An Investigation of

Flow-based Algorithms for Sybil Defense.

by

Michael Bradley

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved February 2018 by the
Graduate Supervisory Committee:

Rida Bazzi, Chair
Andrea Richa
Arunabha Sen

ARIZONA STATE UNIVERSITY

May 2018

ABSTRACT

Distributed systems are prone to attacks, called Sybil attacks, wherein an adversary may generate an unbounded number of bogus identities to gain control over the system. In this thesis, an algorithm, DownhillFlow, for mitigating such attacks is presented and tested experimentally. The trust rankings produced by the algorithm are significantly better than those of the distributed SybilGuard protocol and only slightly worse than those of the best-known Sybil defense algorithm, ACL. The results obtained for ACL are consistent with those obtained in previous studies. The running times of the algorithms are also tested and two results are obtained: first, DownhillFlow's running time is found to be significantly faster than any existing algorithm including ACL, terminating in slightly over one second on the 300,000-node DBLP graph. This allows it to be used in settings such as dynamic networks as-is with no additional functionality needed. Second, when ACL is configured such that it matches DownhillFlow's speed, it fails to recognize large portions of the input graphs and its accuracy among the portion of the graphs it does recognize becomes lower than that of DownhillFlow.

To Mary, David, Eric, and Shanna.

~L.W.

TABLE OF CONTENTS

Page

Page

LIST OF TABLES

LIST OF FIGURES

DEFINITIONS AND ALGORITHMS

Number: Title Page

I – INTRODUCTION

i) Statement of the Problem

Distributed systems are prone to what is called the Sybil attack [13], where an adversary may generate an unbounded number of fake identities, called Sybil identities, to gain control over the system. This attack is applicable to practically any type of distributed system where users may organize into a trust network. These systems can differ in size, connectivity, the presence of a central authority, or any number of other factors. On a small scale, threats such as spam attacks are applicable pretty much anywhere. On a larger scale, social networks such as Facebook [16] can span millions, and even billions, of users, and suffer from the threat of fake accounts created to spam honest users, forge identities or any other range of uses. Furthermore, Sybil attacks can carry harmful consequences: applications such as P2P file sharing with no trusted authority run the risk of data loss if an adversary can compromise enough of the system to outvote honest users.

A wide literature of protocols aiming to help mitigate such attacks – which are commonly referred to as "Sybil defense" protocols – has developed. In this thesis, we investigate Sybil defense mechanisms as well. We focus on Sybil defense protocols that are based upon flow algorithms and their variants. To our knowledge, we are the first to do so; most Sybil defense protocols rely on random walks and their associated properties and distributions.

Sybil defense protocols fall into two major categories: 1) protocols that allow a honest user to "accept" or "reject" any other node in the trust network [24, 28, 29], and 2) protocols that allow an honest user to generate a full "trust ranking" t across *all* other nodes [3, 10, 11, 15]. In the former case, the robustness of a protocol – how accurately it can distinguish honest nodes from Sybils – is assessed qualitatively using such metrics as the number of Sybil node accepted per attack edge and the number of false positives (honest nodes declared Sybil by the protocol). In the latter case, the robustness of a protocol is assessed using such metrics as precision/recall or the area under the Receiver Operating Characteristic (abbreviated ROC) curve of the trust ranking [14].

While most recent approaches focus on the latter case, as we do, there is also a severe lack of consistency within the literature regarding how the robustness of the Sybil defense protocols are actually judged. Many protocols are individually claimed to be "optimal" and shown to achieve better results under the specific setup assumed by their underlying research. Some studies [10] focus primarily on small graphs (less than 100,000 nodes), sometimes using techniques such as BFS tracing to prune larger graphs down to a smaller size. The simulations conducted by most studies often introduce only a small number of attack edges [10, 11, 24, 29], trust links where an honest user falsely trusts a Sybil node. It is not clear if any of these techniques affect the robustness of the protocols at large, and it is difficult to obtain an "all-things-equal" comparison of the many protocols in the literature today. Moreover, there is a lack of consistency regarding the structure of the social graphs assumed: while many protocols assume that the honest

region of the network is fast-mixing (we define this in section II), it was shown by Mohaisen et al. [19] that this does not appear to actually be true in practice, instead supporting a notion that social graphs fall into a "community" structure of several tightly-knit subsets of nodes loosely connected with one another. Some protocols [10, 28] claim that this does not affect the fast-mixing property of their input graphs. Still others [24] assume an expander graph, a stronger condition than fast-mixing.

For this reason, our study largely follows the framework laid out by Alvisi et al. in [3], which addresses many of these issues and serves as a baseline to work from. We use precision/recall as our metric of choice and study the ACL community detection algorithm [4], which they showed to obtain near-optimal results on a variety of social graphs, as the current "leader". We aim both to 1) compare our results to those of ACL as well as other protocols and 2) demonstrate a replication of their results.

Our main contribution is an algorithm, termed DownhillFlow, which achieves good results on a wide range of social graphs under the two models of attack defined in [3]. We compare DownhillFlow to ACL and the distributed protocol SybilGuard, and find that DownhillFlow's results are slightly less robust than ACL's and noticeably more robust than SybilGuard's. Furthermore, DownhillFlow is very fast, taking slightly over 1.2 seconds for the 310,000-node DBLP graph, compared to about 45 minutes for ACL. Moreover, if ACL is restricted to run for up to 4 seconds, its results are not as good as those of DF. This performance advantage of DownhillFlow grows as the graph size

increases. We also show how an unoptimized version of DownhillFlow can be adapted to form a distributed protocol, present ideas to adapt the missing optimizations to the distributed setting and show how certain limitations of the distributed DownhillFlow protocol can be addressed by drawing a link to a specialized version of the electronic cash problem. Last, based on its running time, we propose several applications for DownhillFlow, the most promising of which is dynamic networks.

ii) System Model and Preliminaries

As customary in the literature, we model the system in terms of an unweighted, undirected graph $G = (V, E)$. Nodes $u \in V$ represent distinct (honest or Sybil) identities in the system, and an edge $e = uv$ represents a mutual trust connection between u and v. $H \subseteq V$ is the set of honest identities in the system, while $S \subseteq V$ is the set of Sybil identities. For simplicity's sake, we use "node" and "identity" interchangeably when describing honest and Sybil identities. Nodes $v \in S$ are Byzantine and all act under control of an adversary, an edge e is called an attack edge if it has one endpoint in H and the other in S. We are interested in algorithms leveraging these trust links to distinguish honest nodes from Sybil nodes.

We assume that the graph G is static. For a node $v \in G$, we denote its neighbors as $N(v)$, its degree as $\deg(v)$ and its distance from a node u as $d_u(v)$. We use e_a to denote the set of attack edges. Throughout the thesis, we use t to denote the trust vector obtained from a (fixed) source node s, which is assumed to be honest, and we use t_v to denote the

trust obtained for a single $v \in V$.

We use two metrics from information theory, *precision* and *recall*. For a trust vector t and any given position k, $1 \leq k \leq |H| + |S|$, the precision at position k is defined to be the percentage of the k highest-ranked nodes that are honest, and the recall is defined to be the percentage of honest nodes that appear in the k highest-ranked nodes. Thus, the two metrics trade off with each other: as a trust ranking recalls more honest nodes, it starts to accept a greater proportion of Sybil nodes as well, becoming less precise.

We consider two models for simulating Sybil attacks, which we refer to as the *random attack* and the *fixed attack*. Both were formalized by Alvisi et al. in [3], although the fixed attack model was previously employed in the literature. These are described below:

Random attack – In this model of attack, the Sybil region S is configured to be an exact copy of H. The adversary attempts to set up $|E|$ attack edges uv joining S to H. The endpoints u and v are chosen degree-preferentially, so for a node u, the probability of being chosen is $\frac{\deg(u)}{2|E|}$. The attack edge is created with probability p, a parameter of the attack, and fails with probability $1 - p$. The expected number of attack edges is thus $p|E|$.

Fixed attack – This model of attack takes two parameters (g, γ) as input, which represent a fixed number of attack edges and Sybil nodes to be created, respectively. In this attack,

nodes in the honest region H are randomly declared to be Sybil nodes until a total of g attack edges are reached, and from there, the Sybil region S is built up using a scale-free topology, such as Barabasi-Alberts [1], until a total of γ Sybil nodes are reached.

We consider both the *centralized* and *distributed* settings: in the centralized setting, a network operator with full knowledge of the graph topology picks a source $s \in H$ (determined, for instance, by manual verification) and runs the algorithm in a fully-trusted manner. In the distributed setting, a node $s \in H$ wishes to obtain its trust vector t. It is assumed that each $v \in V$ is only aware of its neighbors $N(v)$ – the nodes it has selected to trust – and it is explicitly assumed here that the edge set E, whose size is often several orders of magnitude above the vertex set V, is too large to fit on one machine. Without this assumption, the distributed setting is not interesting as a source node s can simply obtain a crawl of the entire graph. A node v in the distributed setting is identified by its public key PK_v: for the sake of simplicity, we omit this step in discussion henceforth and refer to nodes directly. We assume that the public key of the source PK_s is known to all nodes, and also leave out the specifics required for nodes v to forward messages back-and-forth to s. Since Sybil nodes all act under control of an adversary, this means that under the distributed setting, the adversary controls all $v \in S$. However, we assume communication is reliable and that nodes $v \in H$ are resilient to drop attacks (utilizing, for instance, timeout mechanisms).

iii) Organization of the Thesis

The rest of the thesis is organized as follows. In section II, we give an overview of the field since the inception of the Sybil attack in [13]. We review most existing protocols, beginning from SybilGuard [29] and continuing through to the state of the art [xxxx, 3, 10, 11, 24, 28, 29]. We present our algorithm in section III, and we discuss our experimental setup in section IV. In section V, we present and discuss results both demonstrating the accuracy of our protocol and analyzing its running time performance. In section VI, we attempt to adapt our algorithm into a distributed protocol, both discussing a concrete approach with accumulators and showing a potential link to a specialized type of e-cash scheme, and discuss the limitations that arise when doing so. We discuss future work in section VII and conclude in section VIII.

II – PREVIOUS WORK

The Sybil attack was formalized by Douceur in [13]. Initial investigations led to a variety of results. Bazzi and Konjevod [6] discussed approaches levering the geometric structure of the network to identify nodes based on their locations, and Bazzi et al. [5] proposed a Sybil-resilient distance vector routing scheme, guaranteeing under most assumptions Sybil nodes could not report a distance from a source node s lower than their actual distance. Kamvar et al. [15] proposed the EigenTrust reputation system, wherein a node v's reputation is determined by ratings given by its neighbors, weighted in turn by the trust of the neighbors.

The first protocol leveraging the structure of the whole social graph to defend against Sybil attacks was SybilGuard [29], which was introduced in 2006 by Yu et al. The key insight of SybilGuard is that if the honest subgraph is *fast-mixing*, then it is likely that two random walks originating from any two $u, v \in H$ will intersect with each other. We give a high-level definition of fast-mixing here and urge the reader to consult, for instance [19] for a more detailed discussion:

Definition 1 (Fast-mixing): A graph G is fast-mixing if a random walk starting from any $v \in V$ converges to the stationary distribution π, where π is the degree-normalized uniform distribution (for each v_i we have $\pi_{v_i} = \frac{\deg(v_i)}{2|E|}$), "quickly" – in $\mathcal{O}(\log|V|)$ steps.

They show how this can be done in a distributed manner by substituting the random walks with *random routes*. Each node v generates a random permutation $k_1, k_2, \ldots k_{\deg(v)}$ of the set 1, 2, … $\deg(v)$ and forms a routing table as follows: if a random route enters v along edge e_i, its next edge is determined by e_{k_i}. They later improve on SybilGuard by introducing SybilLimit [28], the key advantage of which is that rather than running one, long random route of length l, they run many shorter routes of length w, each under a unique routing table. They are able to show that this approach is sufficient to bound the number of Sybil nodes accepted to $\mathcal{O}(\log|\mathrm{V}|)$ per attack edge. However, their approach comes with the drawback that the input graphs must be *pre-processed* by iteratively removing nodes with degree lower than 5. This is required both to ensure fast-mixing and because a node v runs a random route through each of its

outgoing edges, and as such the number of routes ran by one v is constrained by its degree.

Since then, multiple protocols were proposed. SybilInfer [11] uses a Bayesian model to assess the likelihood that a random trace was initiated by honest nodes. They claim their protocol works in the distributed setting, however Yu et al. point out [28] its design is incomplete. Gatekeeper [24] is adapted from the SumUp [25] protocol, which was originally designed for online content voting. In Gatekeeper, a source s picks m random nodes to act as ticket sources, each of which distributes t tickets in a distributed manner; the value t is adjusted adaptively in order to ensure enough nodes receive tickets and thus are accepted. Gatekeeper claims to limit the number of Sybils accepted to $\mathcal{O}(1)$ per attack edge. However: 1) they assume the input graph is an expander graph, a stronger condition than the fast-mixing widely assumed elsewhere, and 2) they require the input graphs satisfy an added balance criterion that restricts the outcome of their ticket distribution mechanism. They essentially only provide guarantees on graphs that conform to a standard that suits their protocol.

All of these protocols rely on the assumption that their input graphs are fast-mixing. Mohaisen et al. report in [19], however, concrete measurements on a variety of social graphs demonstrating that this assumption doesn't hold in practice. They report that the graphs that do turn out to be fast-mixing, such as Facebook, are online social networks where there is no special need to be careful about which friend requests are accepted.

But the graphs that rely on strong, out-of-band trust links assumed by SybilLimit-like protocols, such as DBLP, are not fast-mixing. They also point out that the pre-processing step required by SybilGuard and SybilLimit is problematic: in some cases it causes over 75% of the nodes to be denied service. Viswanath et al. [27] report that social graphs more closely take on a structure of several, loosely coupled *communities*, each of which are individually tightly connected, and point out that community detection algorithms serve largely the same purpose as the random walks in prior protocols.

The next protocol we consider, SybilRank [10], provides the insight that the landing distribution of an early-terminated random walk – namely, length $\mathcal{O}(\log t)$, where t is the mixing time of the honest region – gives honest nodes enough of an advantage over the stationary distribution to be effectively distinguished from Sybil nodes. SybilRank works by initializing a fixed amount of trust on a given number of seed nodes, and utilizing power iteration to compute this distribution in an efficient manner: intuitively, rather than simulating many random walks, it computes the exact distribution one step at a time until t steps. They heuristically set $t = \log|V|$ and demonstrate how the community structure can be handled by manually distributing seed nodes across several communities. They compare against several other protocols and Mislove's community detection [18] algorithm. This algorithm functions as a greedy heuristic: to identify a community C of surrounding nodes, the algorithm repeatedly adds the neighbor $v \in N(C)$ such that adding v results in minimal normalized conductance (according to the expected value).

