutions can only limit the *rate* with which the attacker can introduce Sybil identities into the system instead of the total number of such identities. Even before the recent surge of interest in social-network-based Sybil defenses, there have been attempts at exploiting the trust graph among users to mitigate the Sybil attack: Advogato [37], Appleseed [87] and SybilProof [9] are the most well-known of these early proposals. However, it is not the goal of these protocols to perform Sybil-resilient node admission. Rather, they aim to calculate the reputation of each user/node in a way that prevents the attacker from boosting its reputation using Sybil identities. Below, we discuss recent work in node admission control and related efforts in Sybil-resilient Distributed Hash Table (DHT) routing.

SybilGuard [84] has pioneered the use of fast-mixing social networks for Sybil-resilient admission control. Using a distributed verification protocol based on random routes, SybilGuard can limit the number of Sybil nodes admitted per attack edge to $O(\sqrt{n}\log n)$. SybilLimit [83] improves this bound to admit no more than $O(\log n)$ Sybils per attack edge with high probability. Yu et al. claim that SybilLimit is nearly optimal in that its security guarantee is only a factor of $O(\log n)$ away from that of any optimal protocol.

SybilGuard and SybilLimit are both designed to work in a distributed setting where each node is initially only aware of its neighbors. By contrast, SybilInfer [18] is a centralized algorithm which assumes complete knowledge of the social graph. SybilInfer uses Bayesian inference to assign each node a probability of being a Sybil. The key observation is that, if the attacker connects more Sybils to its few attack edges, the conductance of graph including the Sybil region becomes smaller to the point that the entire graph is not fast-mixing, thereby causing the detection of the Sybil nodes. Unlike SybilGuard, SybilLimit, Gatekeeper and SumUp, SybilInfer does not consisder worst case attacks and has no analytical bound on the number of Sybil nodes admitted per attack edge. In [60], Quercia et al. propose a Sybil-defense mechanism for the mobile setting where a node collects graph information from those nodes that it has previously encountered and analyzes the partial graphs to determine the likelihood of a node being Sybil. Like SybilInfer, there is no formal bound for the algorithm in [60].

Viswanath et al. [75] has performed a comparative study of SybilGuard, SybilLimit, SybilInfer and SumUp. The study reveals two potential limitations of social-network based admission control. First, many *small* social networks (up to tens of thousands of nodes) exhibit community structure (i.e. not fast-mixing), thus causing existing protocols to falsely reject many honest nodes as Sybils. This finding suggests that Sybil-resilient admission control must be performed on large-scale social networks: the larger the graph, the better connected communities are to each other and the faster the mixing time. Thus, our evaluations use real world social graphs that consist of hundreds of thousands of nodes. Second, given a known admission controller, the attacker can strategically acquire attack edges close to the controller to gain unfair advantage. In Gatekeeper, we address this limitation by having a controller select a few random vantage points for ticket distribution. Viswanath's work compares all existing schemes in a centralized setting even though SybilGuard and SybilLimit are originally designed to work as a distributed protocol. It is worth pointing out that Sybil defense is more challenging in a distributed setting than in a centralized setting. This is because, in a centralized setting, the attacker must decide upon the graph structure of the Sybil region before the admission algorithm starts to execute. On the other hand, in a distributed setting, the attacker has the freedom to change the Sybil region of the graph arbitrarily during protocol execution to maximize its gain. More details on the comparision among different social network based Sybil defenses can be found in a recent survey [82] by Haifeng Yu.

A Sybil-resilient DHT [36, 17] ensures that DHT lookups succeed with high probability in the face of an attacker attempting to pollute the routing tables of many nodes using Sybil attacks. Danezis et al. [17] leverage the bootstrap tree of the DHT to defend against the Sybil attack. Two nodes in such a tree share an edge if one node introduced the other one into the DHT. The assumption is that Sybil nodes attach to the tree at a small number of nodes, similar to the few attack edge assumption in SybilGuard and SybilLimit. Whānau [36] uses social connections between users to build routing tables in order to perform Sybil-resilient lookups. Existing Sybil-resilient node admissions can potentially simplify the construction of distributed Sybil-resilient protocols by bounding the number of Sybil identities admitted in the first place.

All of the above social network based Sybil defenses [84, 83, 73, 72, 18, 36] do not consider privacy concern when using social network. Revealing all or a portion of the social graph can help the adversary to deanonymize the real world identity of nodes in the graph [51]. Recently, Prateek Mittal designed a Sybil-resilient DHT routing protocol, called X-Vine [48], which considers privacy concern. Unlike Whānau, messages in X-Vine only pass social links when they travel from the source to the destination. As a result, each node in the DHT only know its neighbors and a small routing table for relaying messages to its successors, thereby not being able to perform deanonymiztion.

5.2 Sybil-resilient online content voting

Ranking content is arguably one of the Web's most important problems. As users are the ultimate consumers of content, incorporating their opinions in the form of either explicit or implicit votes becomes an essential ingredient in many ranking systems. This section summarizes related work in vote-based ranking systems. Specifically, we examine how existing systems cope with Sybil attacks [19] and compare their approaches to SumUp.

