Sybil-Resilient Online Content Rating

Dinh Nguyen Tran, Bonan Min, Jinyang Li, Lakshminarayanan Subramanian
Courant Institute of Mathematical Sciences
New York University
Email:$\{$*trandinh,bonan,jinyang,lakshmi*$\}$*@cs.nyu.edu*

Abstract

Obtaining user feedback (using votes) is essential in ranking user-generated online content. However, any online voting system is susceptible to the Sybil attack where adversaries can out-vote real users by creating several Sybil identities. In this paper, we present *SumUp*, a Sybil-resilient online content rating system that leverages trust networks among users to defend against Sybil attacks with strong security guarantees. SumUp addresses the basic *vote aggregation problem* of how to aggregate votes from different users in a trust network in the face of Sybil identities casting an arbitrarily large number of bogus votes. By using the technique of *adaptive vote flow* aggregation, SumUp can significantly limit the number of bogus votes cast by adversaries to no more than the number of attack edges in the trust network (with high probability). SumUp leverages user voting history to further restrict the voting power of adversaries who continuously misbehave to below the attack edges. Using detailed evaluation of several existing social networks (Digg, YouTube, Flickr, LiveJournal), we show SumUp's ability to handle Sybil attack. By applying SumUp on the voting trace of Digg (online news voting site), we have detected strong evidence of attack on many articles marked "popular" by Digg.

1 Introduction

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. In the past few years, there have been several incidents where user-content hosting sites have been contaminated with bogus content from adversarial users resulting in huge monetary losses [2, 4]. The survival and the popularity of user-content hosting sites is largely dependent on their ability to rate content quality and detect bogus content. The problem of mislabeled or low quality content also exists in peer-to-peer systems due to its open nature of allowing any node to publish content [9, 18, 27].

People have long realized the importance of ranking online content. For well-linked hypertext files, algorithms such as PageRank [3], HITS [14] exploit the underlying link structure for ranking. However, many forms of online content such as videos, news articles, reviews and document scans have no well defined links to each other. For these forms of content, the best source of ranking information comes from the explicit votes of users themselves. In fact, given the diversity of user content, obtaining explicit votes from users has remained an essential component for ranking content quality. A large fraction of existing user-content hosting sites such as news (Digg, Reddit), videos (YouTube), documents (Scribd), consumer reviews (Yelp, Amazon) rely on user votes to rank content.

A fundamental problem with any user-based content rating system is 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, the famous Slashdot poll on the best computer science school motivated students to deploy automatic scripts to vote for their schools repeatedly [10]. Some companies also advertise services that help clients promote their content to the top spot on popular sites such as YouTube by voting from a large number of Sybil accounts [25].

In this paper, we present SumUp, a Sybil-resilient online content rating system that prevents adversaries from arbitrarily distorting voting results. Like other proposals [7, 16, 20, 24, 29], SumUp leverages the trust relationships that already exist among users (e.g. in the form of social relationships) by exploiting the fact that it takes some human efforts for a user to become trusted by another user. Hence, it is difficult for the attacker to obtain a large number of trust links from honest users, but he may create many links among Sybil identities themselves.

The concept of using trust networks to defend against Sybil attacks is not new [5, 17, 20, 28, 29, 31]. For example, SybilLimit [28] uses random routing in social networks to reduce the number of accepted Sybil identities to $O(\log n)$ per attack edge (edge from a honest node to an adversary node) among n honest identities. Ostra [20] uses user history on top of a social network to thwart unwanted communication. While these prior works are related in spirit, they do not directly address the online content rating problem in this paper.

SumUp addresses the *vote aggregation problem* which can be stated as follows: *Given that m users in a trust network vote on the quality of a specific online content, of which an arbitrary fraction of votes may be from Sybil identities created by an attacker, how do we aggregate votes in a Sybil resilient manner?* An ideal solution for the vote aggregation problem should satisfy three properties. First, if all m votes are from honest users, the solution should aggregate almost all votes. Second, if the attacker has e_A attack edges (from honest users), 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 is diminished.

SumUp offers strong security guarantees for addressing the vote aggregation problem in the face of Sybil attacks. In particular, SumUp achieves the three aforementioned properties of an ideal solution with high probability. 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.

SumUp collects votes from a trusted source by computing a set of max-flow paths on the trust graph from the source to all voters. Because only votes on paths with non-zero flows are counted, the number of bogus votes collected is limited by the number of attack edges instead of links among Sybil identities. SumUp uses a tunable parameter, C_{max}, that dictates the maximum number of votes that can be aggregated by the system. SumUp assigns link capacities appropriately based on C_{max} so that if the number of honest voters is less than C_{max}, then SumUp can aggregate a large fraction of their votes (more than 80%). SumUp also probabilistically limits the number of bogus votes cast by the attacker to no more than the number of attack edges. By leveraging user voting history, SumUp can further restrict the power of the attacker who continuously misbehaves to below his attack edges.

Using a detailed evaluation of several existing social networks (Digg, YouTube, Flickr, LiveJournal), we show that SumUp successfully limits the number of bogus votes by the attacker to the number of attack edges and is also able to collect $> 80\%$ of votes from C_{max} honest voters. By applying SumUp on the voting trace and social network of Digg (online news voting site), we have detected hundreds of suspicious articles that have been marked "popular" by Digg. Based on manual sampling, we believe that at least 50% of suspicious articles found by SumUp have a strong evidence of Sybil attacks.

The paper is organized as follows. In Section 2, we discuss related work and in Section 3 we define the system model and the vote aggregation problem. Section 4 outlines the overall approach of SumUp and Sections 5 and 6 present the detailed design. In Section 7, we describes our evaluation results. Finally in Section 8, we discuss specific issues and conclude in Section 9.

2 Related Work

Ranking content is arguable one of Web's most important problems. Ideally, the quality of a piece of content should be judged based on information contained in the content. Unfortunately, there is no general machine learning-based method to determine the quality of such diverse content as images, videos and news articles etc. As a result, incorporating users' opinions in the form of either explicit or implicit votes becomes essential for online content rating. This section summarizes related work in vote-based ranking systems. Specifically, we examine how existing systems cope with Sybil attacks [8] and compare their approaches to SumUp.

2.1 Hyperlink-based ranking

PageRank [3] and HITS [14] 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 higher ranking if it is linked to by more pages with high rankings.

The original PageRank is vulnerable to Sybil attack. The attacker can significantly amplify the ranking of page A by creating many web pages that link to each other and also to A. The recommended defense mechanism is to reset PageRank computation to a small set of trusted web pages with a small probability ϵ [22]. However, since ϵ is small, the attacker can still achieve a big win [30].

2.2 User Reputation-based ranking

A user reputation system computes a rating 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 weight each vote using the user's reputation. There are two general types of reputation systems: those that compute reputation using user history and those that use trust networks. We discuss their strengths and limitations in the context of vote aggregation.