In [3], Alvisi et al. further investigate parallels between Sybil defense and community detection. They select the ACL algorithm [4], proving that if the mixing time of a community $C \subseteq V$ is sufficient, it allows for almost all honest $v \in C$ to compute a trust ranking, the first $|C|$ positions of which consist almost entirely of honest nodes in C. This is a significant improvement as it is the first result to mathematically address the issue of individual fast-mixing communities, rather than addressing only the mixing time of the entire honest region and proceeding on a best-effort basis. They demonstrate that ACL achieves robust results on a variety of social graphs, formalize two models of simulation for Sybil attacks, and examine what happens under much stronger attacks than those assumed elsewhere in the literature; namely, their model scales according to graph size and introduces attack edges proportionally to the number of *edges*, not vertices (or constant), in the graph. As an example, under the LiveMocha graph we use later with p = 0.1, their model induces $|e_a|$ > 200,000, whereas $|e_a|$ in other studies is assumed to be on the order of magnitude of thousands. Even the empirical analysis of SybilLimit conducted by Yu et al. [28] shows that on all graphs, the number of Sybil nodes accepted $\rightarrow |V|$ as $|e_a|$ approaches 100,000.

We must also describe the ACL algorithm itself: rather than handling normal random walks, ACL approximates the degree-normalized distribution that a node is visited by a random walk of geometric random length. At each step, the random walk has probability α of returning to the source node, called the *jumpback parameter*. This provides many of the guarantees of random walk theory while prioritizing nodes close to the source node,

ranking more highly nodes in the same community as the source. This distribution is approximated by a repeated series of steps wherein nodes take the correct amount of trust for themselves and propagate the remainder to their neighbors – for a more detailed discussion, refer to [4] or Alvisi's implementation [3]. The running time of ACL is $\mathcal{O}(\frac{1}{\alpha\epsilon})$, where ϵ is an *error parameter* setting how close the approximated distribution must be to the limit distribution; this creates a tradeoff between its speed and the robustness of its results. We discuss this in more depth in sections IV and V.

It is also worth discussing the SumUp vote collection protocol [25]. SumUp, the ticket distribution of which was reused in Gatekeeper, is, in fact, based around flow. To collect C_{max} votes for an object, a source s distributes C_{max} tickets to a set T of nodes through breadth-first search until T has C_{max} outgoing edges. s then calculates a set of *flow paths* to the nodes $v \in V$ who have voted for that object. These are combined within the flow envelope to create a flow of total capacity C_{max}. This is a different and more restricted problem than that of universal Sybil defense, however, since rather than evaluating the trustworthiness of any other node in the graph, it aims to select specifically C_{max} nodes out of a predefined set of nodes the size of which is expected to be much smaller than the total number of nodes in the graph. As such, this approach is not adaptable to the problem of universal Sybil defense, and in fact, Tran et al. report on these limitations themselves [24] when presenting Gatekeeper.

The intuition behind why flow algorithms seem promising under the context of Sybil

defense is fairly straightforward. Suppose we have a graph $G = (V, E)$ with honest and Sybil subgraphs H and S respectively. If $s \in H$ and e_a is the set of attack edges, then e_a defines a cut separating H and S. Therefore, if we can find an s, v-flow f of value $f > |e_a|$, we must necessarily have $v \in H$. Ideally, in this way, we could cleanly separate the nodes in H from the nodes in S. Furthermore, it is not known whether this approach requires as strong of a set of assumptions as those assumed in many random walk-based protocols, which is particularly important to handle the "community" structure found in social graphs in practice.

III – OUR APPROACH

a) "Naïve" Approach

To get an intuition of a flow-based approach for Sybil defense, consider whether a standard maximum flow algorithm, such as Ford-Fulkerson or preflow-push, can provide good results in the context of Sybil defense. The most obvious way to apply such algorithms to our problem is in the following manner: for a source node $s \in H$, calculate the maximum flow to all other nodes $v \in V$ and set $t_v = f_{max}(s, v)$.

We find, however, that this approach is not sufficient. We provide intuition why: for an attacked graph G, let e_a be the set of attack edges, and let $v \in V$. Since s's trust value for v, t_v, is the maximum flow from s to v, it is equivalent to the minimum s, v-cut. However, we find that the capacity of the minimum s, v-cut is not bounded by $|e_a|$ as expected. In fact, almost always, we have both $\deg(s)$ and $\deg(v) < |e_a|$, meaning that

to find a smaller cut, we need only look at the edges incident to s and v. This means that this approach cannot possibly distinguish between honest and Sybil nodes: to do so, we need for honest nodes to attain a flow of at least $|e_a|$. As an example, on the epinions graph with $p = 0.01$, we have $|e_a| \approx 1{,}000$, but $deg_{max} = 443$. When we consider the average degree and not the maximum, the outlook is even worse: we have $deg_{avg} = 7$. Indeed, in practice, we find that this does place a tight bound on the limiting cut for most v, regardless of whether v is honest or Sybil: for most v, we have $t_v = \max(\deg(s), \deg(v))$.

One possible way to mitigate this is to incorporate the neighbors of s and v into the flow calculation: meaning, calculate $T_1 = \{u : d_s(u) \leq 1\}$, $T_2 = \{u : d_v(u) \leq 1\}$ and set $t_v = f_{max}(T_1, T_2)$. However, we find this is still not sufficient: on average for the epinions graph, this only allows for 100~200 units of flow to be pushed from s to v, still not enough to reach $|e_a|$. Moreover, this approach carries the risk that a source node $s \in H$ hoping to calculate its trust ranking will find that it is adjacent to the Sybil region, at which point the adversary can take an unlimited amount of flow from s.

We conclude that by itself, the concept of maximum flow is insufficient to solve our problem. We thus investigate algorithms that use flow as a core concept while being more specialized toward the goal of Sybil defense.

b) Our Approach

Our approach is based on the insight that if flow is only allowed to be pushed "downhill" – meaning, pushed uniformly to nodes at greater distance from the source node s – then this leverages the assumption that the adversary's ability to create attack edges is limited and provides multiple other properties suitable for Sybil defense. (Note that since the flow is pushed uniformly, the flow is separated throughout the graph exponentially by BFS level.) To illustrate this, if $v \in S$ is a Sybil node with exactly one attack edge uv, then the flow v receives from the honest region of the graph will primarily come from the one attacked vertex u, since the flow passed to S over attack edges elsewhere must find its way over to v, being split apart exponentially at each hop. Moreover, the flow received will be limited as v only has one incoming edge, as opposed to, ideally, several incoming edges for a node in the honest region. Furthermore, since the flow is spread out exponentially, it prioritizes nodes in close range of the source node s, which intuitively should make nodes in s's community more likely to be highly ranked. This approach also prioritizes nodes in denser areas of the graph, as even though a node v cannot take flow from all of its neighbors, such v would likely still have a significant proportion of their edges coming from nodes at the next lower distance which they could receive flow from. Since social graphs consist mostly of nodes split into dense communities, this would serve to benefit honest nodes.

This algorithm, which we call the "DownhillFlow" algorithm (abbreviated DF), is shown below:

Algorithm 1: DF1(s, d_s)

1. $t_s \leftarrow 1$
2. $t_v \leftarrow 0 \; \forall v \in G \setminus \{s\}$
3. $Q \leftarrow \{s\}$
4. **while** $Q \neq \emptyset$ **do:**
5. Extract $v \leftarrow Q$.
6. **for** $w \in N(v) : d_s(w) = d_s(v) + 1$ **do:**
7. **if** $t_w = 0$ **then** Insert $Q \leftarrow w$.
8. $t_w \leftarrow t_w + t_v/\deg(v)$
9. $t_v \leftarrow t_v/\deg(v) \;\; \forall v \in G$
10. **return** t

Essentially, the algorithm functions by running a breadth-first search starting from the source node s. When a node v is processed, it splits its flow evenly among its $\deg(v)$ neighbors w. Then v pushes the flow to w if $d_s(w) > d_s(v)$ and discards it otherwise. Note that since we have $d_s(w) > d_s(v)$ with the edge $vw \in E$, we must necessarily have $d_s(w) = d_s(v) + 1$.

Some generalizations are made in the above algorithm for readability. We use d_s to indicate the distance vector of all nodes from s, which the algorithm takes as input. In practice, however, this is unnecessary as the distance calculation can be streamlined in the BFS mechanism of the algorithm. Also, as with many other Sybil defense protocols, the trust values are all normalized by dividing by degree. Rather than reiterating through all of the nodes to do this, this step can be done inside of the while loop in steps 4~8 after each node pushes flow to its neighbors, as its trust value is not used again after this is done. In section V, we will show that this degree-normalization improves the precision of the algorithm (and in fact turns out to be necessary for the algorithm to compute results that are significantly above random) and speculate on why this is the case.

Initial tests of this algorithm showed promising results; however, there is still room for improvement. Namely, the edges vw such that $d_s(v) = d_s(w)$ are entirely ignored by the algorithm, since an edge vw has flow pushed across it if and only if $d_s(v) = d_s(w) + 1$ or vice versa. This seems to go against the goal of ranking nodes more highly that fall into denser areas of the graph.

We can remedy this by adding an extra step to the algorithm wherein a node v accepts flow from its same-level neighbors. Note that care needs to be taken to ensure that the same-level push step is simultaneous, i.e. nodes do not mix up their same-level flow before pushing to other nodes at the same level. We show how this is done here:

Algorithm 2: DF2(s, d_s)

1. $t_s, s_s \leftarrow 1$
2. $t_v, s_v \leftarrow 0\ \forall v \in G \setminus \{s\}$
3. $Q \leftarrow \{s\}$
4. **while** $Q \neq \emptyset$ **do:**
5. Extract $v \leftarrow Q$.
6. **for** $w \in N(v) : d_s(w) = d_s(v)$ **do:**
7. $s_v \leftarrow t_w/\deg(w)$
8. **for** $w \in N(v) : d_s(w) = d_s(v) + 1$ **do:**
9. **if** $t_w = 0$ **then** Insert $Q \leftarrow w$.
10. $t_w \leftarrow t_w + (t_v + s_v)/\deg(v)$
11. $t_v \leftarrow (t_v + s_v)/\deg(v)\ \ \forall v \in G$
12. **return** t

Each node v keeps track of a separate value, s_v for its trust obtained from flow pushed from same-level nodes, and this value is not added into its total trust t_v until immediately before v pushes its trust to its higher-level neighbors. This serves to modify DF1 such that rather than ignoring same-level edges, they are instead utilized "both ways": same-level nodes u and v that share an edge uv will both push the flow they receive from

lower-level nodes to each other across that edge.

This provides a slight improvement over DF1, but still suffers from the issue that flow can only be pushed through same-level edges at most once per BFS level, as each node pushes to its higher-level neighbors after receiving its same-level flow. It is not clear if it is possible to efficiently work around this limitation with this approach – not only is the asymptotic running time of the algorithm affected if the number of same-level push steps is not bounded by a constant, it is not clear exactly how many same-level pushes is optimal.

A different approach is to use a token system to determine where flow should be pushed instead of relying entirely on the distance from s:

Algorithm 3: DF3(s, d_s)

1. $t_s \leftarrow 1$
2. $t_v \leftarrow 0 \; \forall v \in G \setminus \{s\}$
3. $T \leftarrow 1$
4. $token_s \leftarrow T$
5. $token_v \leftarrow 0 \; \forall v \in G \setminus \{s\}$
6. $Q \leftarrow \{s\}$
7. **while** $Q \neq \emptyset$ **do:**
8. Extract $v \leftarrow Q$.
9. **for** $w \in N(v) : token_w = 0$ or $token_w > token_v$ **do:**
10. **if** $token_w = 0$ **then:**
11. $T \leftarrow T + 1$
12. $token_w \leftarrow T$
13. Insert $Q \leftarrow w$.
14. $t_w \leftarrow t_w + t_v/\deg(v)$
15. $t_v \leftarrow t_v/\deg(v) \; \forall v \in G$
16. **return** t

In this version of the algorithm, each node v has a unique token, $token_v$, assigned to it. Here we start with $token_s = 1$ and each time a new node is seen, we increment the token value by 1 and assign it to that node. However, the tokens need not be constrained except that they must be unique and for nodes v and w with $d_s(v) > d_s(w)$, we must have $token_v > token_w$. DF3 accomplishes this since tokens are assigned only once, when we first see a node, and since it runs using BFS as a base, it processes all nodes in non-decreasing order of distance, and thus there is no way to assign a node at a higher distance a token of lower value. One caveat of this approach is that when we process a node v, we assign many tokens all at once to its neighbors $w \in N(v)$, each of which must be unique. Therefore, there must be some way of picking the order v's neighbors are processed. For our purposes, it suffices to pick them in random order.

This approach is not perfect – it is possible that a node v with several same-distance edges vw still will not receive flow from any of them, since we may have $token_w > token_v$ for every such $w \in N(v)$. However, in terms of the edges used, it is a significant improvement over the first version of the algorithm as here all edges are used in one direction or the other, and improves on DF2 in that it does allow for some, limited, capability for flow to be continually pushed across a BFS level.

Initial tests showed better results for this version of the algorithm than both DF1 and DF2. It is not obvious to us if there is a better method of incorporating the same-distance edges into our algorithm in a way that preserves the efficiency of the algorithm (as well

as some other relevant properties that we will discuss later). As such, all results reported henceforth for the centralized setting will be for DF3.

Note that all versions of the algorithm have time complexity $\mathcal{O}(|V| + |E|)$. For DF1 and DF3, the while loop in steps 4 and 7 respectively processes each vertex at most once, and the for loop in steps 6/9 runs at most once in total for each edge of the graph. For DF2, the while loop processes each vertex at most once, and the for loop processes same-level edges at most twice and all other edges at most once. As the distance calculation and normalization by degree can both be streamlined within the while loop, neither of these steps add to the time complexity of the algorithm.

c) Theoretical Guarantees and Relation to Random Walks

We do not prove any theoretical guarantees of DownhillFlow in this thesis, instead aiming to show its effectiveness experimentally through comprehensive simulation. We leave analysis of the theoretical guarantees of DownhillFlow to future work. For our purposes, we do not believe this to be a problem, primarily for reasons discussed in I and II: without preprocessing, as with SybilGuard, it is very hard to define constraints that apply universally to all social graphs, and even traits widely assumed about social graphs in the literature, such as fast-mixing, do not appear to hold in practice [17, 19]. Moreover, the "defense in depth" approach put forth by Alvisi [3] provides an argument that it is not too much of a disaster if one individual protocol fails to address all possibilities: theoretical weaknesses of one protocol can be made up for by additionally

using other protocols with orthogonal weaknesses. For this reason, we believe that it is of utmost importance to demonstrate robustness in practice first and delve into theory later.

It is worth noting that although DownhillFlow is designed to function as a flow algorithm, it turns out that the trust vector it computes can be modeled in terms of a specific type of random walk. We show how this is done here: let $s \in H$ be a source node, and let d be the distance vector from s to all nodes $u \in G$. At any given step of the walk, with v being our current node, we pick a uniformly random $w \in N(v)$. Then we move to w if and only if $d_w = d_v + 1$ and stop otherwise. This walk, which we call a "downhill random walk", shares similarity to the geometric-length random walks used by ACL, with a major difference: rather than having probability α of the walk ending at all hops, the probability instead varies at each hop, according to the ratio of each v's higher-level neighbors to all its neighbors. Since many of ACL's community-detection guarantees stem from the use of its geometric-length random walks, we speculate based on this similarity that the distribution calculated by DownhillFlow effectively serves as a method of more loosely identifying a community of nodes surrounding v. We leave further discussion to future work.

IV – EXPERIMENTAL SETUP

a) Robustness Tests

To measure the robustness of DownhillFlow's results, we compare against the ACL

community detection algorithm, which is implemented as in [2]. We also compare against SybilGuard, as although it was the first widely cited protocol using the social graph structure as a foundation for Sybil defense, it is one of the few existing protocols that retains the quality of being distributed. We aim to show that DownhillFlow obtains results better than those of SybilGuard, and only slightly worse than those of ACL.