5.2.1 Hyperlink-based ranking

PageRank [6] and HITS [33] are two popular ranking algorithms that exploit the implicit human judgment embedded in the hyperlink structure of web pages. A hyperlink from page A to page B can be viewed as an implicit endorsement (or vote) of page B by the creator of page A. In both algorithms, a page has a higher ranking if it is linked to by more pages with high rankings. Both PageRank and HITS are vulnerable to Sybil attacks. The attacker can significantly amplify the ranking of a page A by creating many web pages that link to each other and also to A. To mitigate this attack, the ranking system must probabilistically reset its PageRank computation from a small set of trusted web pages with probability ϵ [52]. Despite probabilistic resets, Sybil attacks can still amplify the PageRank of an attacker's page by a factor of $1/\epsilon$ [86], resulting in a big win for the attacker because ϵ is small.

5.2.2 User Reputation Systems

A user reputation system computes a reputation value for each identity in order to distinguish well-behaved identities from misbehaving ones. It is possible to use a user reputation system for vote aggregation: the voting system can either count votes only from users whose reputations are above a threshold or weigh each vote using the voter's reputation. Like SumUp, existing reputation systems mitigate attacks by exploiting two resources: the trust network among users and explicit user feedback on others' behaviors. We discuss the strengths and limitations of existing reputation systems in the context of vote aggregation and how SumUp builds upon ideas from prior work.

Feedback based reputations In EigenTrust [31] and Credence [76], each user independently computes *personalized* reputation values for all users based on past transactions or voting histories. In EigenTrust, a user increases (or decreases) another user's rating upon a good (or bad) transaction. In Credence [76], a user gives a high (or low) rating to another user if their voting records on the same set of file objects are similar (or dissimilar). Because not all pairs of users are known to each other based on direct interaction or votes on overlapping sets of objects, both Credence and EigenTrust use a PageRank-style algorithm to propagate the reputations of known users in order to calculate the reputations of unknown users. As such, both systems suffer from the same vulnerability as PageRank where an attacker can amplify the reputation of a Sybil identity by a factor of $1/\epsilon$.

Neither EigenTrust nor Credence provide provable guarantees on the damage of Sybil attacks under arbitrary attack strategies. In contrast, SumUp bounds the voting power of an attacker on a single object to be no more than the number of attack edges he possesses irrespective of the attack strategies in use. SumUp uses only negative feedback as opposed to EigenTrust and Credence that use both positive and negative feedback. Using only negative feedback has the advantage that an attacker cannot boost his attack capacity easily by casting correct votes on objects that he does not care about.

DSybil [85] is a feedback-based recommendation system that provides provable guarantees on the damages of arbitrary attack strategies. DSybil differs from SumUp in its goals. SumUp is a vote aggregation system which allows for arbitrary ranking algorithms to incorporate collected votes to rank objects. For example, the ranking algorithm can rank objects by the number of votes collected. In contrast, DSybil's recommendation algorithm is fixed: it recommends a *random* object among all objects whose sum of the weighted vote count exceeds a certain threshold.

Trust network-based reputations A number of proposals from the semantic web and peer-to-peer literature rely on the trust network between users to compute reputations [87, 37, 27, 62, 9]. Like SumUp, these proposals exploit the fact that it is difficult for an attacker to obtain many trust edges from honest users because trust links reflect offline social relationships. Of the existing work, Advogato [37], Appleseed [87] and Sybilproof [9] are resilient to Sybil attacks in the sense that an attacker cannot boost his reputation by creating a large number of Sybil identities "behind" him. Unfortunately, a Sybil-resilient user reputation scheme does not directly translate into a Sybil-resilient voting system: Advogato only computes a non-zero reputation for a small set of identities, disallowing a majority of users from being able to vote. Although an attacker cannot improve his reputation with Sybil identities in Appleseed and Sybilproof, the reputation of Sybil identities is almost as good as that of the attacker's non-Sybil accounts. Together, these reputable Sybil identities can cast many bogus votes.

5.2.3 Sybil Defense using trust networks

Many proposals use trust networks to defend against Sybil attacks in the context of different applications: SybilGuard [84] and SybilLimit [83] help a node admit another node in a decentralized system such that the admitted node is likely to be an honest node instead of a Sybil identity. Ostra [47] limits the rate of unwanted communication that adversaries can inflict on honest nodes. Sybil-resilient DHTs [17, 35] ensure that DHT routing is correct in the face of Sybil attacks. Kaleidoscope [66] distributes proxy identities to honest clients while minimizing the chances of exposing them to the censor with many Sybil identities. SumUp builds on their insights and addresses a different problem, namely, aggregating votes for online content rating. Like SybilLimit, SumUp bounds the power of attackers according to the number of attack edges. In SybilLimit, each attack edge results in $O(\log n)$ Sybil identities accepted by honest nodes. In SumUp, each attack edge leads to at most one vote with high probability. Additionally, SumUp uses user feedback on bogus votes to further reduce the attack capacity to below the number of attack edges. The feedback mechanism of SumUp is inspired by Ostra [47].