History based reputations In EigenTrust [13], identities rate each other based on past transactions among them. A good transaction between two honest identities results in a good rating while a honest entity fooled by a misbehaving identity will give it a low rating. In Credence [27], identities rate each other based on the similarity between their voting records on the same set of file objects. There are two limitations with using history-based user reputation in a voting system. First, an honest user with little history cannot quickly bootstrap himself to reach the required reputation threshold. On the other hand, an attacker can engage in good transactions to improve his reputation in order to cause greater damage in the future. Second, it is difficult to convert pair-wise ratings into a global reputation for each identity in a Sybil-resilient fashion. Both Credence and EigenTrust use PageRank-style reputation propagation to obtain a global rating for each user. Therefore, they have the same vulnerability as PageRank where an attacker can boost his ranking by a factor of $1/\epsilon$ using Sybil identities. Credence also limits Sybil attacks by having a central certification authority rate-limit the number of identities generated in the system.

Trust network based reputations A number of proposals from the semantics web and peer-to-peer literature rely on the trust network between users to compute reputations [5, 11, 17, 23, 31]. Since a trust link reflects offline social relationship between a pair of users, it is more difficult for an attacker to obtain many trust edges from honest users. Of the existing work, Advogato [17], Appleseed [31] and Sybilproof [5] 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 proof user reputation scheme does not directly translate into a Sybil proof voting system: Advogato only computes a non-zero reputation for a small set of identities, causing a majority of users not 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 the attacker's non-Sybil accounts. Together, these reputable Sybil identities can cast many bogus votes.

2.3 Sybil Defense using trust networks

Many systems use trust networks to defend against Sybil attacks for different applications: SybilGuard [29] and SybilLimit [28] help a node to admit another node in a decentralized system with high probability that the admitted node is an honest node instead of a Sybil identity. Ostra [20] limits the rate of unwanted communication that adversaries can inflict on honest nodes. Sybil-resilient DHTs [7, 16] ensure that DHT routing is correct in the face of Sybil attacks. Kaleidoscope [24] 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 regardless of the number of Sybil identities. In SybilLimit, each attack edge results in $O(\log n)$ Sybil identities accepted by honest nodes. SumUp limits the number of bogus votes to be no more than the number of attack edges 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 specific feedback mechanism used by SumUp is inspired by Ostra [20].

3 The Vote Aggregation Problem

In this section, we will describe the Sybil-resilient vote aggregation problem that SumUp addresses. To set the appropriate context, we begin by describing the system model of SumUp and the associated threat model.

A voting system consists of a collection of identities that cast votes for different objects. Any user can create an identity in the system. A vote from identity i on object o is associated with a value to express user i's opinion on the quality of object o. In the simplest case, each vote can be viewed as a positive or a negative vote (+1 or -1). Alternatively, to make votes more expressive, the value of a vote can range within a set of values with higher values indicating more favorable opinions. To rank objects, a ranking method typically requires an *aggregation metric* that combines all the votes for a given object. Aggregating votes in the face of Sybil attacks is a challenging problem since the attacker can easily create many Sybil identities to out-vote real users.

Leveraging trust networks is essential in defending against Sybil attacks [5, 17, 28, 29, 31]. In SumUp, user i creates a trust link to j if i believes that j does not collude with the attacker and will vote honestly. Even though each trust link is directional, each link's creation requires the consent of both users. We assume there are multiple adversarial identities, all of whom collude as a single attacker. As in [29], we refer to a link from an honest user to an adversarial identity as an attack edge. While creating Sybil identities and linking them to adversarial entities is easy, establishing trust relationships between honest users and adversarial identities is much harder, resulting in a relative small number of attack edges, e_A. We refer to votes from adversaries and Sybil identities as bogus votes.

In addition to a small e_A, it is essential to break the symmetry between honest nodes and Sybil nodes; in principle, the attacker can create many Sybil identities to make the Sybil network identical to the honest network. A recent impossibility result by Cheng and Friedman [5] shows that there exists no symmetric sybilproof reputation function. To break this symmetry, we need to bootstrap the network with one or more trusted identities which act as vote collectors. We also refer to the vote collector as the *source* node in subsequent discussions.

We describe SumUp in a centralized setup where the votes of different users and trust relationships between different users are collected and maintained by a single trusted central entity. This central entity is responsible for aggregating the user votes for each object and ranking objects. As such, SumUp knows the entire trust network and individual votes for different objects. This centralized mode of operation fits the structure of online web sites such as Digg, YouTube, Facebook, LiveJournal etc. where the votes of different users and trust relationships between

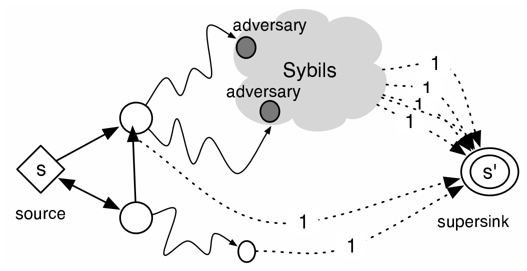

Figure 1: Vote aggregation can modeled as a max-flow problem by creating links with capacity one from voters to the supersink. Straight lines refer to trust links while curly lines refer to a path of multiple trust links. Adversarial identities are shown in black while Sybil nodes are shown in the dark cloud.

different users are collected and maintained by a single trusted entity. We describe how SumUp can be applied in a distributed setting in Section 8.

Under the system and attack model of SumUp, the sybil-resilient vote aggregation problem can be described as follows:

Vote Aggregation Problem: Let $G = (V, E)$ be a trust network with a trusted vote collector $s \in V$. V comprises of an unknown set of honest users $V_h \subset V$ (including s) and the attacker controls all vertices in $V - V_h$, many of which represent Sybil identities. Let e_A represent the number of attack edges from $V - V_h$ to honest users in V_h. Given that nodes in G cast votes on a specific object, a vote aggregation mechanism *ideally* achieves three properties:

1. Aggregate all votes from honest users.
2. Limit the number of bogus votes from the attacker by e_A independent of the number of Sybil identities in $V - V_h$.
3. Eventually ignore votes from nodes that repetitively cast bogus votes.

The three aforementioned properties are *ideal properties*, in that, it may not be possible to develop an algorithm that perfectly satisfies these conditions. Our system, SumUp, achieves approximate versions of these properties.

4 SumUp: Basic Approach

This section describes the basic idea of *adaptive vote flow* that SumUp uses to address the vote aggregation problem. Using this approach, SumUp can significantly limit the number of bogus votes without affecting the number of honest votes that can be gathered. As we show later in Section 5.4, SumUp can probabilistically achieve the three properties required to address the vote aggregation problem.