We use five graphs, which we selected from both Stanford Network Analysis Project [23] and Online Network Repository [22]. The graphs range in size from $|V| \approx 26{,}000$ to $|V| \approx 300{,}000$. The graphs, along with various properties of them, are listed in Table 1:

Graph	$\lvert V\rvert$	$\lvert E\rvert$	deg_{avg}	deg_{max}	Diam.
epinions	26,588	100,120	7.53	443	17
Twitter	81,306	1,768,149	33.01	3,383	7
Slashdot	82,168	948,464	12.27	2,552	11
LiveMocha	104,103	2,193,083	42.13	2,980	6
DBLP	317,080	1,049,866	6.62	343	21

Table 1: Number of nodes, number of edges, average and maximum degree, and diameter for the epinions, Twitter, Slashdot, LiveMocha, and DBLP graph datasets used in our study. The datasets were obtained from [22, 23].

Our methodology closely resembles that of Alvisi et al. in [3]. We simulate Sybil attacks on the above five graphs under the two models of attack introduced in section Ib, and generate full trust rankings across all nodes in each graph. For the random attack model, we test with with p = 0.01, 0.03, 0.05, 0.07, 0.09. For the fixed attack model, we test two sets of parameters, intended to represent a weak attack ($g \approx \frac{1}{100}|E|$) and a strong attack ($g \approx \frac{1}{10}|E|$). γ is set according to $|V|$: with the exception of epinions, we set γ = 25,000 when $|V| < 100{,}000$ and γ = 50,000 otherwise. Because of its small size, we set γ =

5,000 for epinions.

The parameters tested are listed in Table 2:

Graph	(g, γ), weak	(g, γ), strong
epinions	(1,000, 5,000)	(10,000, 5,000)
Twitter	(16,000, 25,000)	(160,000, 25,000)
Slashdot	(10,000, 25,000)	(100,000, 25,000)
LiveMocha	(20,000, 50,000)	(200,000, 50,000)
DBLP	(10,000, 50,000)	(100,000, 50,000)

Table 2: Parameters for the fixed attack model. Both weak and strong attacks are simulated. The number of Sybil nodes γ remains the same for each graph, while the number of attack edges g varies.

To reduce the variance of our results, we test each iteration of the experiment from ten sources and take the average of the values we obtain. We speculate in section VII on whether it may be possible to obtain better results by combining the values in other manners besides the average. Since SybilGuard requires its input graphs to be preprocessed such that all nodes of degree fewer than 5 are iteratively removed, we perform that step as well when testing SybilGuard, but not with DownhillFlow or ACL. We also limit the source selection to nodes that do not fall within distance 2 of the Sybil region after the graph is attacked. When this is impossible, we allow nodes that fall within distance 2, but not within distance 1 (i.e. nodes adjacent to the Sybil region) – this is sufficient to cover all cases.

We configure ACL to have $\alpha = 10^{-3}$ and $\epsilon = 10^{-6}$ for epinions, $\epsilon = 10^{-7}$ for all other graphs. These settings are identical to the ones proposed by Alvisi et al. in [3]. Throughout all tests henceforth, the value of α is fixed, and we investigate only

variations to the value of ϵ. Some additional care needs to be taken to compute the trust ranking in the case of SybilGuard. While DownhillFlow and ACL naturally compute a trust vector t across all nodes in the graph, SybilGuard functions differently: it's a protocol designed such that a honest source s can choose to accept or reject any other single node in the graph v. Thus, for our experiment, we set t_v to be the number of s's random routes that accept v. Note that because the graphs in SybilGuard are preprocessed, $\deg(s) \geq 5$, and since s originates a random route from each edge incident to it, one source s will always have at least 5 random routes. Also note that in SybilGuard, a node v is accepted if at least $\frac{\deg(s)}{2}$ of s's routes accept v, so we have essentially eliminated this check and used the count of accepted routes directly. We also need to set the length of the random routes l. Yu et al. [29] recommend a route length of $l = \Theta(\sqrt{n}\log(n))$, but this bound is asymptotic and we must set the hidden constant. We determined this by manually testing several such constants and using the one for which SybilGuard generated the best results. This was 1/40 for epinions and 1/100 for other graphs.

Table 3 shows the effects of preprocessing the graphs for SybilGuard and the exact route lengths used for random attack, p = 0.01:

b) Running Time Tests

We analyze the running time of DownhillFlow by comparing it to that of ACL. DownhillFlow has an asymptotic running time of $\mathcal{O}(|V| + |E|)$ and takes no parameters,

	$\|V\|$, original	$\|V\|$, pre-processed	% removed	l
epinions	26,588	5,904	77.79%	15
Twitter	81,306	62,516	23.11%	27
Slashdot	82,168	26,752	67.44%	16
LiveMocha	104,103	79,811	23.33%	31
DBLP	317,080	98,942	68.79%	36

Table 3: Information for our graphs related to SybilGuard. The original number of nodes, number of nodes remaining after preprocessing, and % removed are shown along with the route length l.

so we can simply measure its running time. However, ACL's situation is more complicated: its running time is $\mathcal{O}(\frac{1}{\alpha\epsilon})$, inversely proportional to the jumpback parameter α and error parameter ϵ. This means that its running time is determined by the desired accuracy of its results: increasing ϵ gives less precise results, but makes the algorithm finish more quickly. In fact, increasing ϵ too much means that some nodes v aren't even recognized by ACL – that is, they have $t_v = 0$. Of course, if $t_v = 0$, v can't possibly be distinguished from any Sybil node. So, we must not only measure the precision of ACL's results, but the fraction of nodes for which ACL actually obtains results. We say that ACL "captures" a node v for a certain ϵ if it returns $t_v > 0$ for that ϵ.

Another point about ACL's running time is that it is entirely independent of the size of the graph, depending on only the jumpback and error parameters α and ϵ. In theory, this would mean that as the graph size tends to infinity, ACL's running time would be less than that of DF. It is not so clear, however, if and where the $\mathcal{O}(|V| + |E|)$ bound overtakes the $\mathcal{O}(\frac{1}{\alpha\epsilon})$ bound on social graphs in use today. It is also not clear the impact of graph size on the value of ϵ necessary to capture the entire honest region.

To give insight into both of these problems, we run DownhillFlow and ACL on epinions and DBLP, the two extremes amongst the graphs we use in testing in terms of $|V|$, under the random attack from ten sources and measure the running time across each run. We assume a qualitative worst-case scenario for purposes of DownhillFlow's results: p is set to the largest possible value where DownhillFlow is capable of distinguishing a large portion of the graph (for our purposes, we require at least 90% precision at 50% recall). For epinions, p = 0.1; for DBLP, p = 0.03. For ACL, we vary ϵ across a range of values. We start at the ϵ value used in the experiments in i (for which ACL captures the entire graph), and increase the value logarithmically until ACL's running time becomes lower than that of DF. For each run of ACL, we measure its running time, its percentage of honest nodes captured HC_ϵ and its precision over the nonzero portion of the trust ranking – in other words, at HC_ϵ recall. Then we compare its precision with that of DF at recall HC_ϵ for that run. We report on how the two algorithms compare to each other when ϵ is raised enough to make ACL match DF's running time. From there, we extrapolate on what would happen on larger graphs.

c) Obtaining Multiple Sources from One Honest s

One pitfall of using only a single source node s is that it may not be able to effectively distinguish nodes outside of its community from nodes in the Sybil region, thanks to the fact that the communities are also separated by sparse cuts. As such, we look into possibilities for a source node s to pick multiple other sources in a way where 1) they are all likely to be honest, and 2) combining the results of these sources gives results more

robust than those generated from s alone. Ideally, by utilizing this type of approach, the source node s could leverage the community sub-structure of the social graph to pick other honest sources throughout the graph's various communities, correctly scoring a wider range of honest nodes highly in the ranking. This idea is not new – algorithms such as Gatekeeper [24] have similar source distribution mechanisms in place, and Cao et al. [10] report about using manual verification to distribute their sources – but the technical workings of such mechanisms vary from algorithm to algorithm. Therefore, we wish to isolate ideas that work well specifically for DownhillFlow.

We investigate the following methods for a source node s (chosen randomly from the honest region, under the constraint that s is not within distance 2 of the Sybil region) to determine multiple other, trusted sources:

- Top 10 positions of the trust ranking t.
- Top 10 positions of the trust ranking t, subject to the condition that the nodes are spread out across different distances. In this case, we force the nodes to be distributed as uniformly as possible from distances 1 to 5.
- Top 10 positions of the trust ranking t such that all nodes are of distance d or greater from s. We test d = 4, 5.
- Top 10 positions of the trust ranking t such that all nodes are of distance d or greater from each other. We test d = 4, 5.
- Ten random nodes from the top k percent of the trust ranking t, chosen uniformly. We test k = 1%, 10%.

We also measure the following for comparison purposes:

- Ten randomly determined honest nodes.
- One node running ACL.

These tests were all ran on the epinions graph under a random attack with $p = 0.1$. The value of ϵ used for ACL was set to 10^{-6}, as with the experiments in IVa.

V – RESULTS

a) Robustness

Figure 1 shows the precision-recall curves for DownhillFlow on all five graphs under the random attack model. On all graphs but DBLP, DownhillFlow generates good results even at $p = 0.09$. Table 4 gives the exact precision values for DownhillFlow on epinions at 50%, 90%, and 95% recall. We also report the corresponding results for ACL in Figure 2 and Table 5, respectively. The results obtained for ACL are consistent with those obtained by Alvisi in [3]. At 95% recall, ACL continues to identify honest nodes with high precision even under stronger attacks.

Figure 3 shows precision-recall curves comparing DownhillFlow's results with those obtained by ACL and SybilGuard on epinions and DBLP at $p = 0.01, 0.05$. We obtained similar results in all other iterations of the experiment, with exception of the strong fixed attack. In all cases but that case, for all graphs, ACL's results are more robust than those of DownhillFlow, which in turn are more robust than those of SybilGuard. Recall that

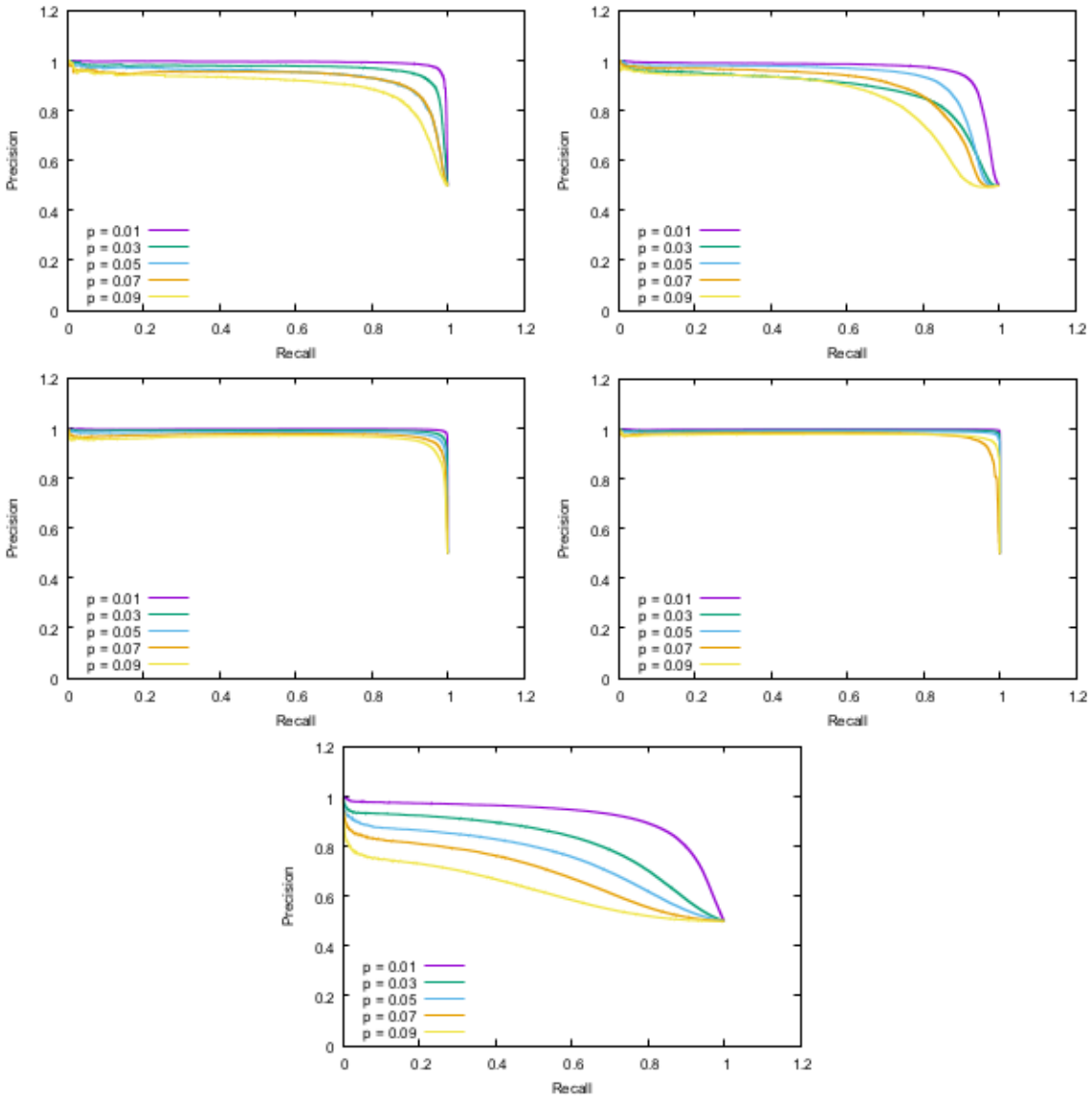


Figure 1: Precision-recall curves for DownhillFlow on epinions (top left), Twitter (top right), Slashdot (middle left), LiveMocha (middle right), and DBLP (bottom) graphs.

	50%	90%	95%
$p = 0.01$	0.996	0.987	0.980
$p = 0.03$	0.979	0.952	0.919
$p = 0.05$	0.962	0.887	0.801
$p = 0.07$	0.955	0.888	0.806
$p = 0.09$	0.929	0.811	0.684

Table 4: Precision of DownhillFlow on epinions by attack strength under the random attack model at 50%, 90% and 95% recall. A value of 0.5 corresponds to a random ranking.