5.3 Incentivizing bandwidth contribution in P2P systems

We survey existing proposals of P2P reputation and currency systems and discuss why they do not completely address the seeder promotion problem. We also discuss existing work on catching misbehaving nodes.

Reputation systems. Reputation systems incentivize peers to upload in order to gain reputation; peers with high reputation values are promised better download performance in the future [31, 21, 41, 57]. Most existing reputation systems are graph-based, where each graph edge is formed between a pair of nodes that have had direct interactions. EigenTrust [31] uses the PageRank-style [6] propagation algorithm on the interaction graph. To reduce the chances of collusion in PageRank calculation [86], OneHop reputation [57] and multi-level tit-for-tat [41] restrict reputation propagation to one hop or a few hops. Max-flow based calculation [21, 9] can be used to defend against collusion.

Graph-based reputation schemes such as PageRank are not designed to capture a node's net contribution in the system. In particular, a node linking to another node with higher reputation achieves higher reputation gain. Thus, a strategic peer can gain unfair advantage by selectively contributing to certain peers. As a result, graph-based reputations do not satisfy the desired fairness property. Moreover, the defense against collusion of existing graph-based reputation systems is weak. For example, colluders can increase the net reputation of the collusion group by $1/\epsilon$, where ϵ is the probability of resetting a random walk, in a PageRank-style reputation computation like EigenTrust [31] or multi-level tit-for-tat [41]. In max-flow based reputation like [21, 9], each colluders get higher reputation by recruiting more nodes, which have interaction with node outside collusion group, into the collusion group. This gives the incentive for honest nodes to collude and the bigger collusion group the higher reputation they gain. However, graph-based reputation can particularly advantageous in a live streaming environment where nodes with different capacities should be incentivized to position themselves in different positions in the underlying stream distribution graph, as is done in the Contracts system [58].

Currency systems. Currency systems must maintain system liquidity according to the current demand and hoarding levels [32]. Karma [74] scales currencies based on the number of active nodes and Antfarm [55] adjusts the amount of tokens according to the number of active nodes and the number of active swarms.

Currency systems such as Dandelion [65], BitStore [61], PACE [11], Antfarm [55] and others [79, 74] incentivize peers to upload to others in exchange for tokens that entitle them to future downloads. The tokens can be directly transferable among peers [79], reclaimed by the central party upon each use [55, 79], or they can completely reside at the central party link in Dandelion [65]. Credo is not a currency system: although it uses the concept of credit transfer, it is only using credits to calculate each node's reputation score. Existing currency system proposals enforce the strict "download as much as you upload" policy where a node with no currency is not allowed to download. Compared to our notion of fairness, the policy enforced by currency systems is less desirable: since all peers achieve the same download speed as long as they have non-zero currency tokens, a peer has no incentives to contribute more than what is necessary to satisfy its own demand. Moreover, currency systems face the daunting challenge of maintaining monetary supply according to current demand and hoarding levels at all times [32] to avoid undesired inflations or the bankruptcy of many nodes.

Some currency proposals such as Antfarm [55] and PACE [11] also have the additional goal of improving the global efficiency of content distribution. Antfarm achieves optimal seeding capacity allocation via central management and PACE relies on peers to set the right download prices in a currency market. Improving content distribution efficiency is an orthogonal goal to providing uploading incentives.

Detecting and punishing deviant behaviors. It is not enough to just incentivize contribution, we also need disincentives to discourage peers from cheating. BAR Gossip [40] and FlightPath [39] introduce the idea of centralized punishments based on cryptographic proof-of-misbehavior and use it to ensure the fair exchange of data within a single swarm. In the context of peer-to-peer storage systems, Ngan et al. [53] and Samsara [15] rely on verifiable records and periodic auditing to check that peers indeed store data as they claim. SHARP [24] also audits to ensure that an autonomous system complies with its resource contract.

Credo does not need general purpose auditing because credits are only used to calculate a peer's reputation instead of serving as resource claims. We borrow the idea of cryptographic proof-of-misbehavior from BAR Gossip [40] to detect and penalize misbehaving nodes.

Chapter 6

Conclusion

This thesis provides robust identity managements which are resilient to Sybil attacks, and illustrates how to use them to provide security and incentive for cooperative systems in various contexts. The main theme of this work is to leverage the social network among users in designing secure and incentive-compatible cooperative systems. In particular, we develop a distributed admission control protocol, called Gatekeeper, that leverages social network to admit most honest user identities and only few Sybil identities into the systems. Gatekeeper's defense is optimal for the case of $O(1)$ attack edges and admits only $O(1)$ Sybil identities (with high probability). In the face of $O(k)$ attack edges (for any $k \in O(n/\log n)$), Gatekeeper admits $O(\log k)$ Sybils per attack edge. Gatekeeper can be used as a robust identity management for both centralized and decentralized cooperative systems. In the context of content voting, we provide a vote aggregation system, called SumUp, that leverages unique properties of content voting systems to provide significantly better Sybil defense compared with applying the general Gatekeeper admission control. SumUp can limit the number of bogus votes cast by adversaries to no more than the number of attack edges (with high probability), which is an order of magnitude better than applying general admission control protocols to the content voting context. Using user feedback on votes, SumUp further restricts the voting power of adversaries who continuously misbehave to below the number of their attack edges. We applied SumUp on the voting trace of Digg and have found strong evidence of attack on many articles marked “popular” by Digg. The defense of both SumUp and Gatekeeper are currently the state-of-the-art in their respective problems as reported in a recent survey [82]. Finally, we provide a robust reputation system, called Credo, that is resilient to both Sybil and collusion attacks, and can be used to incentivize bandwidth contribution in peer-to-peer content distribution networks.