The concept of *max-flow* has been applied to several reputation systems based on trust networks [5, 17]. The flow concept is critical to limit the number of votes that Sybil identities can propagate for an object. When applied in the context of the vote aggregation problem, the approach is to compute the max-flow in the underlying trust network from the vote collector to the set of voting nodes (voters). As Figure 1 shows, we create an imaginary link with capacity value 1 from each voter to the supersink and compute the max-flow from the source (vote collector) s to the super-sink. We refer to the maximum number of bogus votes that adversaries can cast as the *attack capacity*. It is clear that with e_A attack edges, the max-flow based computation bounds the attack capacity by e_A. The fundamental problem with this max-flow based approach is that trust networks inherently are *sparse networks* and the total number of votes that can be collected using max-flow is very small. For example, networks from social networking sites like YouTube and Flickr all have a small median node degree from 1 to 5, thereby limiting the max-flow to the small node degree.

One possible option to increase the max-flow is to enhance the capacity of each link. However, increasing link capacities to accommodate more votes also correspondingly increases the capacity of the attacker to cast more bogus votes. Hence, there is a fundamental tradeoff in a Sybil-resilient voting system between the maximum number of honest votes it can collect and the number of potentially bogus votes the source might collect. Any symmetric and proportional increase in link capacities will not address this trade-off.

The *adaptive vote flow* technique aims to address this tradeoff that enables a trusted vote collector to collect a large fraction of honest votes while limiting the number of bogus votes by Sybil identities. The design of adaptive vote flow is centered around two basic observations. First, the number of honest users voting for an object, even if a popular one, is significantly smaller than the total number of users. Second, honest users voting for an object are relatively more spread out than bogus votes from Sybil identities behind a single attack edge.

The adaptive vote flow computation uses three key ideas. First, the algorithm restricts the maximum number of votes collected by the system to a value C_{max}. Explicitly constraining the maximum number of votes collected is essential to limit the number of bogus votes from adversarial nodes. Unfortunately, if the actual number of honest voters exceeds C_{max}, the algorithm will ignore the extra votes. We can partly deal with this issue by adaptively increasing C_{max}, provided lower values of C_{max} do not indicate any Sybil voting behavior (based on flow constraints). As we show in Section 5.4, when setting C_{max} to $O(\sqrt{n})$, the expected number of bogus votes is limited to $1 + o(1)$ per attack edge.

The second and important aspect of SumUp relates to *capacity assignment*: how do we assign flow capacities to each trust link to address the tradeoff between collecting C_{max} honest votes in a sparse network as well as restrict

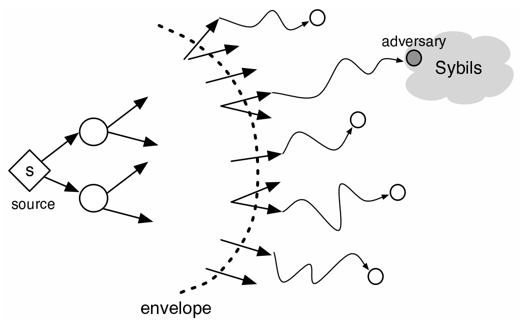

Figure 2: The vote envelope in capacity assignment. Beyond the envelope, all link capacities are assigned to be one.

the number of bogus votes per attack edge to one. SumUp uses the idea of the vote collector distributing 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 illustrated in Figure 2, the ticket distribution introduces a *vote envelope* which is essentially a cut with C_{max} vote entry points; beyond the envelope all the edges have capacity 1. Any adversarial node beyond the envelope can propagate at most 1 vote per attack edge independent of the number of Sybil identities behind the attack edge. SumUp redistribute tickets (using voting history) to deal with attack edges within the vote envelope.

The final key idea in SumUp is to leverage user history to penalize adversarial nodes which continuously propagate bogus votes. One cannot penalize individual identities since the adversary may always propagate bogus votes using new Sybil identities. Given the flow approach, the attack edge is guaranteed to be present in the path from the source to the identity [20]. SumUp uses this observation to re-adjust the ticket distribution across links to restrict adversarial nodes from continuously propagating bogus votes along attack edges.

5 The SumUp voting system

In this section, we describe a static capacity assignment algorithm that aims to achieve two out of the three vote aggregation properties discussed in Section 3: (a) Collect all votes of honest users, as long as less than C_{max} voters vote. (b) Restrict the number of bogus votes to one per attack edge. Later in Section 6, we show how to adapt the capacity assignment based on user voting history to deal with continuous misbehavior by adversarial nodes.

The capacity assignment algorithm consists of three steps. The first is a pruning step that could reduce the number of attack edges without seriously affecting honest users (Section 5.1). The next step is to assign positive capacity values to all links in the pruned graph such that the source is able to collect most of the C_{max} votes while minimizing attack capacity (Section 5.2). The last step is max-flow computation using a fast approximation algorithm that incrementally updates the vote count when users cast votes over a period of time (Section 5.3). We formally analyze the security properties of SumUp in Section 5.4 and discuss some practical issues with setting system parameters in Section 5.5. Table 1 summarizes various terms used in this Section.

Symbol	Meaning
e_A	Number of attack edges
C_{max}	Max number of votes to be collected (a system parameter)
C_A	Attack capacity, i.e. max number of bogus votes the source can collect
d_{in_thres}	Min number of incoming links to a node that are not pruned (a system parameter)

Table 1: A glossary of symbols and their meanings in the paper.

5.1 Pruning the trust network

The attacker capacity is directly affected by the number of attack edges. The smaller the value of e_A, the fewer bogus votes the source will collect. The goal of link pruning is to bound the in-degree of each node to a constant value, d_{in_thres}. This is primarily to restrict the damage of attackers with several incoming edges from honest nodes (greater than d_{in_thres}). 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 his vote collected. In other words, pruning should not cause a node to lose its connectivity from the source if a path from the source exists in the original network. The minimally connected network that satisfies this requirement is a tree rooted at the source. A tree topology minimizes attack edges because each adversarial identity is left with exactly one incoming link regardless of how many actual honest nodes that link to him in the original network. However, a tree topology is also overly restrictive for honest nodes because each node only has one path from the source: if that path reaches its full flow capacity, the user's vote cannot be collected. In particular, when C_{max} identities vote, a tree can collect only $(1-\frac{1}{e})C_{max} \approx 0.63 * C_{max}$ votes because a significant fraction of voters' paths conflict. 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 effect of the specific parameter choice of d_{in_thres} in Section 7.

Pruning each node to have at most d_{in_thres} incoming links is done as follows. We associate each node with a level according to its shortest path distance from the source. The source's level is 0 and the other nodes' levels can be calculated efficiently using a breadth-first-search tree from the source. In the first step, 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 each node with fewer than d_{in_thres} incoming links. And 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. This particular way of pruning is chosen to work well with SumUp's capacity assignment, to be described in the next Section. For nodes with more than d_{in_thres} incoming links in the original network, pruning reduces their incoming edges. We speculate that it is easier to trick honest nodes to trust an adversary node which has many honest neighbors than trust a singleton adversary node. Therefore, the number of attack edges in the pruned network is likely to be smaller than those in the original network.

5.2 Capacity assignment

The goal of capacity assignment is twofold. On one hand, the assignment should allow the source to collect almost all votes if less than C_{max} honest nodes vote. On the other hand, the assignment should minimize the attack capacity, C_A, to be not much more than the number of attack edges (e_A) in the pruned network.