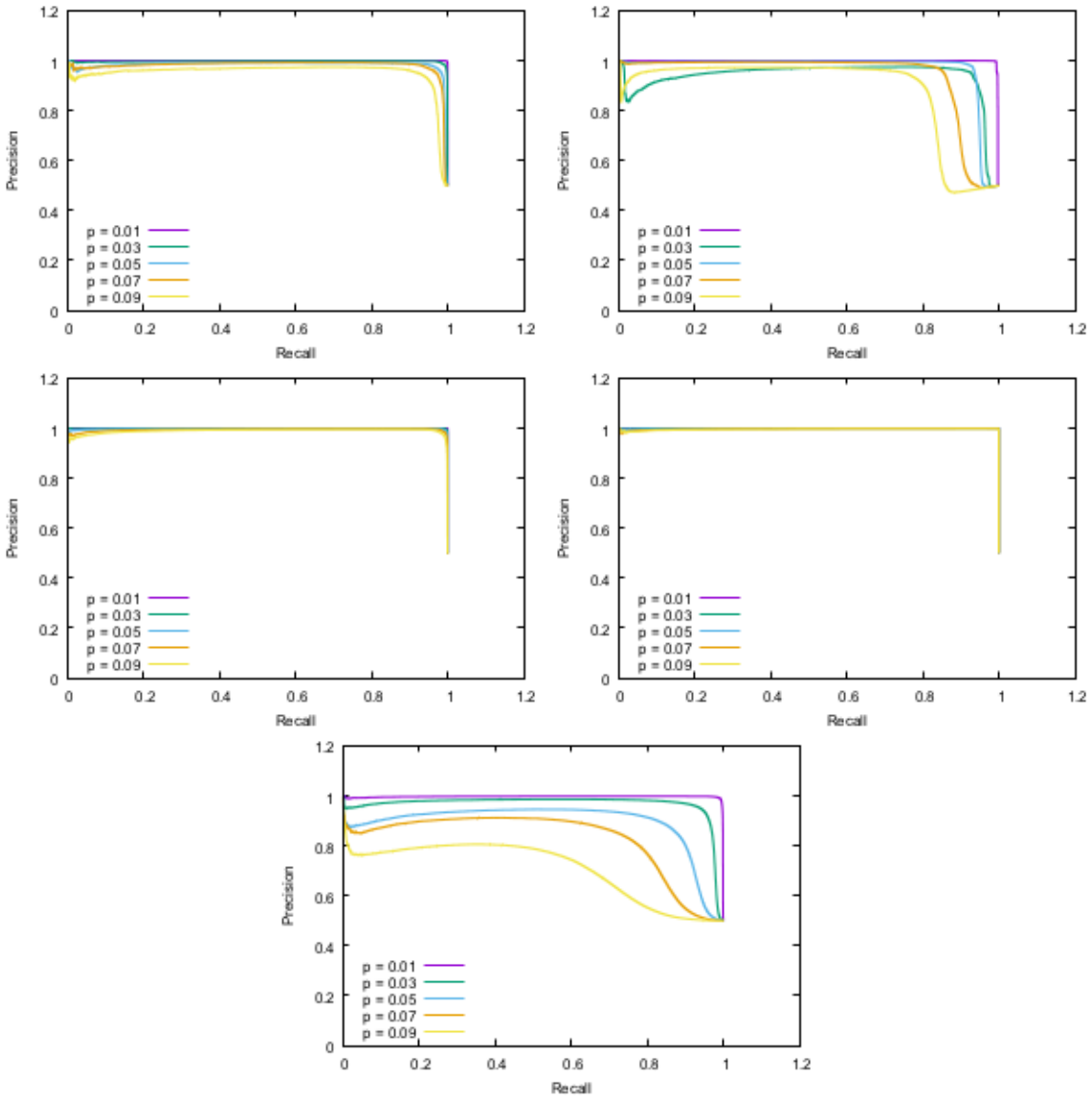


Figure 2: Precision-recall curves for ACL on epinions (top left), Twitter (top right), Slashdot (middle left), LiveMocha (middle right), and DBLP (bottom) graphs.

	50%	90%	95%
$p = 0.01$	1.000	1.000	1.000
$p = 0.03$	0.998	0.998	0.998
$p = 0.05$	0.992	0.989	0.983
$p = 0.07$	0.991	0.986	0.968
$p = 0.09$	0.971	0.961	0.922

Table 5: Precision of ACL on epinions by attack strength under the random attack model at 50%, 90% and 95% recall. A value of 0.5 corresponds to a random ranking.

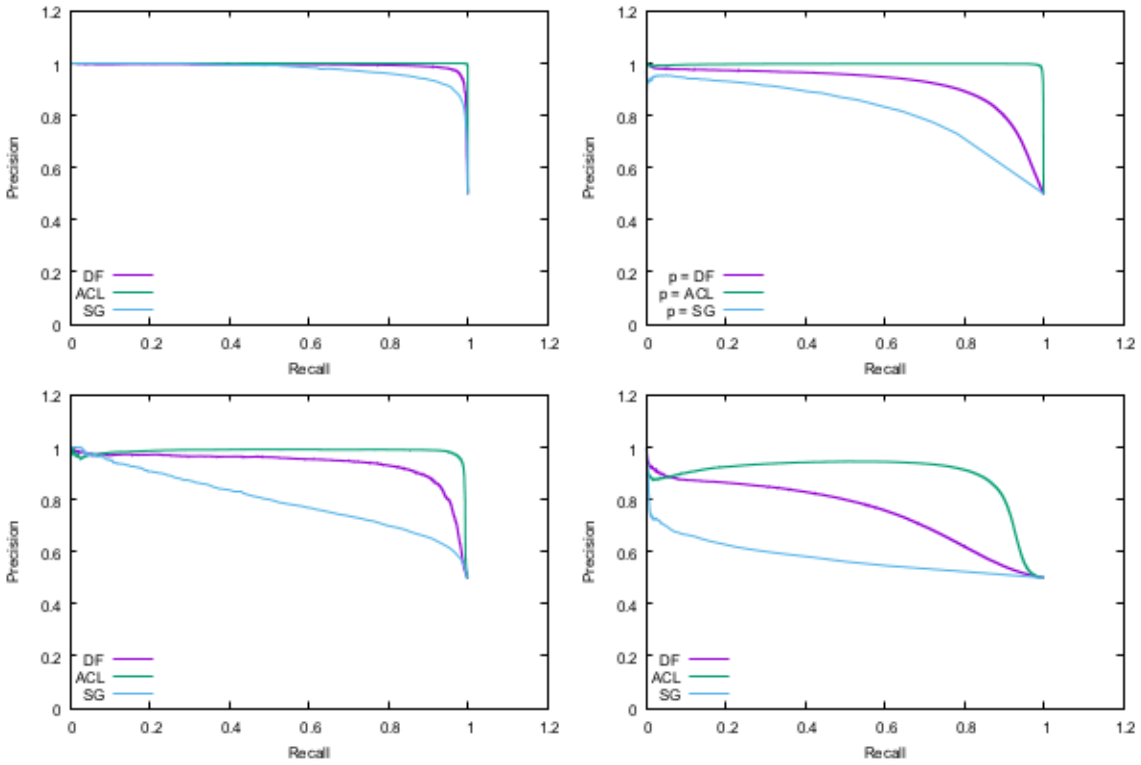


Figure 3: Precision-recall curves comparing ACL, DownhillFlow and SybilGuard. The tests shown are epinions (left) and DBLP (right) at $p = 0.01$ (top) and $p = 0.05$ (bottom). SybilGuard degrades much more quickly than DownhillFlow under stronger attacks.

for SybilGuard, we report results on the preprocessed input graphs, compared to the raw input graphs for DownhillFlow and ACL. Perhaps somewhat counter-intuitively, under the strong fixed attack, SybilGuard overtakes both DownhillFlow and ACL on the twitter and DBLP graphs. The results for LiveMocha are also mixed, as SybilGuard stays slightly ahead of ACL for most of the trust ranking, but its precision decreases rapidly at around 60% recall. DownhillFlow and ACL's results continue to remain mostly correlated – whichever one is better varies depending on the graph. However, it must also be noted that under many of the strong fixed attacks, almost no trust rankings were noticeably above random, except on LiveMocha. Also note that the fixed attack model was the model SybilGuard was originally studied under in [29].

The preprocessing step for SybilGuard proved to be problematic in several iterations of the experiment: we found that for strong enough attacks, it was impossible to find ten source nodes not within distance 2 of the Sybil region after preprocessing. This did not happen with any of the raw input graphs, and as such DownhillFlow and ACL did not suffer from this problem. Note that even when suitable source nodes were possible to pick after SybilGuard's preprocessing step, which was the case in all iterations for both epinions and DBLP as well as all iterations for which $p = 0.01$, DownhillFlow's results were still more robust than those of SybilGuard in all iterations besides the strong fixed attacks. Additionally, the results for SybilGuard on other graphs besides epinions and DBLP were qualitatively very similar to those for epinions on DBLP under all attack strengths, usually being satisfactory until $p = 0.03$ but experiencing a sharp drop at the $p = 0.05$ level regardless of how the distance was constricted. While the values we obtained in section IV support the result of Mohaisen et al. in [19], this result seems to provide an argument that not only does preprocessing rule out nodes, it can actually serve to endanger the nodes that do remain in the graph after the preprocessing step is finished.

For a full collection of precision-recall curves for all iterations of the experiment (taking graph, attack type/strength and protocol as parameters), consult Appendix A. Both graphs comparing by values of p and graphs comparing by protocol are included there.

b) Running Time

Table 6 shows the average running time across the ten source nodes for DownhillFlow

and ACL on epinions and DBLP, respectively.

	epinions	DBLP
DF	0.099 s	1.263 s
ACL, $1.0*10^{-7}$	---	45.6 min +
ACL, $2.0*10^{-7}$	---	9.5 min +
ACL, $4.5*10^{-7}$	---	17.313 s
ACL, $1.0*10^{-6}$	4.4 min +	4.397 s
ACL, $2.0*10^{-6}$	54.165 s +	1.920 s
ACL, $4.5*10^{-6}$	0.405 s	1.585 s *
ACL, $1.0*10^{-5}$	0.099 s *	0.742 s *

Table 6: Running time of DownhillFlow vs. ACL on epinions and DBLP graphs (average, 10 runs). The speed/ϵ tradeoff is shown for ACL and compared to DownhillFlow (no parameters), starting at the value used in a). Values marked with + show ACL captures all honest nodes, i.e. $t_v > 0\ \forall v \in H$. Values marked with * begin from the highest ϵ where ACL's running time is slower.

For DownhillFlow, the running time in our implementation cleanly follows its asymptotic bound: $|V| + |E|$ is slightly over 10 times larger for DBLP (1,366,946) than epinions (126,708). When ACL captures the entire honest region, its running time also cleanly follows its asymptotic bound. The running times for DBLP are approximately ten times those of epinions, while the value of ϵ was reduced by a factor of 1/10. When the ϵ value is set high enough for ACL to start missing nodes, however, the running time improves at a rapid rate, faster than its asymptotic bound would suggest. Eventually, increasing ϵ enough allows ACL's running time performance to overtake that of DownhillFlow.

So what happens to ACL's results at that point? As Table 6 shows, on both graphs ACL does not match DownhillFlow's speed until well after it begins to miss nodes. We examine ACL's behavior on both graphs individually. Table 7 shows more detailed information for the epinions graph, including the percentage HC_ϵ of the honest region

captured and precision at HC_ϵ recall:

	epinions	# $v \in H$ missed	HC_ϵ	ACL prec. at HC_ϵ recall	DF prec. at HC_ϵ recall
DF	0.099 s	---	---	---	---
ACL, $1.0*10^{-6}$	4.4 min +	0 +	100.00% +	---	---
ACL, $2.0*10^{-6}$	54.165 s +	0 +	100.00% +	---	---
ACL, $4.5*10^{-6}$	0.405 s	926	96.51%	0.648	0.581
ACL, $1.0*10^{-5}$	0.099 s *	8,058 *	69.69% *	0.759 *	0.874 *

Table 7: Running time of DownhillFlow versus ACL on epinions, including the HC_ϵ obtained by ACL under various ϵ values. The two rightmost columns compare DownhillFlow's precision versus that of ACL among the % of the trust ranking recognized by ACL, p = 0.1.

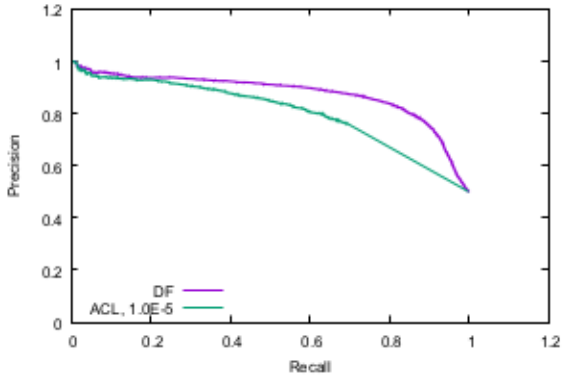


Figure 4: Precision-recall curve of DownhillFlow compared to ACL on epinions ran at $\epsilon = 10^{-5}$, p = 0.1. At 69.69% recall, ACL's curve begins to drop linearly as its results correspond to random.

This seems to indicate a mix of results: DownhillFlow enjoys a significant precision advantage compared to ACL at $\epsilon = 10^{-5}$ among the nonzero portion of ACL's trust ranking, but at the next lower ϵ level, this advantage is lost.

Next we focus on DBLP. Table 8 shows detailed information for the DBLP graph. On DBLP, ACL's decrease in ability to capture honest nodes is more noticeable. At the

point where ACL matches DownhillFlow, ACL only captures 31.10% of the honest region; even at the next lower ϵ level, we have only $HC_{2.0*10^{-6}}$ = 56.66%, while DownhillFlow still obtains slightly better precision. This seems to indicate that

	DBLP	# $v \in H$ missed	HC_ϵ	ACL prec. at HC_ϵ recall	DF prec. at HC_ϵ recall
DF	1.263 s	---	---	---	---
ACL, 1.0*10^{-7}	45.6 min +	0 +	100.00% +	---	---
ACL, 2.0*10^{-7}	9.5 min +	0 +	100.00% +	---	---
ACL, 4.5*10^{-7}	17.313 s	10,156	96.79%	0.580	0.516
ACL, 1.0*10^{-6}	4.397 s	60,590	80.89%	0.690	0.637
ACL, 2.0*10^{-6}	1.920 s	137,410	56.66%	0.757	0.775
ACL, 4.5*10^{-6}	1.585 s *	218,443 *	31.10% *	0.798 *	0.832 *
ACL, 1.0*10^{-5}	0.742 s *	266,348 *	15.99% *	0.828 *	0.845 *

Table 8: Running time of DownhillFlow versus ACL on DBLP, including the HC_ϵ obtained by ACL under various ϵ values. The two rightmost columns compare DownhillFlow's precision versus that of ACL among the % of the trust ranking recognized by ACL, p = 0.03.

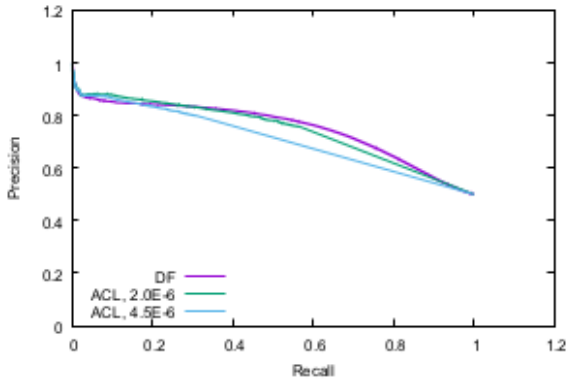


Figure 5: Precision-recall curve of DownhillFlow compared to ACL on DBLP ran at ϵ = 4.5 *10^{-6}, 2.0 * 10^{-6}, p = 0.03.

DownhillFlow's advantage is greater on larger graphs, as the larger the graph, the smaller ϵ required for ACL to capture the complete honest region.