6.1 Future work

We believe that the work in this thesis is only an early contribution in the research area of security and privacy for cooperative systems. The importance of cooperative systems will continue to grow in the future. There are increasingly more attractive domains for adversaries to exploit. We expect to push the limit of existing designs and to contribute new techniques in this area. Below are potential future research projects.

Combating information censorship: Recent events during the so-called ”Arab Spring” have shown the power of the Internet when it comes to organizing large groups of people for protests. Such events have also shown how willing repressive governments are to censor the Internet or to disconnect their populace from the Internet entirely such as the incidents in Egypt and Libya in early 2011. We believe that during these government-imposed communication blackout periods, people should still be able to exchange information and organize among themselves. Our goal is to design and build a networked system to help the citizens in such countries to disseminate information during such blackouts in the presence of an adversarial government.

One promising solution is that the citizens in these countries form a cooperative system in which citizens use their mobile devices to exchange messages with nearby devices through WiFi during communication blackout. This opportunistic communication is promising since it is hard for the adversarial government to block it. However, the adversarial government can still create Sybil identities and use them to flood the system with bogus (junk) messages, or confuse the citizens by generating dishonest votes on authentic messages. The defense provided by SumUp can potentially prevent this threat. However, SumUp requires each citizen to know the entire social graph in order to filter out bogus messages. This raises privacy concerns since the government can perform deanonymization to ascertain the identity of the protesters, and thus to persecute them. We are designing a enhanced version of SumUp so that it does not require the knowledge of the entire social graph, thereby addressing this privacy concern.

Reputation-based routing: Secure routing protocols that do not rely on a public-key infrastructure are desirable because of the relative ease of deployment. For example, deploying a PKI for inter-domain routing over the Internet is a serious challenge. In the BGP setting, we can view the network of autonomous systems (AS) as a cooperative system in which the ASes are sharing the view of the network through exchanging routing tables. The relationships among ASes can also be viewed as a trust graph where the adversary has only a few links with other ASes. However, the adversary can generate and propagate a large amount of bogus routing information in order to hijack traffic. Our goal is to minimize the number of incorrectly computed routes by honest ASes due to bogus routing information. A promising solution that we are investigating right now is to embed a decentralized reputation mechanism into existing routing protocols such as path-vector and link state protocols to reduce the risk of choosing routes via the adversary.

Appendix A

Proof for Theorem 2.1

Theorem A.1. *Suppose the graph $\mathcal{G}$ is a fixed degree sequence random graph constructed using the pairing method. For any ticket source u, the expected number of tickets that u disseminates to reach $n/2$ nodes is $E[A_0] = \Theta(n)$.*

Instead of directly proving this theorem, we will prove that when a ticket source disseminate n tickets, the expected number of reachable nodes is at least $\frac{\alpha}{2 \cdot d^2 \cdot (1+\alpha)} n$. This mean a ticket source needs to distribute $\Theta(n)$ tickets to reach $\Theta(n)$ nodes.

Proof. Consider a ticket source u that disseminates $A_0 = n$ tickets. We compute how many nodes are reachable by bounding the number of tickets that are dropped at each level. Let L_i be the number of nodes at level i from u and $S_i = L_0 + L_1 + ... + L_i$. Let q_i be the fraction of dead-end nodes at level i. A dead-end node is a node that does not have a neighbor at a distance further away from the ticket source, i.e. after consuming one ticket, a dead-end nodes will drop all the remaining tickets. Note that both L_i and q_i are random variables. Let C_i denote a configuration of all the nodes and edges from level 0 to i. Similarly, C'_{i+1} is the configuration of all the nodes end edges at levels greater than or equal to $i + 1$. Now, consider the case where we fix the configuration

$C_i = c$ and $C'_{i+1} = c'$ and only consider different variations in the edge configuration between level i and $i+1$. In this case, A_i, S_i, q_{i+1}, L_{i+1} are fixed. Let D_{i+1} be the number of tickets that are dropped at level $i+1$. We want to bound D_{i+1}. Tickets only get dropped by dead-end nodes. Let x denote the number of edges from level i to dead-end nodes at level $i+1$ and y denote the number of edges from level i to non-dead-end nodes at level $i+1$. Among the $x+y$ edges from level i, the dead-end nodes at level i can pick randomly any x edges to connect in a random graph. Therefore, the expected number of tickets go to the dead-end nodes is $A_i \frac{x}{x+y}$. We know that $x \leq d \cdot q_{i+1} L_{i+1}$ and $y \geq (1 - q_{i+1}) L_{i+1}$. Therefore,

$$\begin{aligned} E[D_{i+1}|C_i = c|C'_{i+1} = c'] &\leq A_i \frac{d \cdot q_{i+1}}{1 + (d-1) q_{i+1}} \\ E[D_{i+1}|C_i = c|C'_{i+1} = c'] &\leq A_i (d \cdot q_{i+1}) \end{aligned}$$