As Section 4 illustrates, the basic idea of capacity assignment is to construct a vote envelope around the source. The vote envelope represents the boundary beyond which all edges in the network have a capacity value of 1. If all the e_A attack edges are beyond the vote envelope, then we can guarantee that the attack capacity is bounded by e_A. Hence the goal is to minimize the chances of an attack edge within the envelope and ensure that there is enough capacity within the envelop for all C_{max} entry points on the envelope to reach the source.

We achieve this goal using a *ticket distribution* mechanism which provides the following: the link capacity decreases with increasing distance from the source. The distribution mechanism is best described using a propagation model where the source is to spread C_{max} tickets across all links in the pruned network. Each ticket corresponds to a capacity value of 1. We group nodes into different levels according to the shortest distance from the source (as is done in Section 5.1) and distribute tickets to nodes one level at a time. If a node at level l has gotten 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 closest set of links that have a capacity of one represent the vote envelope of the source. One can visualize C_{max} such links as entry points collecting C_{max} votes among all nodes.

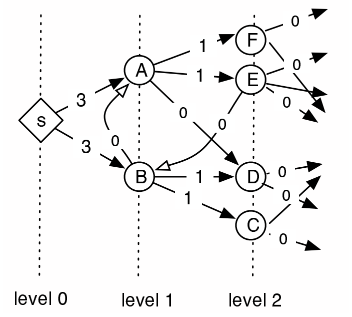

Figure 3: Each link shows the number of tickets distributed to that link from S (C_{max}=6). Each node consumes one ticket and distributes the remaining evenly via its outgoing links to the next level. Tickets are not distributed to links between nodes at the same level (B→A) or to links from a higher to lower level (E→B). The capacity of each link is equal to one plus the number of tickets.

During ticket distribution, each node must keep at least one ticket to itself so that the total number of tickets under distribution decreases at each higher level, eventually reaching zero after a small number of levels. Because each ticket corresponds to one capacity value, distributing fewer tickets lowers the chances of distributing tickets to attack edges, reducing C_A. Each node should not keep more than one ticket to itself since otherwise the t_{in} tickets distributed to a node can not be fully utilized by voters at higher levels. As a result, each node consumes exactly one ticket during distribution.

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

5.3 Incremental vote collection

Pruning and capacity assignment is done once for a given trust network and vote collector. The task remains to collect votes on each object from the resulting flow network. Existing max-flow algorithms such as Ford-Fulkerson and Preflow push [6] are not directly applicable for vote collection. In particular, existing algorithms do not care about which of the many competing paths they choose for max-flow. As a result, they are more likely to choose one of the many vote flows from the attackers than one from an honest user to traverse a congested link. Furthermore, if the max-flow capacity is reached, existing algorithms cannot incrementally update the results to account for subsequent votes whose opinions might have shifted. To address these shortcomings, we exploit the structure in capacity assignments to find an approximated max-flow solution that is both incrementally updatable and fast to compute.

The approximation algorithm used by SumUp is inspired by the Ford-Fulkerson method. The algorithm works incrementally to try to collect one additional vote from each new voter. To collect a vote, the approximation algorithm tries to find an augmenting path from the source to the corresponding voter in the residual graph. To find an augmenting path if one exists, the original Ford-Fulkerson performs a breadth-first-search from the source which costs $O(E)$ running time. To speed up this process, we perform an approximate search using depth-first-search (DFS) from the voter toward the source. At each step, the DFS prefers exploring links from neighbors at a lower level if such links with non-zero remaining capacity exist. We limit the running time of DFS by bounding the number of the backtracking steps it can take to find a path (e.g. 300). The choice of DFS makes a good heuristic because of our capacity assignment strategy: the capacity of links at higher levels is propagated from links at lower levels so preferentially exploring links at lower levels is more likely to find an non-zero capacity path.

When the number of votes collected so far approaches C_{max}, there will be no augmenting paths found for new voters. Instead of ignoring these votes, we probabilistically replace some existing votes with new ones. Such shuffling is necessary so that the final set of C_{max} collected votes can reflect a random sample among all votes. To support shuffling, each link keeps track of the set of identities whose votes it is carrying as well as the number of replacement attempts it has seen on those votes (x). If a new voter has failed to find an augmenting path during its bounded DFS search, it restarts the DFS and terminates the search upon exploring an incoming link with full capacity. With probability $1/x$, the replacement algorithm chooses a random vote among those carried by the full link and replaces it with the new vote. If the replaced voter has a complete path to the source, the entire path is taken over by the new voter.

Algorithm 1 shows the pseudocode for incremental vote update via replacement. The final set of collected votes can be obtained from the set of existing voters kept at the immediate outgoing links from the source. Vote replacement allows the final set of collected votes to be a random sample C_{max} votes from all voters without restarting the vote collection from scratch each time.

Algorithm 1 Replacement algorithm, run after the bounded DFS fails to find an augmenting path from voter i to the source s.

```
E(j,k) ← A full link encountered during DFS search from $i$ to $s$.
V ← set of existing voters on E(j,k)
x ← number of replacement attempts on E(j,k)
rand ← a random number in $[0, 1)$
if rand < 1/x then
    $r$ ← a random voter in V
    replace $r$ in V with $i$
    for each link e in $r$'s path to $s$ do
        replace $r$ with $i$ in the set of existing voters on e
    end for
    number of replacement attempts on E(j,k) ← x+1
end if
```

5.4 Security Properties

In this section, we provide a formal analysis of the security properties of SumUp under the assumption that the trust network can be approximated as an expander graph. Existing studies on various types of social networks have shown that social networks are indeed expander-like [15]. We specifically prove bounds on the expected attack capacity C_A and the expected number of votes collected if C_{max} honest users vote. This analysis also assumes that the trusted vote collector is a random node in the topology. In practice, even if the attacker is close to the source, we can use voting history (as shown in Section 6) to re-adjust link capacities to deal with this problem. Alternatively, one also can compute a "personalized ranking" by using each user as the vote collecting source for aggregating votes.

Theorem 5.1 *Given that the trust network G on n nodes is a bounded degree expander graph, the expected capacity per attack edge is $\frac{E(C_A)}{e_A} = 1 + O(\frac{C_{max}(\log C_{max})}{n})$ which is $1 + o(1)$ if $C_{max} = O(n^\alpha)$ for $\alpha < 1$. If e_A is a constant, the capacity per attack edge is bounded by 1 with high probability.*

Proof Sketch Let l_i represent the number of nodes in level i with $l_0 = 1$ representing the source. Let E_i represent the number of edges from level $i-1$ to i. Then, the average capacity of the E_i edges is bounded by C_{max}/E_i. The probability that one of the nodes in level i is an adversarial node is l_i/n. Given the expander graph property, we are guaranteed that l_i and E_i will increase exponentially with i. Hence, the distance of the vote envelope from the source is $\log C_{max}$. Beyond the vote envelope, the attack edge capacity is bounded by 1. Hence for a randomly placed adversarial node, the expected capacity can be calculated as $1 + \sum_i (l_i/n) \times (C_{max}/E_i)$. Given that l_i and E_i exponentially increase with i, we can show that the expected capacity is $1 + O(C_{max} \log C_{max}/n)$. In addition, the number of nodes within the envelope $\sum_i l_i$ is bounded by C_{max}. Hence, the attack capacity for a randomly placed adversarial node is exactly equal to 1 with probability $1 - \frac{C_{max}}{n}$. When e_A is a constant, the probability that any attack edge has capacity greater than 1 is bounded by $\frac{e_A \times C_{max}}{n}$. Hence for $C_{max} = O(n^{\alpha})$ for $\alpha < 1$, the expected attack capacity per attack edge is $1 + o(1)$ and the attack capacity is bounded by 1 with high probability.