It is worth noting that it is not necessarily a bad thing if ACL fails to capture 100% of the honest nodes – on both graphs, a proper choice of ϵ ($4.5 * 10^{-6}$ for epinions and $4.5 * 10^{-5}$ for DBLP) allows ACL to capture a significant majority of the honest region (>96% in both cases), while still incurring the rapid decrease in running time that occurs when ACL does not capture the whole graph. While it was previously known (and in fact a part of ACL's design) that higher values of ϵ resulted in faster running time and some nodes $v \in V$ having $t_v = 0$, to our knowledge, we are the first to report actual running times showing that ACL can capture a large portion of the honest subgraph while still being fast.

That being said, we next revisit our assumption of a worst-case scenario for DownhillFlow and ask: is DownhillFlow's advantage more significant under a *weak* attack? And if so, how much greater of a proportion of nodes can DownhillFlow rank precisely than is capturable by ACL? Table 9 and Figure 6 show the percentage of nodes recalled by DownhillFlow at 99.00% and 98.00% precision and the precision-recall curves for DownhillFlow and ACL, respectively, under a very weak attack ($p = 0.001$). For ACL, we report the values for $\epsilon = 2.0 * 10^{-6}$ and $4.5 * 10^{-6}$:

Value	% nodes
DF, 0.990 prec.	84.16%
DF, 0.980 prec.	92.53%
ACL, $HC_{2.0*10^{-6}}$	59.85%
ACL, $HC_{4.5*10^{-6}}$	34.69%

Table 9: Comparing the percentage of nodes DownhillFlow can recall at precision 99%, 98% versus the percentage of nodes captured by ACL at $\epsilon = 2.0 * 10^{-6}$, $4.5 * 10^{-6}$, $p = 0.001$. Note that these HC_ϵ values are higher than those for $p = 0.03$ since there are fewer attack edges.

Although we have assumed an unreasonably weak attack, these findings seem to confirm our intuition: when the attack strength is weak enough, DownhillFlow can recall a far greater proportion of the graph at good precision than ACL can capture (at any precision). We have yet to investigate how this effect downscales as the attack strength increases upward back to $p = 0.01 \sim 0.03$.

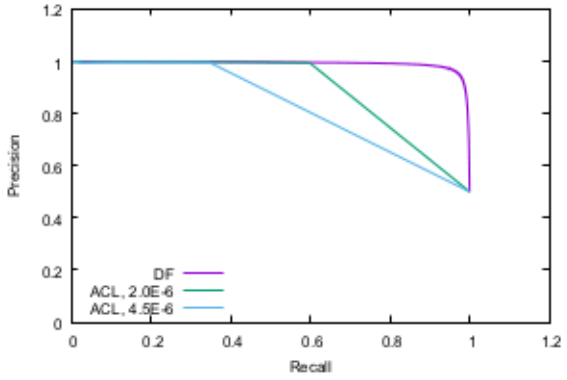
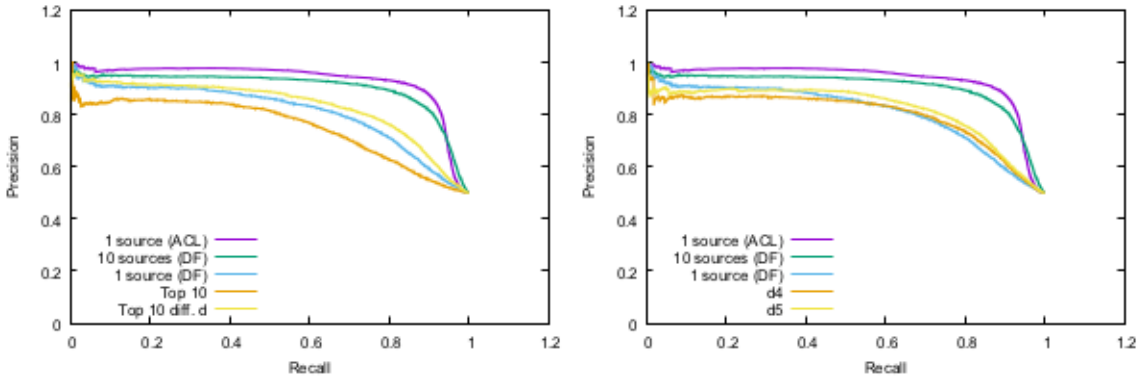


Figure 6: Precision-recall curve for DownhillFlow versus ACL at $\epsilon = 2.0 * 10^{-6}$, $4.5 * 10^{-6}$ for $p = 0.001$.

c) Obtaining Multiple Sources from One Honest s

Figure 7 shows the precision-recall curves for the different methods of obtaining multiple sources from one honest s.



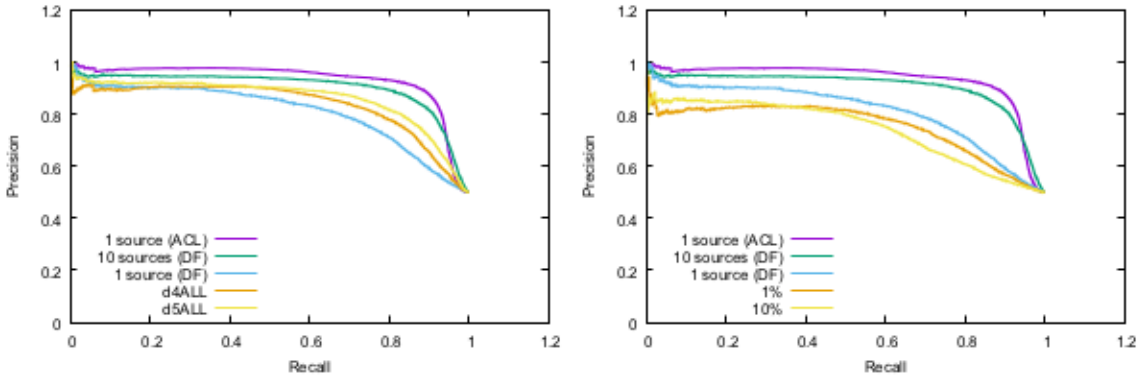


Figure 7: Precision-recall curves comparing various ways of choosing multiple sources from a trust ranking. For comparison, we show the results for one source, ten random sources and one node running ACL on all graphs. The remaining curves are as follows. Top-left: Top 10 (top ten nodes) and top 10 different d (top-ranked nodes at different distances spread uniformly $1 < d_s(v) < 5$); Top-right: $d4$ and $d5$ (top-ranked nodes at distances 4, 5 respectively); Bottom-left: $d4$ALL and $d5$ALL (top-ranked nodes all distances 4, 5 from each other respectively); Bottom-right: 1% and 10% (ten random nodes from the top 1% and 10% of the trust ranking respectively).

The results range from moderately better than with one s (but still slightly worse than random) to noticeably worse. Particularly, selecting the top 10 nodes from the trust ranking and picking 10 nodes randomly from the highest k% of the ranking fared worse off than one node. In the former case, we speculate this is because picking the top 10 nodes with no further processing usually just results in a group of nodes clustered around s. In the latter case, we speculate that due to its randomness, the approach is too susceptible to accepting either 1) "bad" sources that are close or adjacent to the Sybil region, or 2) Sybil nodes themselves.

The most robust results were obtained by selecting nodes from the top of the trust ranking in a way where they had to be distance at least d from each other; this approach both with d = 4, 5 outranked the other approaches. We suspect that out of the approaches we tried,

this approach is the most effective at spreading the source nodes throughout the entire graph, and thus being more likely to place source nodes within different communities.

d) Results Without Degree Normalization

We also tested how DownhillFlow performs without the degree-normalization step at the end as it was not immediately obvious how this step is beneficial to the algorithm. Figure 8 shows the precision-recall curves for DownhillFlow on epinions, without the degree-normalization step. Skipping over the degree-normalization step makes the resulting trust list worse by a significant margin:

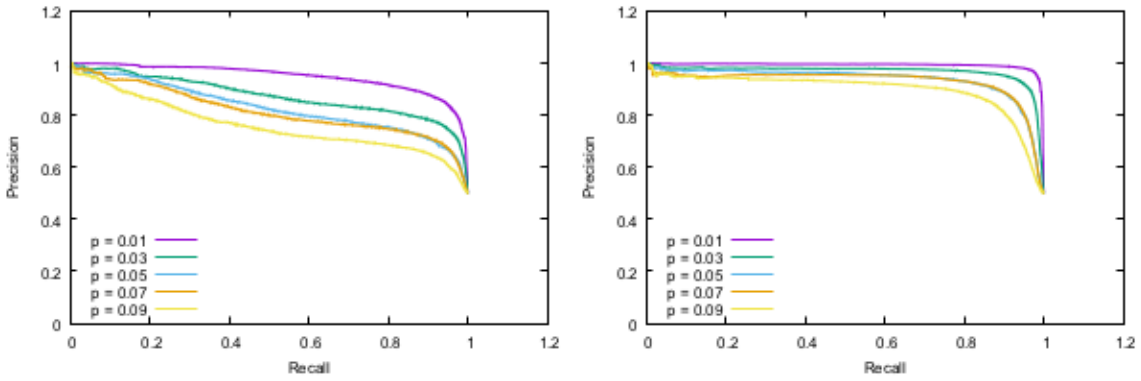


Figure 8: Precision-recall curve for DownhillFlow on epinions without degree-normalization (left). For comparison, the original curve is shown (right).

The reason many random walk distribution-based protocols (such as SybilRank [10] and ACL [3]) normalize by degree is because the stationary distribution π is not a uniform distribution: it is proportional to the degree of the nodes. Thus normalizing by degree eliminates the bias nodes would otherwise receive by having high degree. Since degree normalization helps the robustness of our algorithm, we would thus expect this bias to be present in the non-normalized results: a manual inspection of the trust ranking produced

at $p = 0.09$ confirms this is the case, with higher-degree nodes being concentrated near the top of the ranking and all Sybils within the top 500 positions of the ranking having $\deg(v) > 12$ (and most having $\deg(v) > 30$). We thus speculate that although our algorithm is based around flow, we can still draw connections to random walk theory when investigating its properties, as with the approach discussed in IIIc.

VI – DISTRIBUTED SETTING

We have shown that in the centralized setting, the results obtained by DownhillFlow are a significant improvement over those obtained by SybilGuard. We have also shown that for the results it achieves, DownhillFlow has excellent running time performance, and that its results overtake those of ACL when ACL's error parameter ϵ is pushed such that ACL's speed matches that of DownhillFlow. However, we are still missing one piece of the puzzle: namely, showing how to adapt DownhillFlow to the distributed setting. To get a suitable distributed protocol, we need 1) the source node s to be able to get its trust vector t without knowledge of the full graph topology, and 2) all nodes $v \in V$ to have only $\mathcal{O}(|V|)$ space to work with. The centralized version of the algorithm makes use of the fact that the entity running the algorithm knows (and stores in memory) the full topology of the graph. Therefore, we need a different approach in order to meet these two assumptions. In this section, we discuss possible approaches to this by drawing on results from two other cryptographic fields: accumulators and e-cash.

We first introduce the notion of a cryptographic accumulator. Cryptographic accumulator schemes were first proposed in [7], and allow for a set of input values $\mathcal{X}$ over an input domain $\mathcal{Z}_i$ to be combined, or "accumulated", into a single value $acc_{\mathcal{X}} \in \mathcal{Z}_a$, such that the original set $\mathcal{X}$ cannot be forged or tampered with. This is done using *witnesses*: for each $x_i \in \mathcal{X}$, we can compute a witness w_{x_i}, which a verification algorithm can use to verify $x_i \in \mathcal{X}$. It must be computationally infeasible to compute a witness w_{x_i} for a value $x_i \notin \mathcal{X}$. We provide a simplified definition, taken from [12], which we urge the reader to consult for a more detailed discussion about the properties enjoyed by such schemes and the various accumulator schemes in use today:

Definition 2 (Cryptographic accumulator): A static cryptographic accumulator scheme is a 4-tuple of efficient algorithms (Gen, Eval, WitCreate, Verify) which are defined as follows:

1. Gen(1^k, t): Takes as input k, a security parameter, and t, the maximum size of the set $\mathcal{X}$ supported by the accumulator scheme. If there is no upper bound on the size of $\mathcal{X}$, then $t = \infty$. Returns the private/public accumulator key pair (SK_{acc}, PK_{acc}).
2. Eval$((SK_{acc}^{\sim}, PK_{acc}), \mathcal{X})$: Takes as input (optional) SK_{acc}, PK_{acc} and the set $\mathcal{X}$ of values to be accumulated and returns the accumulated value $acc_{\mathcal{X}}$.
3. WitCreate$((SK_{acc}^{\sim}, PK_{acc}), acc_{\mathcal{X}}, x_i)$: Takes as input (optional) SK_{acc}, PK_{acc} an accumulated value $acc_{\mathcal{X}}$ and a value x_i. Computes and returns a witness w_{x_i} if $x_i \in \mathcal{X}$ and returns false otherwise.

4. Verify($PK_{acc}, acc_{\mathcal{X}}, w_{x_i}, x_i$): Takes as input PK_{acc}, an accumulated value $acc_{\mathcal{X}}$, a witness w_{x_i} and a value x_i. Returns true if w_{x_i} is a witness proving $x_i \in \mathcal{X}$ and false otherwise.

$\widetilde{SK_{acc}}$ is used in the above definition to indicate that the secret key passed as input is optional; henceforth, when an algorithm is called without the secret key, we denote this by using $\emptyset$ in its place. Note that the definition provided above is for a *static* accumulator scheme. This differs from a *dynamic* accumulator scheme in that essentially, the set $\mathcal{X}$ is fixed in place when the Eval algorithm is called. In a dynamic scheme, additional algorithms are provided to allow for addition and deletion of elements to $\mathcal{X}$ and to update the witnesses w_{x_i} when a value $x_j \neq x_i$ is added to or deleted from $\mathcal{X}$. We focus on the static setting here as dynamic schemes are harder to construct and we do not need the added functionality they provide. Note also that we do not instantiate any cryptographic accumulator schemes in this thesis. We do, however, assume the existence of a black-box static, unbounded accumulator scheme over an input domain $\mathcal{Z}_i$ such that there exists a hash function $\mathcal{H}: \mathcal{T} \rightarrow \mathcal{Z}_i$ accepting a set of *flow tickets* $\mathcal{T}$, which we define shortly, as its domain.

We can apply accumulators to DownhillFlow by taking advantage of the fact that to know where a node v should push flow, we need only look at the nodes $w \in N(v)$ and their corresponding $d_s(w)$s. Each v makes use of several flow tickets, denoted $\langle v, w \rangle$, indicating it is to push flow to a target node w. Then each v forms a set $ALLT_v$ of all

tickets going in and out from itself. We can then store the entire set $ALLT_v$ using one value acc_v. v forwards this information to s by composing another set, VER, of (ticket, witness) pairs the witness of which indicates that the ticket $\langle i, j \rangle$ is a part of the value acc_i. Note we have either i or $j = v$. s can use the witnesses from the VER set to verify that the tickets it receives are legitimate, and thus can update v's trust accordingly. s maintains a hash table $\mathcal{M}$ linking nodes to (trust value, acc, $|ALLT|$) tuples, and thanks to the space-saving property accumulators provide, this all takes only $\mathcal{O}(|V|)$ space. Note also that s must instantiate its own $ALLT$ and acc values and propagate its own tickets for use by its neighbors.