By varying the configuration C'_{i+1}, we have:

$$E[D_{i+1}|C_i = c] \leq A_i \cdot d \cdot E[q_{i+1}|C_i = c] \tag{A.1}$$

Next we bound $E[q_{i+1}|C_i = c]$. For a fixed configuration $C_i = c$, we need to construct a fixed degree random graph for $n - S_{i-1}$ nodes for levels greater than or equal to i. Given a random node v in the set of $n - S_i$ nodes that have distance farther than i; $d(v)$ is the degree of the node v. Let $p(v)$ denote the probability of one of the edges of v pointing to nodes at level i. We have $p(v) < \frac{L_i d}{n - S_{i-1}} < \frac{d \cdot S_i}{n}$. Now we calculate the conditional probability $\frac{Pr(\text{all edges of v point to nodes at level i})}{Pr(\text{at least one edge of v points to node at level i})}$. This conditional probability can be written as: $\frac{p(v)^{d(v)}}{1-(1-p(v))^{d(v)}} \leq p(v)$. Hence, we have

$E[q_{i+1}|C_i = c] \leq d\frac{S_i}{n}$.

So, from Equation A.1, we get:

$$\begin{aligned} E[D_{i+1}|C_i = c] &\leq A_i \cdot d^2 \frac{S_i}{n} \\ E[D_{i+1}|C_i = c] &\leq A_0 \cdot d^2 \frac{S_i}{n} \\ E[D_{i+1}] &\leq A_0 \cdot d^2 \frac{E[S_i]}{n} \end{aligned}$$

This formula states that the expected number of tickets exponentially decreases when the distance to the ticket source gets smaller. Let δ denote the level where $E[S_\delta] = \frac{\alpha}{2 \cdot d^2 \cdot (1+\alpha)} n$. The expected number of tickets that are dropped or consumed after reaching the level δ is at most:

$$\begin{aligned} &< A_0 \frac{d^2}{n}(E[S_0] + \cdots E[S_\delta]) + E[S_\delta] \\ &< A_0 \frac{d^2}{n} \frac{\alpha + 1}{\alpha} E[S_\delta] + E[S_\delta] \\ &< n \end{aligned}$$

Hence, disseminating n tickets is enough to reach $\frac{\alpha}{2 \cdot d^2 \cdot (1+\alpha)} n$ in expectation. Based on this bound, we can change Gatekeeper so that each ticket source scale the number of tickets by a constant factor to reach $n/2$ nodes. Hence, in expectation, we can show that a random source needs to to distribute a $\Theta(n)$ tickets to reach $n/2$ nodes.

$\square$

Bibliography

[1] Digg. http://www.digg.com.

[2] ALON, N., AND SPENCER, J. H. *The probabilistic method*. John Wiley and Sons, 2008.

[3] BADEN, R., SPRING, N., AND BHATTACHARJEE, B. Identifying close friends on the Internet. In *HotNets* (2009).

[4] BEATCAPTCHAS.COM. Beatcaptchas.com. http://www.beatcaptchas.com/prices.html.

[5] BOLLOBAS, B. *Random Graphs*. Cambridge University Press, 2001.

[6] BRIN, S., AND PAGE, L. The anatomy of a large-scale hypertextual web search engine. In *WWW* (1998).

[7] CAPTCHA. Captcha. http://www.captcha.net/.

[8] CASTRO, M., AND LISKOV, B. Practical byzantine fault tolerance. In *Proceedings of the third symposium on Operating systems design and implementation* (Berkeley, CA, USA, 1999), OSDI '99, USENIX Association, pp. 173–186.

[9] CHENG, A., AND FRIEDMAN, E. Sybilproof reputation mechanisms. In *P2PECON '05: Proceedings of the 2005 ACM SIGCOMM workshop on Economics of peer-to-peer systems* (New York, NY, USA, 2005), ACM.

[10] CHOW, S. S. M. Running on Karma - P2P reputation and currency systems. In *CANS* (2007), pp. 146–158.

[11] CHRISTINA APERJIS, M. J. F., AND JOHARI, R. Peer-assisted content distribution with prices. In *CoNext'08: Proc. ACM SIGCOMM Conference on emerging Networking EXperiments* (2008).

[12] Clarke, I., Sandberg, O., Wiley, B., and Hong, T. W. Freenet: A distributed anonymous information storage and retrieval system. In *INTERNATIONAL WORKSHOP ON DESIGNING PRIVACY ENHANCING TECHNOLOGIES: DESIGN ISSUES IN ANONYMITY AND UNOBSERVABILITY* (2001), Springer-Verlag New York, Inc., pp. 46–66.

[13] Cohen, B. Incentives build robustness in bittorrent. In *Economics of Peer-to-Peer Systems* (2003).

[14] Cormen, T., Leiserson, C., and Rivest, R. *Introduction to Algorithms*. The MIT Press, 1985.