While the expected attack capacity is $1 + o(1)$ per attack edge, the high probability bound on the attack capacity is for constant e_A. In the general case, there is a tradeoff between e_A and the choice of C_{max}. The probability that a randomly chosen node adversary has a capacity greater than 1, is $e_A C_{max}/n$. Hence the choice of C_{max} is inversely related to the choice of e_A. To bound attack capacity to be no more than e_A, it is essential to choose a small C_{max}. Since the value of e_A is not known, we choose a value of $C_{max} = O(\sqrt{n})$ in our system primarily due to convenience. We discuss the issue of setting C_{max} in Section 5.5.

Theorem 5.2 *Given that the trust network G on n nodes is a d-regular expander graph and C_{max} random honest voters, the expected fraction of votes that can be collected out of C_{max} votes is $1 - o(1)$ if $C_{max} = O(\sqrt{n})$.*

Proof Sketch Figure 2 illustrates the intuition that SumUp creates an envelop of with C_{max} entry points around the source via which votes are collected: if a vote flow reaches an entry point without conflicting with other flows, it is guaranteed to be collected by the source since paths within the envelop are assigned enough capacity. Therefore, in order to prove that the expected fraction of votes that can be collected is high, we need to show that the minimum cut of the graph between the set of C_{max} entry points and the set of C_{max} voters is large. Fortunately, expanders are well-connected graphs. In particular, the Expander mixing lemma [21] states that for any set S and T in a d-regular expander graph, the expected number of edges between S and T is $d \cdot |S| \cdot |T|/n$. Let S and T be the set of nodes that form a minimum cut between the entry points and the voters, i.e. $|S| + |T| = n$. Additionally, $|S| \geq C_{max}$ since S contains all the entry points. $|T| \geq C_{max}$ since T contains all the voters. Therefore, the min-cut value is $= d \cdot |S| \cdot |T|/n \geq d \cdot C_{max}(n - C_{max})/n$. Since max-flow calculated by SumUp between S and T is at least $1/d$ of the min-cut value (because SumUp assigns a capacity of 1 to the link from a voter to the supersink as opposed to ∞ in a typical max-flow setup), the fraction of votes that can be collected is $d \cdot C_{max}(n - C_{max})/(n \cdot d \cdot C_{max}) = 1 - o(1)$ if $C_{max} = \sqrt{n}$.

5.5 Practical Issues

Finding n: In our setting, the trust network is known to the central entity a priori and hence the value of n can be determined based on the size of the network. However, the trust network can potentially be pre-populated with a large number of Sybil identities which can significantly alter the actual value of n. One way to deal with this problem is to use the *expanded ring search* approach. Given the trusted vote collector, one can choose a diameter Δ and constrain the trust network to only the nodes within a distance Δ from the collector. In addition, to prevent an adversary from creating several Sybil identities as direct neighbors, we need to impose a degree bound d_{max} on each node. One can repeat the vote aggregation analysis for increasing values of Δ.

Setting C_{max}: We previously set C_{max} to $\sqrt{n}$ primarily for formally proving the security properties of SumUp. Ideally, to set C_{max}, we need to obtain the distribution of the number of votes for different objects and use the 95^{th} or 99^{th} percentile to set C_{max}. By analyzing voting patterns in different social networks, we found that in practice, the most popular objects in networks with over a million nodes have a few thousand votes. Hence, we feel that $\sqrt{n}$ is a reasonable choice for C_{max}; in addition, this value of C_{max} results in an attack edge having a capacity value of one for up to $\sqrt{n}$ attack edges. In practice, to determine the relatively rank of popular objects that receive more than C_{max} votes, we can use an adaptive ranking strategy by varying C_{max}. If the vote flow for an object at $C_{max} = \sqrt{n}$ is roughly equal to $\sqrt{n}$ (which means there is no signal of Sybil voting behavior), then we can recompute the vote flow for higher values of C_{max}. Using this adaptive strategy, we can relatively rank two popular objects.

6 Leveraging user feedback

The basic design outlined in Section 5 has two limitations. First, although the expected attack capacity is bounded by the number of attack edges, there might be cases where C_A is high when some adversarial identities happen to be close to the source. Second, 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 explore ways to utilize user feedback to address these two limitations.

We let the source node associate negative feedback with a voter if he disagrees with the vote. For example, if the source finds out that an object is a bogus file or a virus, he associates negative feedback to all voters who have voted highly on that object. Only negative feedback is used. This is to prevent adversaries from boosting their voting power by casting truthful votes on objects they do not care about.

When the source gives negative feedback to a vote, SumUp increments the negative history associated with each link along the entire path from the source to that voter. It is necessary to increment the negative history of 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, where a user that has received spam penalizes all links along the path to the email sender [20]. Unlike Ostra, SumUp uses a customized flow network per source and only allows the source to incorporate feedback for its associated network. Using only the source's feedback ensures that feedback is always trustworthy, thus preventing adversaries from framing honest nodes. Since every user in SumUp can act as his own source to compute a personalized ranking, each user will be able to customize a flow network using his own feedback.

SumUp uses the negative history of each link in two ways. First, we adjust each link's capacity assignment so that links that have previously carried more bogus votes have lower capacities. This helps to reduce attack capacity if the adversaries happen to be close to the source. Second, we eliminate links whose negative history has exceeded a certain threshold. Therefore, if adversaries continuously misbehave, the attack capacity will drop below e_A over time. We describe these approaches in detail in the rest of the Section.

6.1 Capacity adjustment

The capacity assignment in Section 5.2 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 more negative histories. Such adjustment is particular useful in circumstances where adversaries are close to the source and hence might receive a large number of tickets.

Let $h_{i,j}$ be the negative history associated with the link $i{\rightarrow}j$. The goal of capacity adjustment is to compute a weight, $w(h_{i,j})$, as a function of the link's negative history. The number of tickets node i distributes to one of its outgoing links is proportional to the link's weight, i.e. $t_{i,j} = t_{out} * w(h_{i,j}) / \sum_{\forall i \rightarrow k} w(h_{i,k})$.