Before proceeding further, we must note a contingency. First, degree-normalization here does not function the same way as with the centralized version of the algorithm: s stores the base t_v and subsequently calculates the final trust value by dividing by $|ALLT_v|$ as opposed to $\deg(v)$. It is possible to mitigate this by requiring nodes to keep track of empty tickets for their neighbors at the same level. However, we then run into a different problem: the adversary has full control over the Sybil region S and can thus arrange it such that S has no nodes at the same BFS level with each other, making $|ALLT_v| = \deg(v)\ \forall v \in S$! We thus conclude that it is sufficient to use $|ALLT_v|$ for all nodes (which can only increase the trust).

The protocol is shown below in algorithms 4.1 (for a source node s) and 4.2 (for a non-source node v):

Algorithm 4.1: DF1_DIST (source node s)

// Initialization
1 $\mathcal{M} \leftarrow \emptyset$
2 $ALLT \leftarrow \emptyset$
// Compute entry for s and push first round of tickets
3 **for each** higher-level node v **do:**
4 $T_v \leftarrow \langle s, v \rangle$
5 $ALLT \leftarrow ALLT \cup \{\mathcal{H}(T_v)\}$
6 $acc \leftarrow \text{Eval}((PK_{acc}, \emptyset), ALLT)$
7 **for each** higher-level node v **do:**
8 **send** $\langle T_v, acc \rangle$ **to** v
// Register entry for s in the hash table $\mathcal{M}$
9 $\mathcal{M}.\text{add}(s, (1, acc, |ALLT|))$
// Receive VER messages from the rest of the graph
10 **for each** $\langle VER_v \rangle$ from $v \neq s$ **do:**
// Initialize info for v
11 $t_v \leftarrow 0$
12 $ALLT_v \leftarrow \emptyset$
// Add flow for each incoming ticket that is verified
13 **for each** $(T := \langle i, j \rangle, w) \in VER_v$ **do:**
14 **if** $j = v$ **and** Verify(PK_{acc}, acc_i, w, T) = true **then** $t_v \leftarrow t_v + \frac{t_i}{|ALLT_i|}$
15 $ALLT_v \leftarrow ALLT_v \cup \{\mathcal{H}(T)\}$
// Compute v's acc value and store entry for v in $\mathcal{M}$
16 $acc_v \leftarrow \text{Eval}((PK_{acc}, \emptyset), ALLT_v)$
17 $\mathcal{M}.\text{add}(v, (t_v, acc_v, |ALLT|_v))$
// Repeat step 10 until a suitable amount of nodes are accepted. s may terminate the
// protocol at its own discretion.

This approach carries a variety of limitations, which we proceed to discuss. The first is that even though the protocol is executed in a distributed manner, the flow computation is all done locally at s (albeit requiring only $\mathcal{O}(|V|)$ space). This is not suitable for a distributed protocol: it would be more intuitive if the computation required to calculate the flow could be distributed throughout the graph as well, in a way where nodes have a proof of work which could be sent back to s.

Algorithm 4.2: DF1_DIST (non-source node v)

```
    // Initialization
1   ALLT ← ∅
2   VER ← ∅
    // Receive tickets from lower-level nodes
3   for each ⟨T_u := ⟨u, v⟩, acc_u⟩ from lower-level nodes u do:
4       w_u = WitCreate((PK_acc, ∅), acc_u, H(T_u))
5       VER ← VER ∪ {(T_u, w_u)}
6       ALLT ← ALLT ∪ {H(T_u)}
    // Compute acc, push info to higher-level nodes and to s
7   for each higher-level node w do:
8       T_w ← ⟨v, w⟩
9       ALLT ← ALLT ∪ {H(T_w)}
10  acc ← Eval((PK_acc, ∅), ALLT)
11  for each higher-level node w do:
12      send ⟨T_w, acc⟩ to w
13  send ⟨VER, acc⟩ to s
    // End
```

Another limitation stems from the algorithm's complexity at s. The for loop in step 10 is ran at most $\mathcal{O}(|V|)$ times. The for loop in step 13 is at most d_{max} times, however it is only ran in total at most twice in for each edge in the graph. In step 14, acc_u is obtained from the hash table $\mathcal{M}$, which is $\mathcal{O}(1)$ average-case complexity (but $\mathcal{O}(|V|)$ worst-case). In step 14, the Verify algorithm is called, and step 15 is constant, so assuming constant-time witness verification, as in for instance [20], these loops contribute $\mathcal{O}(|V| + |E|)$ total average complexity. However, step 16 calls the Eval algorithm, and as such, its time complexity at that step is determined by the time complexity of the Eval algorithm. Further, it takes $ALLT_v$ as input, which is $\mathcal{O}(|V|)$ size worst-case. However, as shown in section IV, this is not indicative of the average-case complexity: $|ALLT_v|$ is tightly bounded by deg_{max}, which for many social graphs $\ll |V|$. Since the Eval algorithm must be efficient, this contributes $\mathcal{O}(|V|deg_{max}{}^2)$ average complexity, which is the

dominating factor.

A third limitation is that the protocol assumes each v is aware of which of its neighbors are lower-level vs. higher-level (or same-level). This is in part based upon the result of Bazzi et al. [5], which presents a secure distance-vector routing protocol wherein Sybil nodes generally may not report a lower distance to s than their actual distance. However, that protocol carries a few corner cases where this condition does not hold: for instance, Sybil nodes v can report a lower distance if there exists a closer $u \in S$. It is not clear to us how to circumvent this.

We next investigate solutions to the above limitations by noting that the requirement that Sybil nodes may not create arbitrary flow shares strong similarity with the security requirement of electronic cash. Electronic cash was originally designed by Chaum [9] as a system for electronic payments, and the many schemes in the literature since share a variety of properties. We give an informal description of these properties below, modeled after the categorization of Okamoto et al. [21]:

1. Privacy – Users cannot be identified by their purchases.
2. Security – Users cannot forge or double-spend coins.
3. Offlineness – Users can make purchases directly to merchants without consulting with the bank.
4. Transferability – Users can transfer their coins to other users.
5. Divisibility – A coin C can be divided into multiple, smaller pieces such that any

total value less than C can be obtained.

Since the field's inception, electronic cash systems have been designed with privacy and security in mind first and foremost. The intuition behind why the security property is useful for our purposes is as follows: if we had a suitable scheme, a source node s could start with some amount of currency, say \$1, and begin by distributing it to its neighbors. From there, each v could add up its currency, report to s on how much they have, and continue by transferring it to nodes at the next BFS level. Under a suitable electronic cash scheme, the security property would guarantee mathematically that v could not forge extra currency, thus enabling s to accept v's report as is with no need for further verification. The privacy property, on the other hand, is not useful for our purposes – there is no need for the source node s to be unable to identify where a node v pushed or received its currency from. In fact, we have the opposite scenario: s would be better off having as much information as possible about v's currency. This property is particularly limiting for our purposes as, indeed, practically all existing electronic cash schemes require substantial technical construction to ensure the privacy property is satisfied.

Moreover, while the divisibility property is suitable for the purposes of electronic cash, it is not strong enough for our purposes: in fact, we need for coins C to support arbitrary rational denominations m/n. No existing electronic cash scheme accomplishes this, and constructing one falls outside the scope of this thesis.

We also must talk about transferability. Assuming an offline scheme, this is obviously

necessary, but what is not as obvious is that nodes $v \neq s$ must have the capability to combine multiple coins $C_1, C_2, \ldots$ into one coin C. This is because at each hop, a node u splits its coins based on $\deg(u)$, and as such, with no way of combining coins, the number of coins in circulation grows exponentially by BFS level. This causes issues for both space and time complexity (as each v would need to total up the value of its coins to report to s, which in turn would need to process this information). Under the online setting, the situation is different: a node v simply cashes in its coins at s, at which point s mints them a new coin valued at the total value of v's coins. Thus neither transferability (nodes push their currency by spending, not transferring) nor the requirement that coins can be combined are needed. However, this creates a bottleneck at s, and indeed, this bottleneck is widely cited in the literature as a compelling reason electronic cash systems should aim for offlineness.

We further continue this discussion in section VII.

VII – OPEN PROBLEMS AND FUTURE WORK

We have experimentally demonstrated the effectiveness of DownhillFlow in the centralized setting, both in terms of its results and its running time. We have also made progress towards realizing an implementation of DownhillFlow in the distributed setting. However, we are not quite fully there: namely, our implementation of the distributed algorithm using accumulators is for the DF1 version of the algorithm (refer to section III), not DF3. While DF1 was showed to obtain promising results in initial testing, its

results were worse than those of DF3 and we have not tested it under the same level of experimental rigor we have done for DF3. To bridge this last gap, we need a way of assigning the tokens used by DF3 in a distributed, Sybil-resilient manner. Recall that the tokens are numeric values that are unique and satisfy the condition that if $u, v \in V$ with $d_s(u) > d_s(v)$, we must have $token_u > token_v$. One idea may be to simply have each v coordinate with each of their same-level neighbors $w \in N(v)$ to agree on a random orientation of each shared edge vw: while this does not explicitly assign numeric tokens, it may be possible this approach would be effective in randomly ordering the nodes at an individual BFS level, the same result provided by the tokens.

We also have not showed the robustness of DownhillFlow's results theoretically. While this does not take away from the fact that its results as is serve as a very good heuristic for computing trust vectors, we must take caution: lack of a mathematical foundation can open the protocol to theoretical corner cases that experimental testing may not have accounted for. Mislove's community detection [18] serves as an example of this, as while it was heuristically shown to obtain robust results on social networks in multiple studies (for instance [3, 10]), attacks exist that can cause it to deterministically admit every node in the Sybil region [2]. Moreover, understanding the mathematical foundation of the algorithm is the first step towards refining it into something more theoretically sound – it may, for instance, be possible that some improvements that would appear minor could be technicalities that can serve to mathematically guarantee "good" results.

We chose SybilGuard as the distributed protocol to compare DownhillFlow against; however, there is at least one other option, namely SybilLimit [28]. SybilLimit was introduced by Yu et al. as an improvement over SybilGuard, and was shown by Alvisi et al. [3] to achieve results more robust than (but qualitatively similar to) most existing random walk-based protocols. Recall that the major difference proposed by SybilLimit over SybilGuard is that rather than running one, long random route of length l for each node, it runs many iterations of the random routes, each with shorter length w and different routing tables. While the results demonstrated for SybilLimit in [3] look promising for our algorithm from a qualitative standpoint, a particularly notable instance of this being the DBLP graph with $p = 0.01$, it is yet to be determined how SybilLimit performs in our experiments when tested with the same rigor.

While we combined our results in section V using the average, it is not known if this is actually the best way to do so: given how DownhillFlow works, putting the results together in a different way may lead to more robust results. In our initial testing, it appeared that actually, removing the highest k trust values for each node and then taking the average led to higher precision at 50% recall (but the precision dropped off steeply afterward). We suspect this is because this enforces the notion that a node v must have high trust from *many* nodes, not just one, to rank highly in the overall ranking. This stops Sybil nodes from being highly ranked just by sharing, for instance, one attack edge close to a source. Furthermore, it is not known whether this type of approach could be generalized to other algorithms such as ACL.

Although we would need further guarantees for our purposes, constructing a rational-valued electronic cash scheme is an interesting problem even when viewed entirely on its own. While many schemes in the literature achieve divisibility, a core foundation of all of these schemes since electronic cash's introduction in [9] is the principle that each coin be given a fixed, constant value, and the insight that this allows coins to be minted and spent in a disjoint, incremental manner. As such, the schemes that achieve divisibility largely do so by requiring the existence of a minimum "base" value for which coins can be broken into. For instance, in [21], divisibility is achieved using a hash tree that splits the values of coins in half repeatedly; however, without a base value (i.e., \$0.01), this approach cannot yield exact results. The scheme in [8] supports withdrawals of size 2^l in a way that requires only $\mathcal{O}(l + k)$ time/space, where k is a security parameter. But in this scheme, each individual coin withdrawn still has a fixed value.

To give an example of why this is insufficient for DownhillFlow's purposes, we need only note that on the epinions graph at p = 0.01, DownhillFlow computes trust values for some nodes on the order of 10^{-10}. DBLP fares even worse, at 10^{-14}. That means that to achieve accurate results, we would need to process over one hundred trillion coins, which is clearly not viable.

Last, we discuss potential applications of DownhillFlow based on its running time. In section V, we showed that DownhillFlow ran very quickly, averaging 0.099 seconds for epinions and 1.263 seconds for DBLP. We also demonstrated that when ACL's error

paremeter ϵ was set such that it ran at similar speeds, DownhillFlow's results overtook those of ACL both in terms of robustness and the percent of honest nodes captured. However: is running time actually a significant advantage in practice?

To look into this, we extrapolate towards what might happen on even larger graphs. We consider the Tuenti graph [26] ($|V| \approx$ 11M, $|E| \approx$ 1.422b) as an observation point as there exist Sybil defense protocol studies [10] that have reported measurements of running times on this graph. Some social graphs, however, are even larger, for instance sinaweibo ($|V| \approx$ 20M), friendster ($|V| \approx$ 65M) and the complete Facebook graph [16] (if we estimate based on the number of users, we have $|V| >$ 2.0b!). For the Tuenti graph, using the asymptotic bound for DownhillFlow, we could expect a running time of roughly 22.1 minutes ($|V| + |E| \approx$ 1.433b). For ACL, on the other hand, an even lower value of ϵ may be required in order to capture the entire honest region of the graph. Based on the running times for epinions and DBLP, setting $\epsilon = 10^{-8}$ (i.e. reducing by another factor of 10) may yield running times of over 45.6 min * 10 ≈ 7.5 hours. This is already an improvement over SybilRank in [10], which claimed a (parallelized) running time of over 20 hours on Tuenti. Since our implementations of DownhillFlow and ACL were not parallelized, it is likely the actual time improvement of both of these algorithms on larger graphs is greater.

However, the people maintaining such graphs are likely social network operators, whose systems have very large live time. As such, they would not be severely time constrained

and would most likely be better off investing in the costly algorithms to get the most precise results. Social network operators could also take advantage of parallelization to speed up costly algorithms such as ACL. So, if not larger graphs, where to go?

We propose that the most suitable application for DownhillFlow is in *dynamic networks*. Such networks are not likely to be too large, and even for DBLP DownhillFlow finishes in slightly over a second. Thus it does not need any added mechanisms to support addition/removal of nodes – the network operator can simply run the protocol and obtain a near-instant assessment of trust for the network at that time. Other suitable applications most likely include hypothetical scenarios where a TTP wishes to generate personalized on-the-fly trust rankings for a node to use: we can, for instance, imagine a file-sharing system where a TTP keeps track of who trusts who, but provides no other services. In that way, a node v could call on the TTP for a personalized snapshot of "good" nodes for themselves before deciding which to work with. However, the number of nodes with the desired file will be $\ll |V|$. As such, both speed and high inclusivity are needed.

Another scenario where high inclusivity is needed is, in fact, in online content voting. As discussed previously in section II, there exists a protocol, SumUp [25], for Sybil defense in this context, and it is not clear whether DownhillFlow or SumUp obtains better results for it, especially since both are based around flow. However, one thing that is apparent is that even though SumUp is claimed to be adaptable to the distributed setting, it isn't under our restrictions: the method shown for this is simply for the source node s to obtain

a trace of the entire graph. As such, a more in-depth investigation may prove fruitful.