[15] Cox, L. P., and Noble, B. Samsara: Honor among theives in peer-to-peer storage. In *19th ACM Symposium on Operating Systems Principles (SOSP)* (2003).

[16] Dabek, F., Kaashoek, M. F., Karger, D., Morris, R., and Stoica, I. Wide-area cooperative storage with cfs. In *Proceedings of the eighteenth ACM symposium on Operating systems principles* (New York, NY, USA, 2001), SOSP '01, ACM, pp. 202–215.

[17] Danezis, G., Lesniewski-Laas, C., Kaashoek, M. F., and Anderson, R. Sybil-resistant DHT routing. In *European Symposium On Research In Computer Security* (2008).

[18] Danezis, G., and Mittal, P. SybilInfer: Detecting sybil nodes using social networks. In *NDSS* (2009).

[19] Douceur, J. The sybil attack. In *1st International Workshop on Peer-to-Peer Systems* (2002).

[20] Dunbar, R. *How Many Friends Does One Person Need?: Dunbar's Number and Other Evolutionary Quirks*. Harvard University Press, 2010.

[21] Feldman, M., Lai, K., Stoica, I., and Chuang, J. Robust incentive techniques for peer-to-peer networks. In *Proceedings of Electric Commerce* (2004).

[22] Feng, Q., and Dai, Y. Lip: A lifetime and popularity based ranking approach to filter out fake files in p2p file sharing systems. In *IPTPS* (2007).

[23] FILENetworks, T. Vuze and deluge to be banned on what.cd and waffles.fm. http://filenetworks.blogspot.com/2010/02/vuze-and-deluge-to-be-banned-on-whatcd. html.

[24] Fu, Y., Chase, J. S., Chun, B., Schwab, S., and Vahdat, A. In *Proceedings of the 19th ACM Symposium on Operating System Principles (SOSP)* (Oct. 2003).

[25] GEAMBASU, R., KOHNO, T., LEVY, A., AND LEVY, H. M. Vanish: Increasing data privacy with self-destructing data. In *Proc. of the 18th USENIX Security Symposium* (2009).

[26] GERTNER, Y., GOLDWASSER, S., AND MALKIN, T. A random server model for private information retrieval or information theoretic pir avoiding database replication. Springer, pp. 200–217.

[27] GUHA, R., KUMAR, R., RAGHAVAN, P., AND TOMKINS, A. Propagation of trust and distrust. In *WWW* (2004).

[28] HSIEH, H. Doonesbury online poll hacked in favor of MIT. In *MIT Tech* (2006).

[29] HU, Y., PERRIG, A., AND SIRBU, M. SPV: Secure path vector routing for securing BGP. In *Proc. of ACM SIGCOMM* (2004).

[30] INVESPBLOG. An interview with Digg top user, 2008. http://www.invesp.com/blog/social-media/an-interview-with-digg-top-user.html.

[31] KAMVAR, S. D., SCHLOSSER, M. T., AND GARCIA-MOLINA, H. The eigentrust algorithm for reputation management in p2p networks. In *WWW '03: Proceedings of the 12th international conference on World Wide Web* (New York, NY, USA, 2003), ACM, pp. 640–651.

[32] KASH, I., FRIEDMAN, E., AND HALPERN, J. Optimizing scrip systems: Efficiency, crashes, hoarders, and altruists. In *ACM Conference on Electronic Commerce* (2007).

[33] KLEINBERG, J. Authoritative sources in a hyperlinked environment. In *Proc. 9th ACM-SIAM Symposium on Discrete Algorithms* (1998).

[34] LESKOVEC, J., LANG, K., DASGUPTA, A., AND MAHONEY, M. W. Statistical properties of community structure in large social and information networks. In *7th international conference on WWW* (2008).

[35] LESNIEWSKI-LAAS, C. A sybil-proof one-hop dht. In *1st Workshop on Social Network Systems* (2008).

[36] LESNIEWSKI-LAAS, C., AND KAASHOEK, M. F. Whānau: A Sybil-proof distributed hash table. In *NSDI'10: Proceedings of the 7th USENIX Symposium on Networked Systems Design and Implementation* (2010), USENIX Association.

[37] LEVIEN, R., AND AIKEN, A. Attack-resistant trust metrics for public key certification. In *SSYM'98: Proceedings of the 7th conference on USENIX Security Symposium, 1998* (Berkeley, CA, USA, 1998), USENIX Association, pp. 18–18.

[38] LEVIN, D., LACURTS, K., SPRING, N., AND BHATTACHARJEE, B. Bittorrent is an auction: analyzing and improving bittorrent's incentives. In *SIGCOMM '08: Proceedings of the ACM SIGCOMM 2008 conference on Data communication* (New York, NY, USA, 2008), ACM, pp. 243–254.

[39] LI, H., CLEMENT, A., MARCHETTI, M., KAPRITSOS, M., ROBISON, L., ALVISI, L., AND DAHLIN, M. Flightpath: Obedience vs. choice in cooperative services. In *Proceedings of the 8th Operating Systems Design and Implementation (OSDI)* (2008).