How should the weights be computed? Clearly, a link with more negative history should have a smaller weight, i.e. $w(h_{i,j}) {<} w(h_{i,k})$ if $h_{i,j} {>} h_{i,k}$. Additionally, we require that if negative histories on two links increase by the same amount, the ratio between the links' weights should remain unchanged. To put it in formulas, the weight function must satisfy: $\forall h', h_{i,j}, h_{i,k}, \quad \frac{w(h_{i,j})}{w(h_{i,k})} = \frac{w(h_{i,j}+h')}{w(h_{i,k}+h')}$. This requirement matches our intuition that if two links have accumulated an equal amount of negative feedback over a period of time, the relative capacities between them should remain the same. The only function that satisfies both requirements is an exponential function of the form $w(h_{i,j}) = a * b^{h_{i,j}}$ where a, b are constants and $0{<}b{<}1$. We set a=1 and b=0.9 by default.

Network	Nodes $\times 1000$	Edges $\times 1000$	Degree 50%(90%)	Directed?
YouTube [20]	446	3,458	2 (12)	No
Flickr [19]	1,530	21,399	1 (15)	Yes
LiveJournal [19]	4,785	76,193	5 (40)	Yes
Synthetic [26]	500	3,072	5 (12)	No

Table 2: Statistics of the social network traces or model used for evaluating SumUp. All statistics are for the strongly connected component (SCC).

6.2 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 have accumulated too many negative histories.

We use the following heuristic for link elimination: we remove a link if its corresponding negative history exceeds five times the assigned capacity. Since we already prune the trust network (Section 5.1) before performing capacity assignment, we add back a previously pruned link if one exists after eliminating an incoming link due to excessive negative history. 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 link connecting him to that attacker will also be eliminated. When we add back a different incoming link to the attacker from the original trust network, the attacker will explore a different set of paths to the source, thus unlikely to affect the same user again. Despite the intuitive argument, there always exists pathological scenarios where link elimination affects some honest users, leaving them with no voting power. To address such potential drawbacks, we can re-enact eliminated links at a slow rate over time. We evaluate the effect of link elimination in Section 7.

7 Evaluation

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

1. On all networks under evaluation (YouTube, Flickr, LiveJournal), SumUp bounds the average number of bogus votes collected to be no more than e_A while being able to collect >80% of votes from C_{max} voters.
2. By incorporating feedback from the vote collector, SumUp dramatically cuts down the attack capacity for adversaries that continuously cast bogus votes.

3. We apply SumUp on 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 a strong evidence of Sybil attacks.

7.1 Experimental Setup

For the evaluation, we use a number of network datasets from different online social networking sites [19] as well as a synthetic social network [26] as the underlying trust network. Online social networks are not ideal for evaluating SumUp because the links created in sites such as YouTube, Flickr and LiveJournal serve a different purpose than those in SumUp. For example, a LiveJournal user might create a link to another user if he is interested in reading that user's blog. By contrast, a SumUp user should create a link to another user only if he trusts the other user to vote honestly. Despite such differences, we expect that the basic structure of the trust network in a deployed SumUp should resemble those in typical social networks because the origin of trust ultimately comes from the social interactions among users. Table 2 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.

In order to evaluate the ability of SumUp to bound bogus votes, we randomly choose a fraction of nodes as colluding adversary nodes. Thus, e_A is the average in-degree of a node times the number of adversary nodes. By default, we set C_{max}=$\sqrt{n}$, d_{in_thres}=3 and the DFS backtracking limit to be 300. For experiments in Section 7.2 and Section 7.3, we choose a random set of nodes as voters and compute each data point using the average of 100 experiments. We apply SumUp on the real world voting trace of Digg in Section 7.4 to examine how SumUp can be used to detect Sybil attacks in the wild.

7.2 Sybil-resilience of the basic design

The main goal of SumUp is to limit attack capacity while allowing honest users to vote. Figure 4 shows the average attack capacity per attack edge as the number of attack edges increases. As seen in Figure 4, the attack capacity (C_A) is much less than the number of attack edges (e_A) in the original network under various network topologies. In these topologies, we introduce some new adversarial nodes, each of them randomly pick d honest nodes in the graph to be neighbor with, where d is the average degree of the topologies. We find that pruning can significantly reduce the number of attack edges in the original graph by at least 40%. In addition, our capacity assignment algorithm can guarantee that the attack capacity is 1 for almost all attack edges. Pruning is less effective in Flickr

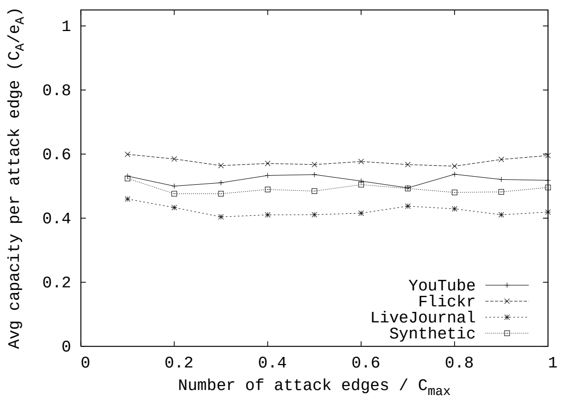

Figure 4: The average capacity per attack edge as a function of increasing attack edges. The attack capacity is less than the number of attack edges due to pruning.

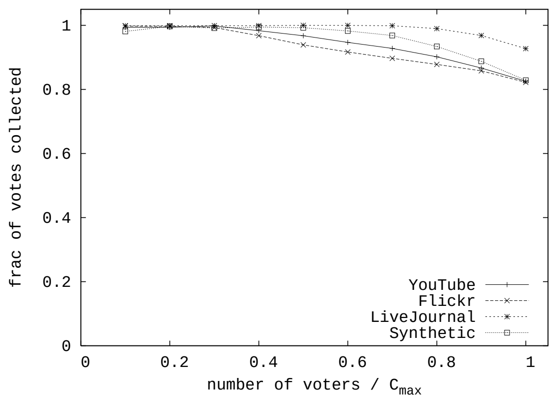

Figure 5: The fraction of votes collected as a function of the number of voters (up to C_{max}). SumUp collects more than 80% votes even when C_{max} users vote.

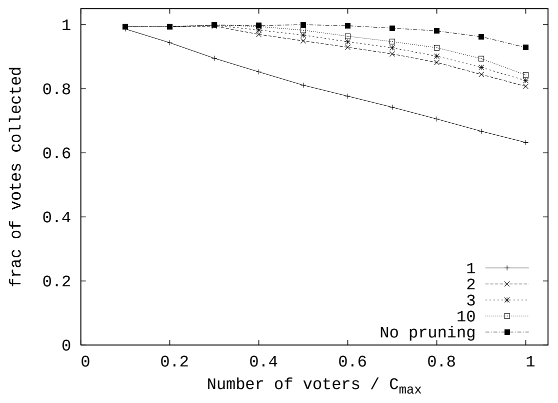

Figure 6: The fraction of votes collected for different values of d_{in_thres}. More than 80% votes are collected for $d_{in\_thres} \geq 3$.