VIII – CONCLUSION

In this thesis, we presented the Sybil defense algorithm DownhillFlow and demonstrated the robustness of its results experimentally. We took steps towards implementing DownhillFlow in the distributed setting, showing a concrete approach based around a black-box cryptographic accumulator with weak assumptions and a link to the problem of constructing a non-private, rational-valued electronic cash scheme. We showed that DownhillFlow's results were more robust than those of SybilGuard, one of the few Sybil defense protocols in circulation today that retains the quality of being distributed. We also showed results for ACL very similar to those obtained by Alvisi et al. in [3], which were more robust by a slight margin than those of DownhillFlow.

We also analyzed DownhillFlow's running time versus that of ACL, a community detection algorithm shown to obtain near-optimal trust ranking on several social graphs, and showed that not only was DownhillFlow significantly faster without measures to terminate ACL early, but that when ACL's error parameter ϵ was adjusted enough to make it match DownhillFlow's speed, DownhillFlow's results were more robust both in terms of precision/recall and the proportion of honest nodes recognized. This effect was most noticeable on larger graphs and under weaker attacks – for DBLP under a weaker attack, DownhillFlow identified 92.53% of the honest region at satisfactory precision, while ACL was only able to capture 31.10% of the honest region at all.

To our knowledge, we are the first both to 1) investigate Sybil defense protocols based around flow, and 2) report concrete running time measurements showing ACL's robustness/speed tradeoff within the context of Sybil defense. Based on DownhillFlow's near-instant running time, we propose dynamic networks as its most promising application.

REFERENCES

[1] Albert, Réka, and Albert-László Barabási. "Statistical mechanics of complex networks." *Reviews of modern physics* 74, no. 1 (2002): 47.

[2] Alvisi, Lorenzo, Allen Clement, Alessandro Epasto, Silvio Lattanzi, and Alessandro Panconesi. "Communities, random walks and social Sybil defense." Technical Report TR-13-04, UTCS, 2013. http://wwwusers.di.uniroma1.it/~epasto/papers/sybil−tr.pdf.

[3] Alvisi, Lorenzo, Allen Clement, Alessandro Epasto, Silvio Lattanzi, and Alessandro Panconesi. "Sok: The evolution of sybil defense via social networks." In *Security and Privacy (SP), 2013 IEEE Symposium on*, pp. 382-396. IEEE, 2013.

[4] Andersen, Reid, Fan Chung, and Kevin Lang. "Local graph partitioning using pagerank vectors." In *Foundations of Computer Science, 2006. FOCS'06. 47th Annual IEEE Symposium on*, pp. 475-486. IEEE, 2006.

[5] Bazzi, Rida A., Young-ri Choi, and Mohamed G. Gouda. "Hop chains: Secure routing and the establishment of distinct identities." *Lecture notes in computer science* 4305 (2006): 365.

[6] Bazzi, Rida A., and Goran Konjevod. "On the establishment of distinct identities in overlay networks." *Distributed Computing* 19, no. 4 (2007): 267-287.

[7] Benaloh, Josh, and Michael De Mare. "One-way accumulators: A decentralized alternative to digital signatures." In *Workshop on the Theory and Application of of Cryptographic Techniques*, pp. 274-285. Springer, Berlin, Heidelberg, 1993.

[8] Camenisch, Jan, Susan Hohenberger, and Anna Lysyanskaya. "Compact E-Cash." In *Eurocrypt*, vol. 3494, pp. 302-321. 2005.

[9] Chaum, David, Amos Fiat, and Moni Naor. "Untraceable electronic cash." In *Proceedings on Advances in cryptology*, pp. 319-327. Springer-Verlag New York, Inc., 1990.

[10] Cao, Qiang, Michael Sirivianos, Xiaowei Yang, and Tiago Pregueiro. "Aiding the detection of fake accounts in large scale social online services." In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pp. 15-15. USENIX Association, 2012.

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

[12] Derler, David, Christian Hanser, and Daniel Slamanig. "Revisiting Cryptographic Accumulators, Additional Properties and Relations to Other Primitives." In *CT-RSA*, pp.

127-144. 2015.
[13] Douceur, John R. "The sybil attack." In *International Workshop on Peer-to-Peer Systems*, pp. 251-260. Springer, Berlin, Heidelberg, 2002.

[14] Hanley, James A., and Barbara J. McNeil. "The meaning and use of the area under a receiver operating characteristic (ROC) curve." *Radiology* 143, no. 1 (1982): 29-36.

[15] Kamvar, Sepandar D., Mario T. Schlosser, and Hector Garcia-Molina. "The eigentrust algorithm for reputation management in p2p networks." In *Proceedings of the 12th international conference on World Wide Web*, pp. 640-651. ACM, 2003.

[16] Facebook. https://www.facebook.com

[17] Leskovec, Jure, Kevin J. Lang, Anirban Dasgupta, and Michael W. Mahoney. "Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters." *Internet Mathematics* 6, no. 1 (2009): 29-123.

[18] Mislove, Alan, Bimal Viswanath, Krishna P. Gummadi, and Peter Druschel. "You are who you know: inferring user profiles in online social networks." In *Proceedings of the third ACM international conference on Web search and data mining*, pp. 251-260. ACM, 2010.

[19] Mohaisen, Abedelaziz, Aaram Yun, and Yongdae Kim. "Measuring the mixing time of social graphs." In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pp. 383-389. ACM, 2010.

[20] Nguyen, Lan. "Accumulators from Bilinear Pairings and Applications." In *CT-RSA*, vol. 3376, pp. 275-292. 2005.

[21] Okamoto, Tatsuaki, and Kazuo Ohta. "Universal electronic cash." In *Annual International Cryptology Conference*, pp. 324-337. Springer, Berlin, Heidelberg, 1991.

[22] Online Network Repository. http://networkrepository.com

[23] Stanford Large Network Dataset Collection. https://snap.stanford.edu

[24] Tran, Nguyen, Jinyang Li, Lakshminarayanan Subramanian, and Sherman SM Chow. "Optimal sybil-resilient node admission control." In *INFOCOM, 2011 Proceedings IEEE*, pp. 3218-3226. IEEE, 2011.

[25] Tran, Dinh Nguyen, Bonan Min, Jinyang Li, and Lakshminarayanan Subramanian. "Sybil-Resilient Online Content Voting." In *NSDI*, vol. 9, no. 1, pp. 15-28. 2009.

[26] Tuenti. https://www.tuenti.com

[27] Viswanath, Bimal, Ansley Post, Krishna P. Gummadi, and Alan Mislove. "An analysis of social network-based sybil defenses." *ACM SIGCOMM Computer Communication Review* 40, no. 4 (2010): 363-374.

[28] Yu, Haifeng, Phillip B. Gibbons, Michael Kaminsky, and Feng Xiao. "Sybillimit: A near-optimal social network defense against sybil attacks." In *Security and Privacy, 2008. SP 2008. IEEE Symposium on*, pp. 3-17. IEEE, 2008.

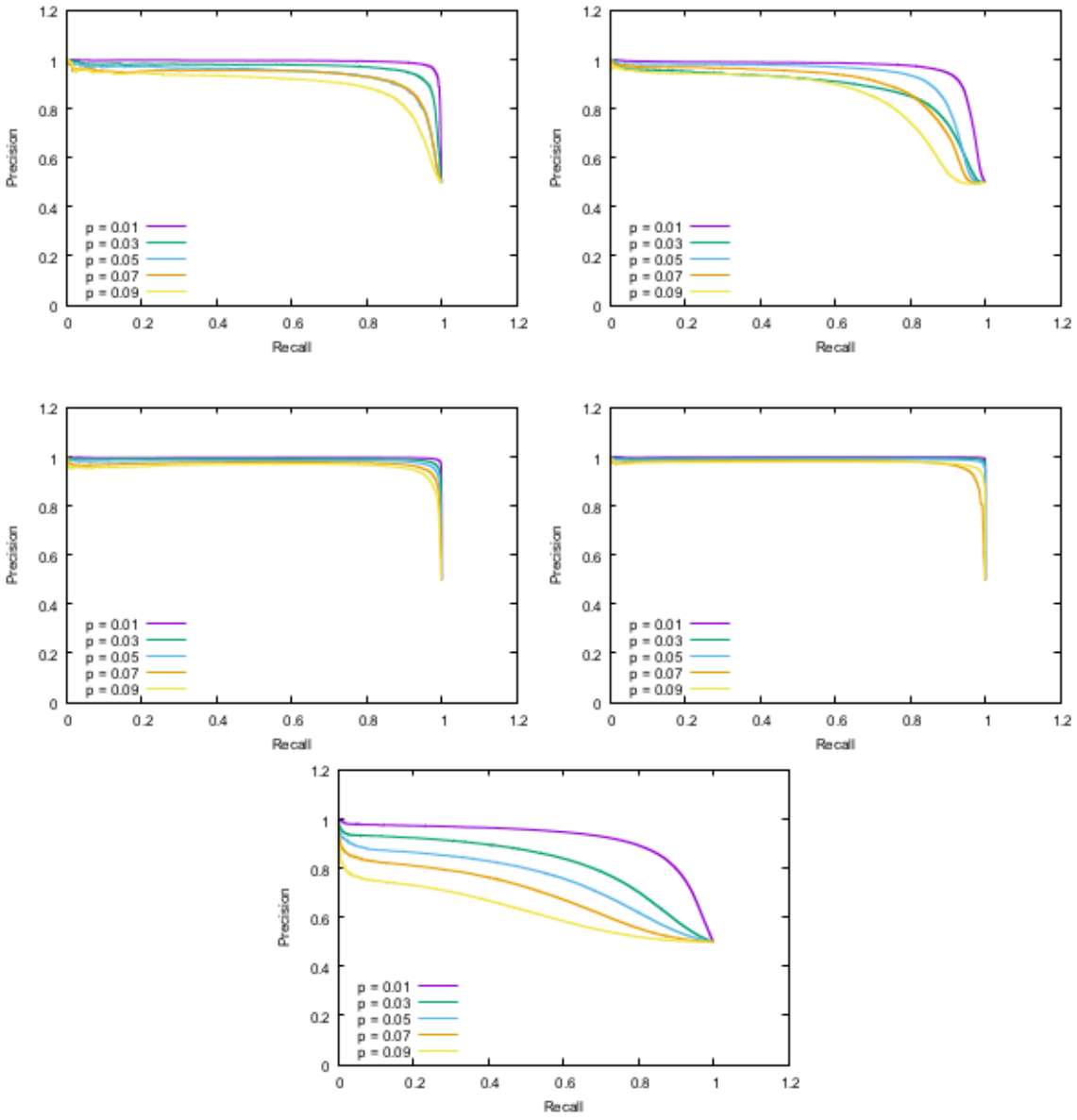
[29] Yu, Haifeng, Michael Kaminsky, Phillip B. Gibbons, and Abraham Flaxman. "Sybilguard: defending against sybil attacks via social networks." In *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 4, pp. 267-278. ACM, 2006.

APPENDIX I

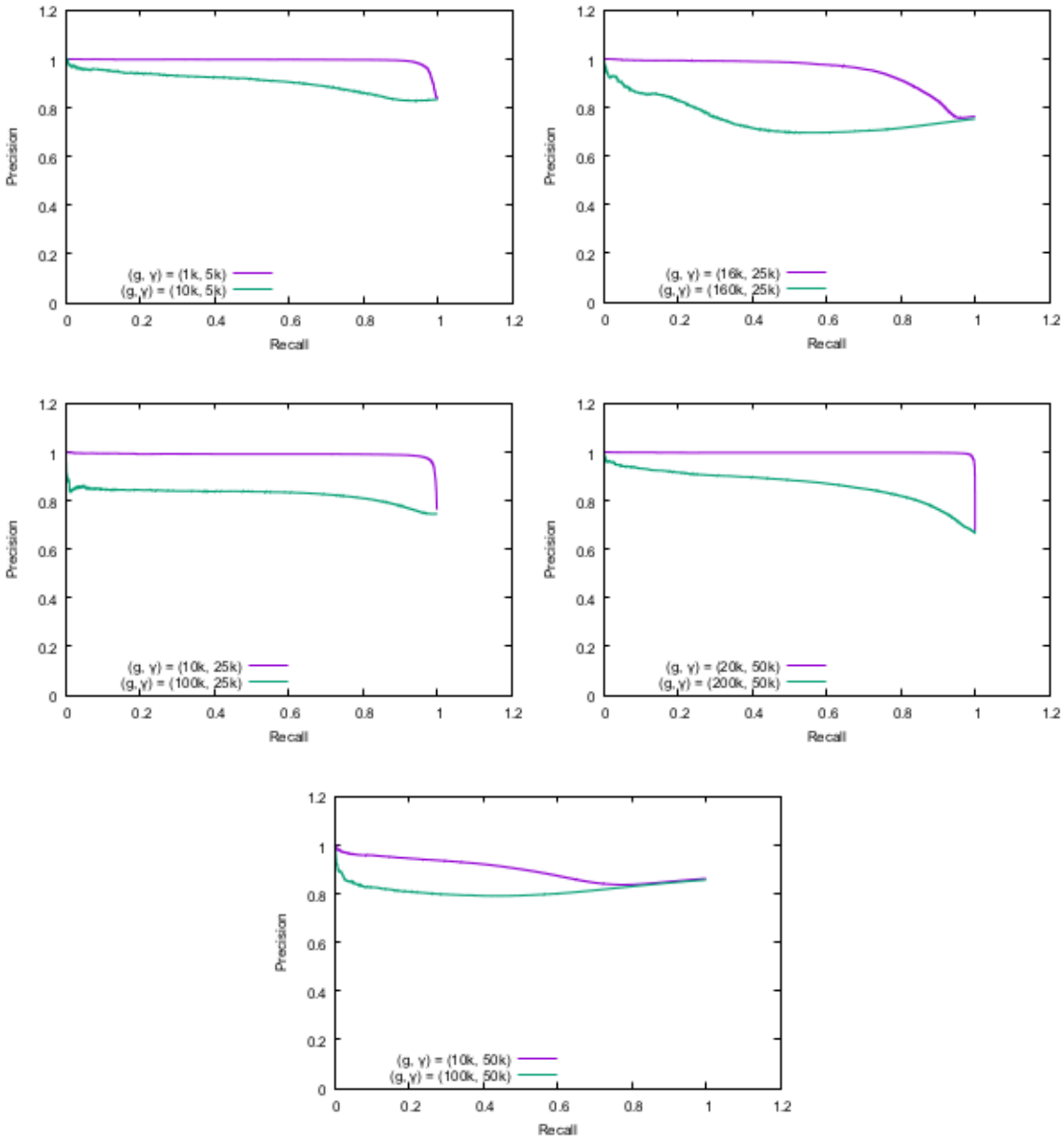
PRECISION-RECALL CURVES GROUPED BY PROTOCOL AND ATTACK STRENGTH, ALL ITERATIONS

In this appendix, all precision-recall curves are presented, organized as in V: epinions (top-left), twitter (top-right), Slashdot (middle-left), LiveMocha (middle-right), DBLP (bottom). Entries 1 ~ 6 are sorted by protocol and compare attack strengths against each other: the DownhillFlow, ACL and SybilGuard protocols are all shown. The remaining entries, 7 ~ 13, are sorted by attack type and compare the protocols against each other: the random attack with $p = 0.01 \ldots 0.09$ and the fixed attack with weak and strong parameters are shown.