[40] LI, H., CLEMENT, A., WONG, E., NAPPER, J., ROY, I., ALVISI, L., AND DAHLIN, M. Bar gossip. In *Proceedings of the 7th Operating System Design and Implementation (OSDI)* (2006).

[41] LIAN, Q., PENG, Y., YANG, M., ZHANG, Z., DAI, Y., AND LI, X. Robust incentives via multi-level tit-for-tat. In *Proceedings of IPTPS* (2006).

[42] LIANG, J., KUMAR, R., XI, Y., AND ROSS, K. Pollution in p2p file sharing systems. In *IEEE Infocom* (2005).

[43] MCKAY, B. D., AND WORMALD, N. C. Uniform generation of random regular graphs of moderate degree. Journal of Algorithms.

[44] MEULPOLDER, J., ACUNTO, L. D., CAPOTA, M., WOJCIECHOWSKI, M., POUWELSE, J., EPEMA, D., AND SIPS, H. Public and private bittorrent communities: A measurement study. In *IPTPS* (2010).

[45] MISLOVE, A., MARCON, M., GUMMADI, K., DRUSCHEL, P., AND BHATTACHARJEE, S. Measurement and analysis of online social networks. In *7th Usenix/ACM SIGCOMM Internet Measurement Conference (IMC)* (2007).

[46] MISLOVE, A., MARCON, M., GUMMADI, K. P., DRUSCHEL, P., AND BHATTACHARJEE, B. Measurement and analysis of online social networks. In *IMC '07: Proceedings of the 7th ACM SIGCOMM conference on Internet measurement* (New York, NY, USA, 2007), ACM, pp. 29–42.

[47] MISLOVE, A., POST, A., DRUSCHEL, P., AND GUMMADI, K. P. Ostra: Leveraging trust to thwart unwanted communication. In *NSDI'08: Proceedings of the 5th conference on 5th Symposium on Networked Systems Design & Implementation* (2008).

[48] MITTAL, P., CAESAR, M., AND BORISOV, N. X-vine: Secure and pseudonymous routing using social networks. In *In Proc. of NDSS* (2012).

[49] MITZENMACHER, M., AND UPFAL, E. *Probability and Computing*. Cambridge Press, 2005.

[50] MOTWANI, R., AND RAGHAVAN, P. *Randomized Algorithms*. Cambridge University Press, 1995.

[51] NARAYANAN, A., AND SHMATIKOV, V. De-anonymizing social networks. In *Proceedings of the 2009 30th IEEE Symposium on Security and Privacy* (Washington, DC, USA, 2009), SP '09, IEEE Computer Society, pp. 173–187.

[52] NG, A., ZHENG, A., AND JORDAN, M. Link analysis, eigenvectors and stability. In *International Joint Conference on Artificial Intelligence (IJCAI)* (2001).

[53] NGAN, T.-W., WALLACH, D., AND DRUSCHEL, P. Enforcing fair sharing of peer-to-peer resources. In *Proceedings of 5th IPTPS* (2003).

[54] NOAM NISSAN, T. R., AND TARDOS, E. *Algorithmic Game Theory*. Cambridge Press, 2007.

[55] PETERSON, R. S., AND SIRER, E. G. AntFarm: Efficient content distribution with managed swarms. In *Proceedings of the 6th conference on Networked Systems Design & Implementation (NSDI)* (2009), USENIX Association, pp. 1–1.

[56] PIATEK, M., ISDAL, T., ANDERSON, T., KRISHNAMURTHY, A., AND VENKATARAMANI, A. Do incentives build robustness in bittorrent? In *NSDI* (2007).

[57] PIATEK, M., ISDAL, T., KRISHNAMURTHY, A., AND ANDERSON, T. One hop reputations for peer to peer file sharing workloads. In *NSDI'08: Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation* (2008), USENIX Association, pp. 1–14.

[58] PIATEK, M., KRISHNAMURTHY, A., VENKATARAMANI, A., YANG, R., AND ZHANG, D. Contracts: Practical contribution incentives for p2p live streaming. In *NSDI'10: Proceedings of the 7th USENIX Symposium on Networked Systems Design and Implementation* (2010).

[59] POST, A., SHAH, V., AND MISLOVE, A. Bazaar: Strengthening user reputations in online marketplaces. In *Proceedings of the 8th Symposium on Networked Systems Design and Implementation (NSDI'11)* (Boston, MA, March 2011).

[60] QUERCIA, D., AND HAILES, S. Sybil attacks against mobile users: Friends and foes to the rescue. In *IEEE INFOCOM* (2010).

[61] RAMACHANDRAN, A., DAS SARMA, A., AND FEAMSTER, N. Bitstore: An incentive-compatible solution for blocked downloads in bittorrent. In *Proc. Joint Workshop on The Economics of Networked Systems and Incentive-Based Computing (NetEcon)* (2007).

[62] RICHARDSON, M., AGRAWAL, R., AND DOMINGOS, P. Trust management for the semantic web. In *Proceedings of the 2nd Semantic Web Conference* (2003).

[63] RILEY, D. Techcrunch: Stat gaming services come to youtube, 2007.