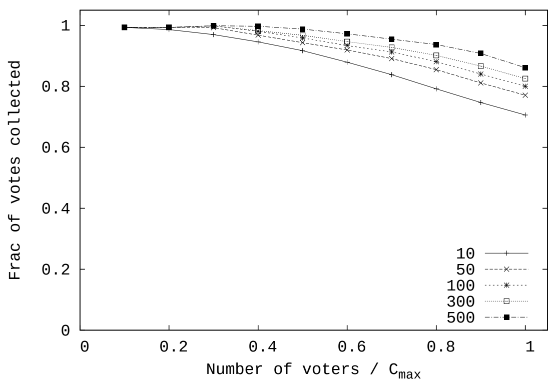

Figure 7: The fraction of votes collected for different DFS backtracking bounds. More than 80% votes are collected for DFS threshold ≥ 300.

than in other networks because there are more leaf nodes in Flickr network. When we apply step 4 of the pruning process, some pruned attack edges in the previous steps are added back. Figure 4 shows the average attack capacity for a randomly chosen source node. When using every node as its source, there are a few source nodes that experience much a larger attack capacity because they happen to be close to an adversary. These source nodes need to rely on feedbacks to re-adjust capacity assignment to reduce attack capacity.

Figure 5 plots the fraction of votes collected by SumUp as a function of the number of voters (up to C_{max}). As Figure 5 shows, even when C_{max} nodes vote, SumUp is able to collect more than 80% votes. This result confirms that capacity assignment via ticket distribution helps collect most votes when C_{max} users vote.

Figure 6 and Figure 7 evaluate two system parameters, d_{in_thres} and the depth-first-search (DFS) backtracking limit. Both parameters affect the fraction of votes that can be collected. Figure 6 shows that setting d_{in_thres} to be one drastically reduces the number of votes collected because of the lack of redundant paths from the source to voters. For a small d_{in_thres} such as 2 or 3, there is enough path redundancy for SumUp to collect most votes. As a result, we use $d_{in\_thres} = 3$ as the default parameter value.

Figure 7 evaluates the design choice of using bounded DFS to find an augmenting path for incremental vote collection. We vary the number of backtracking steps allowed during DFS and examine how this threshold impacts the fraction of votes collected. As Figure 7 shows, bounding DFS to less than 300 steps results in many fewer votes being collected. SumUp uses 300 steps as the default backtracking threshold for bounded DFS. In our experiments, bounded DFS allows SumUp to run at least an order of magnitude faster than the Ford-Fulkerson algorithm.

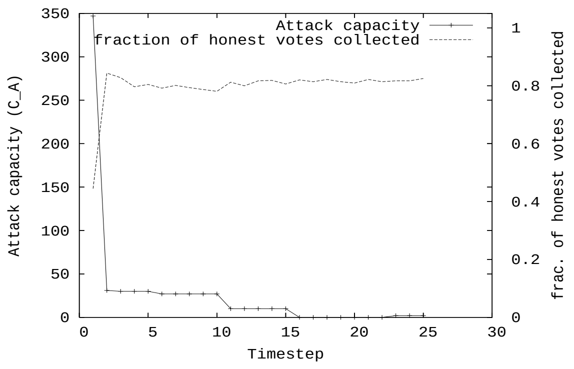

Figure 8: The change in attack capacity as adversaries continuously cast bogus votes. Capacity adjustment and link elimination dramatically reduces C_A while still allowing SumUp to collect more than 80% honest votes.

7.3 Benefits of incorporating feedback

We evaluate the benefits of capacity adjustment and link elimination using feedback from the vote collector. Figure 8 corresponds to an experiment with one randomly chosen source in YouTube with $C_{max} = 634$. There are 8 adversarial identities ($e_A = 64$) that cast bogus votes on a different object at each time step. At the same time, a random set of 634 honest users are also casting votes on the same object. At time step zero, because the chosen source happens to link to one of the adversaries directly, C_A is high ($> e_A^{'}$), resulting in 350 bogus votes collected and only 45% fraction of honest votes collected. After incorporating the source's feedback after the first time step to re-adjust the capacity adjustment, C_A is drastically reduced to approximately $e_A^{'}$. After another 5 time steps during which adversaries continue casting bogus votes, most attack edges in the current network are eliminated and previously pruned edges are added back. 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 never remains at zero forever. Figure 8 also shows that link elimination has little effects on honest nodes: as links are being eliminated, the fraction of votes collected from honest users always remains at about 82%, which is the same as without link elimination.

7.4 Defending Digg against Sybil attacks

Is there evidence of Sybil attacks in real world voting systems? Can SumUp successfully limit bogus votes from Sybil identities? In this Section, 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 online 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 displays them on

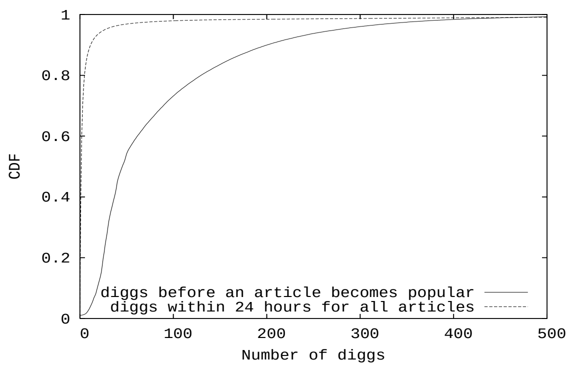

Figure 9: Distribution of diggs for all popular articles before being marked as such and for all articles.

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

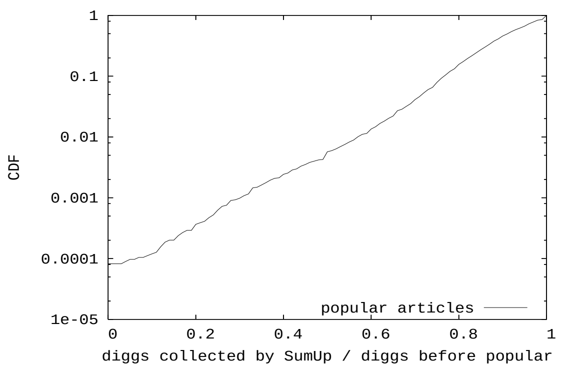

Figure 10: The distribution of the fraction of diggs collected by SumUp over all diggs before an article is marked as popular.

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. Digg's popularity ranking algorithm is intentionally not revealed to the public to mitigate gaming of the system. A Digg user can create a "follow" link to another user if he wants to follow the other's activity and to browse articles submitted by the other user. We have crawled the Digg site 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 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 only 15% of total nodes, 88% of votes come from nodes in SCC.

The most profitable avenue for attack lies in getting a submitted article to be marked as popular, therefore promoting an article 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, unlike experiments done in Section 7.2 and Section 7.3, 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.