[64] SIRIVIANOS, M., KIM, K., AND YANG, X. Introducing Social Trust to Collaborative Spam Mitigation. In *INFOCOM* (2011).

[65] SIRIVIANOS, M., PARK, J. H., YANG, X., AND JARECKI, S. Dandelion: Cooperative content distribution with robust incentives. In *USENIX Annual Technical Conference* (2007).

[66] SOVRAN, Y., LIBONATI, A., AND LI, J. Pass it on: Social networks stymie censors. In *Proc. of the 7th International Workshop on Peer-to-Peer Systems (IPTPS)* (Feb 2008).

[67] STEIN, T. Personal communication with tao stein, a facbook engineer.

[68] TOIVONEN, R., ONNELA, J.-P., SARAMÄKI, J., HYVÖNEN, J., AND KASKI, K. A model for social networks. *Physica A Statistical Mechanics and its Applications 371* (2006), 851–860.

[69] TORRENTFREAK. Bittorrent still dominates global internet traffic, 2010. http://torrentfreak.com/bittorrent-still-dominates-global-internet-traffic-101026/.

[70] TRAN, N., CHIANG, F., AND LI, J. Efficient cooperative backup with decentralized trust management. *To appear in ACM Transaction on Storage (TOS)* (2012).

[71] TRAN, N., LI, J., AND SUBRAMANIAN, L. Netecon '10: Proceedings of the 5th international workshop on economics of networked systems. ACM.

[72] TRAN, N., LI, J., SUBRAMANIAN, L., AND CHOW, S. S. Optimal sybil-resilient node admission control. In *The 30th IEEE International Conference on Computer Communications (INFOCOM 2011)* (Shanghai, P.R. China, 4 2011).

[73] TRAN, N., MIN, B., LI, J., AND SUBRAMANIAN, L. Sybil-resilient online content voting. In *NSDI'09: Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation* (2009), USENIX Association, pp. 15–28.

[74] VISHNUMURTHY, V., CHANDRAKUMAR, S., AND SIRER, E. Karma: A secure economic framework for peer-to-peer resource sharing. In *P2P-ECON* (2003).

[75] VISWANATH, B., POST, A., GUMMADI, K., AND MISLOVE, A. An analysis of social network-based sybil defenses. In *SIGCOMM* (2010).

[76] WALSH, K., AND SIRER, E. G. Experience with an object reputation system for peer-to-peer file-sharing. In *NSDI'06: Proceedings of the 3rd conference on 3rd Symposium on Networked Systems Design & Implementation* (2006), USENIX Association, pp. 1–1.

[77] WILSON, C., BOE, B., SALA, A., PUTTASWAMY, K., AND ZHAO, B. User interactions in social networks and their implications. In *Proceedings of ACM EuroSys* (2009).

[78] WOLCHOK, S., HOFMANN, O. S., HENINGER, N., FELTEN, E. W., HALDERMAN, J. A., ROSSBACH, C. J., WATERS, B., AND WITCHEL, E. Defeating vanish with lowcost sybil attacks against large dhts. In *In Proc. of NDSS* (2010).

[79] YANG, B., AND GARCIA-MOLINA, H. Ppay: micropayments for peer-to-peer systems. In *CCS '03: Proceedings of the 10th ACM conference on Computer and communications security* (New York, NY, USA, 2003), ACM, pp. 300–310.

[80] YANG, M., CHEN, H., ZHAO, B. Y., DAI, Y., AND ZHANG, Z. Deployment of a large-scale peer-to-peer social network. In *USENIX WORLDS* (2004).

[81] YANG, Z., WILSON, C., WANG, X., GAO, T., ZHAO, B. Y., AND DAI, Y. Uncovering social network sybils in the wild. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference* (New York, NY, USA, 2011), IMC '11, ACM, pp. 259–268.

[82] YU, H. Sybil defenses via social networks: a tutorial and survey. *SIGACT News 42* (October 2011), 80–101.

[83] YU, H., GIBBONS, P., KAMINSKY, M., AND XIAO, F. SybilLimit: A near-optimal social network defense against Sybil attacks. In *IEEE Symposium on Security and Privacy* (2008), IEEE Comoputer Society, pp. 3–17.

[84] YU, H., KAMINSKY, M., GIBBONS, P. B., AND FLAXMAN, A. Sybilguard: defending against sybil attacks via social networks. In *SIGCOMM '06: Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications* (New York, NY, USA, 2006), ACM, pp. 267–278.

[85] YU, H., SHI, C., KAMINSKY, M., GIBBONS, P. B., AND XIAO, F. DSybil: Optimal Sybil-resistance for recommendation systems. In *IEEE Security and Privacy* (2009).

[86] ZHANG, H., GOEL, A., GOVINDAN, R., AND MASON, K. Making eigenvector-based reputation systems robust to collusions. In *Proc. of the Third Workshop on Algorithms and Models for the Web Graph* (2004).

[87] ZIEGLER, C.-N., AND LAUSEN, G. Propagation models for trust and distrust in social networks. *Information Systems Frontiers 7*, 4-5 (2005), 337–358.