Although the precise algorithm for deciding popularity is not known, we speculate that the number of diggs is a top contributor to an article's popularity status. Figure 9 shows the distribution of the number of diggs an article has received before it is 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 in Figure 9 indicates that the number of diggs indeed 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 source node and C_{max}=2000 to collect all diggs on an article before it is marked as popular. Figure 10 shows the distribution of the fraction of votes collected by SumUp over the total number of diggs that an article has received before becoming popular. The distribution is over for the entire set of 137,480 popular articles since Digg's launch. Our previous evaluation on various network topologies suggests that SumUp should be able to collect at least 80% of all votes (see Figure 5). Indeed, Figure 10 shows that more than 90% of all popular articles have more than 80% of their votes collected. Interestingly, there are a fair number of popular articles with fewer than 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 expected votes is due to real world Sybil attacks.

Since there is no ground truth dataset available to verify if few collected diggs are indeed the result of attacks, we resort to manual sampling. We set different fractions of collected diggs as a threshold for determining if an article is suspicious. Table 4 shows the result of manually inspecting 30 random articles from 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 receive hundreds of votes after being shown on the front page of Digg. We find no obvious attack evidence for roughly half of sampled articles. Interviews with Digg attackers [12] reveal that, although there is a fair amount of attack activities on Digg, attackers do not typically promote obviously bogus material. This is likely due to Digg being a highly monitored system as there are fewer than a hundred articles becoming popular every day. Instead, attackers try to get normal or even good content ranked as popular articles as promotion for others or to boost his profile within the Digg community.

As another evidence that a lower than expected fraction of collected diggs signals possible attacks, we examine Digg's *bury* data for articles submitted after 2008/08/13 of which 5794 are marked as popular. Figure 11 plots the correlation between the average number of bury votes on an article *after* it becomes popular vs. the fraction of the diggs SumUp has collected before it is marked as popular. As Figure 11 reveals, the higher the fraction of diggs collected by SumUp, the fewer the bury votes an article is going to receive after being marked as popular. Assuming most bury votes come from honest users that genuinely dislike the article, a large average of bury votes is a good indicator that the article is of dubious quality.

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 4: 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 the number of voters who registered on the same day as the article's submission exceeds 30% of all voters.

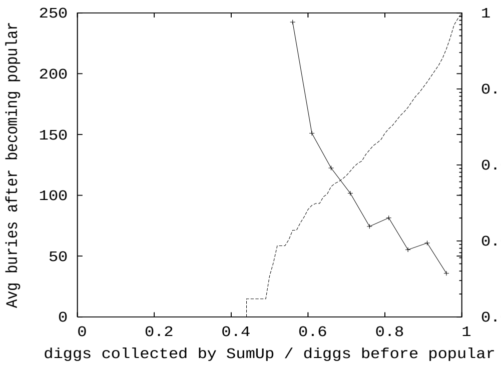

Figure 11: The average number of buries an article has received *after* it is marked as popular as a function of the fraction of diggs collected by SumUp *before* it is marked as popular. The Figure covers $5,794$ popular articles with bury data available. The distribution of the fraction of diggs collected is also shown in the second line.

What are the voting patterns on suspicious articles? Since 88% diggs come from nodes within the SCC, we should expect only 12% of diggs to originate from the rest of the network which mostly consist of nodes with no incoming follow links. For most suspicious articles, the reason for 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, it is likely that attackers might not bother to connect his Sybil identities to the SCC or to each other. Interestingly, we find 5 suspicious articles with sophisticated voting patterns where one voter links to a large number of identities that also vote on the same article. We believe the many identities behind that single 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 that differ in only one letter.

8 Discussion

Centralized vs Decentralized Setup: Even though SumUp is presented in a centralized setup such as a content-hosting Web site, it can be implemented in a distributed fashion to rank objects in peer-to-peer systems. We outline one such distributed design for SumUp. In the peer-to-peer environment, each node and its corresponding user is identified by a self-generated public key. A pair of users create a trust link relationship between them by signing the trust statement with their private keys. Each node gossips with each other or performs a crawl of the network to obtain a complete trust network between any pair of public keys. This is different from Ostra [20] and SybilLimit [28] which addresses the harder problem of decentralized routing where each user only knows about a small neighborhood around himself in the trust graph. In the peer-to-peer setup, each user naturally acts as his own source to compute a personalized ranking of objects. To obtain all votes on an object, a node can either perform flooding (like Credence [27]) or retrieve votes stored in a distributed hash table. In the latter case, it is important that the DHT itself be resilient against Sybil attacks. Recent work on Sybil-resilient DHT [7,16] and server selection [28] address this challenge.

Relationship to SybilLimit and SybilGuard: The approach taken in this paper is different in set up from the decentralized model taken in SybilGuard and SybilLimit. We operate in a centralized vote collector model which is an easier problem than the decentralized setup. While SybilLimit can reduce the number of Sybil identities to $O(\log n)$ per attack edge, we can reduce the number of votes to $1 + o(1)$ per attack edge. We are able to obtain this improvement primarily at the cost of limiting the maximum number of votes to C_{max}. Also, the C_{max} in our result is also inversely related to the number of attack edges e_A to maintain the probabilistic security guarantee. Specifically if we set $C_{max} = O(n)$, we achieve the same bound of $O(\log n)$ as SybilLimit.

9 Conclusion

This paper presented SumUp, a content voting system that leverages trust networks among users to defend against Sybil attacks. By using the technique of adaptive vote flow aggregation, SumUp aggregates a collection of votes with strong security guarantees: with high probability, the number of bogus votes collected is bounded by the number of attack edges while the number of honest votes collected is high. With detailed evaluations, we also show that the security properties of SumUp hold on practical social networks. We demonstrate the real-world benefits of SumUp by evaluating it on the voting trace of Digg: SumUp has detected many suspicious articles marked as "popular" by Digg. We have found a strong evidence of Sybil attacks on many of these suspicious articles.

References

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

[2] ARRINGTON, M. Engadget knocks $4 billion off apple market cap on bogus iphone email, 2007.

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

[4] BURDICK, D. Steve jobs heart attack bogus report sends apple stock reeling, 2008.

[5] 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, pp. 128–132.

[6] CORMEN, T., LEISERSON, C., AND RIVEST, R. *Introduction to Algorithms.* The MIT Press, 1985.

[7] DANEZIS, G., LESNIEWSKI-LAAS, C., KAASHOEK, M. F., AND ANDERSON, R. Sybil-resistant dht routing. In *European Symposium On Research In Computer Security* (2008).

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

[9] 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).

[10] FORD, B. Pseudonym parties: An offline foundation for online accountability. In *Unpublished manuscript* (2007).

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

[12] INVESPBLOG. An interview with digg top user, 2008.

[13] 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.

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

[15] 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).

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

[17] 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.

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

[19] 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).

[20] 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).

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

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

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

[24] 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).

[25] TECHCRUNCH. Stat gaming services come to youtube, 2007.

[26] 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.

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

[28] 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).

[29] 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.

[30] 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).

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