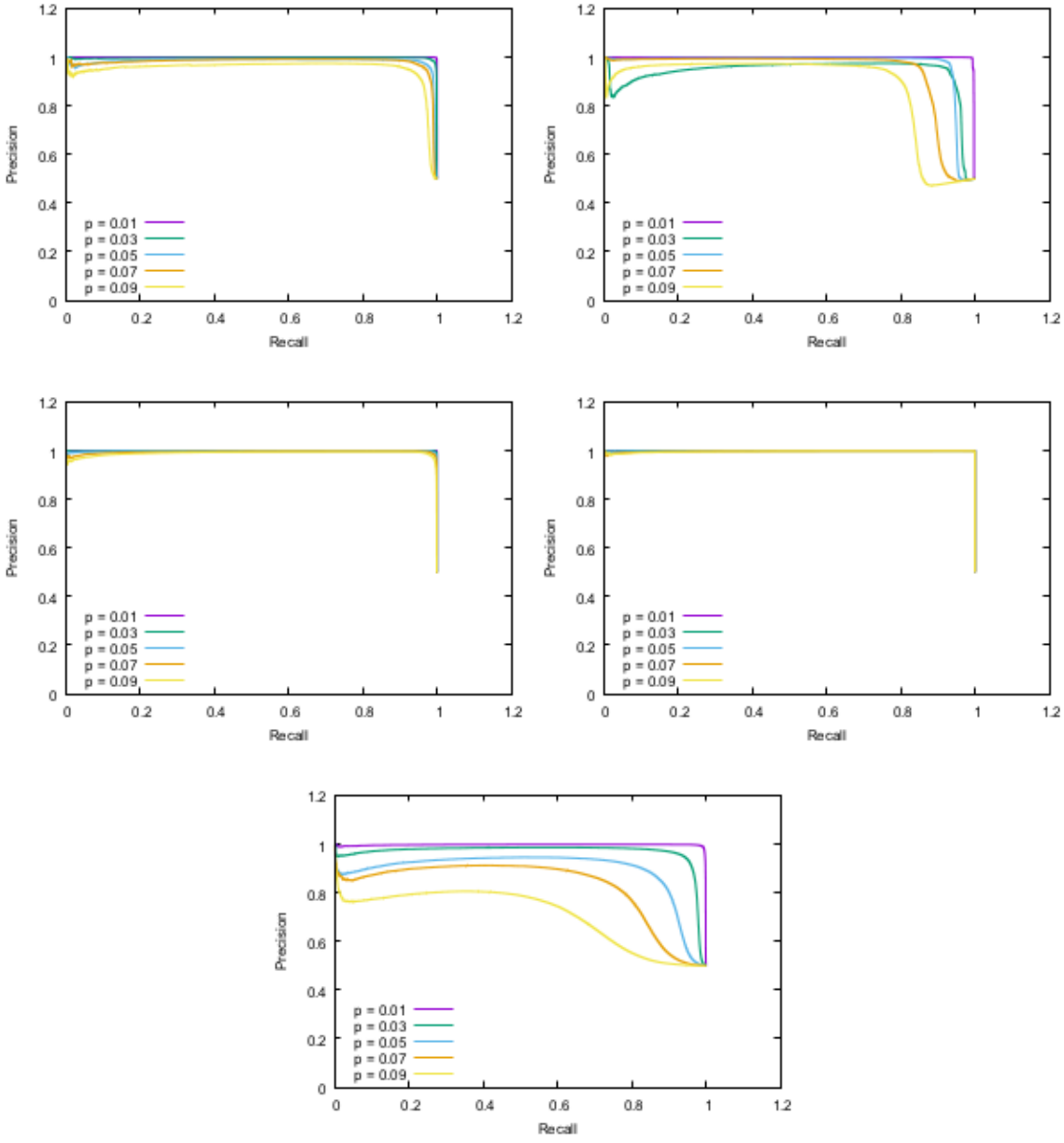
1) DownhillFlow, random model



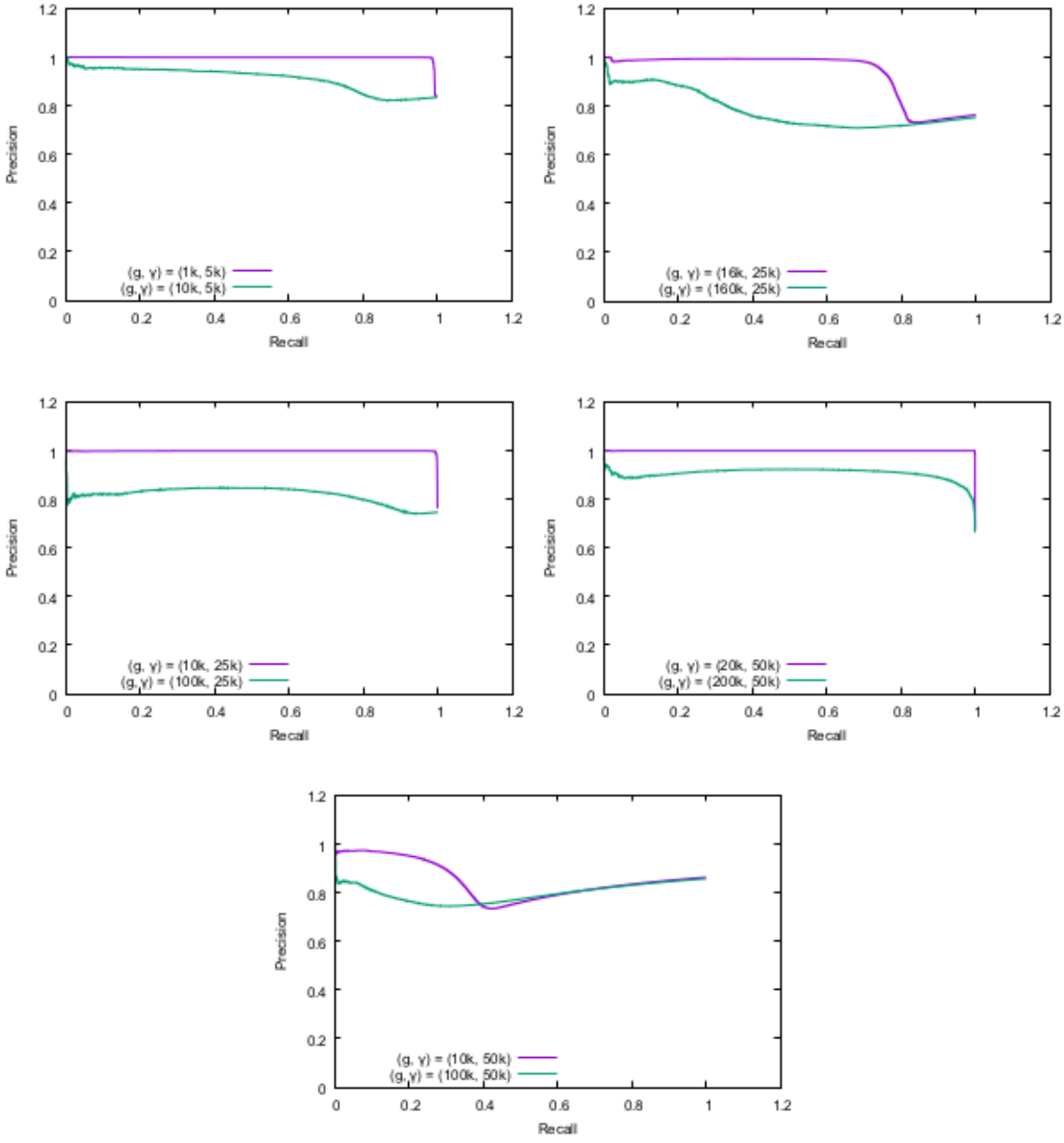
2) DownhillFlow, fixed model



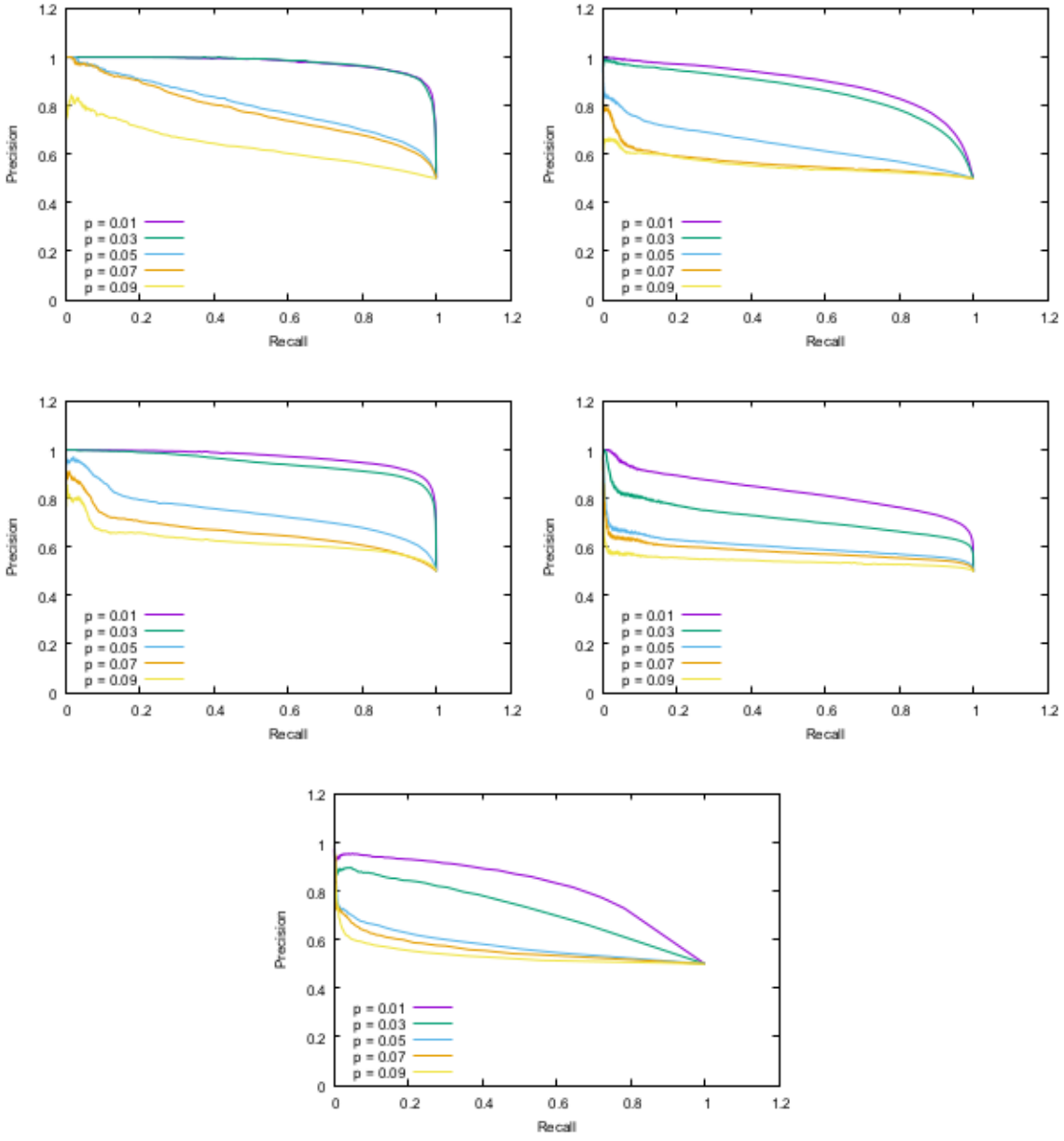
3) ACL, random model



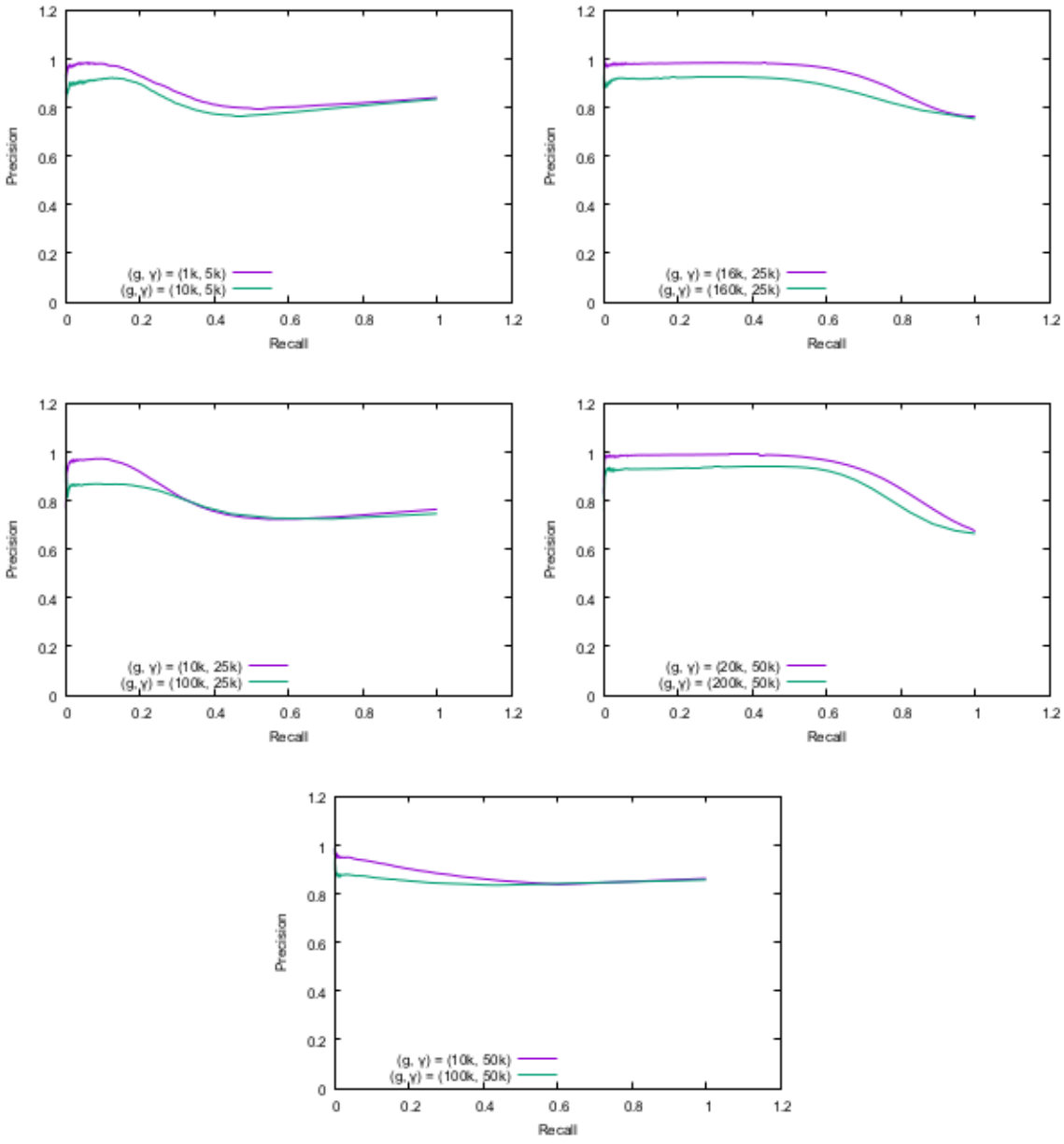
4) ACL, fixed model



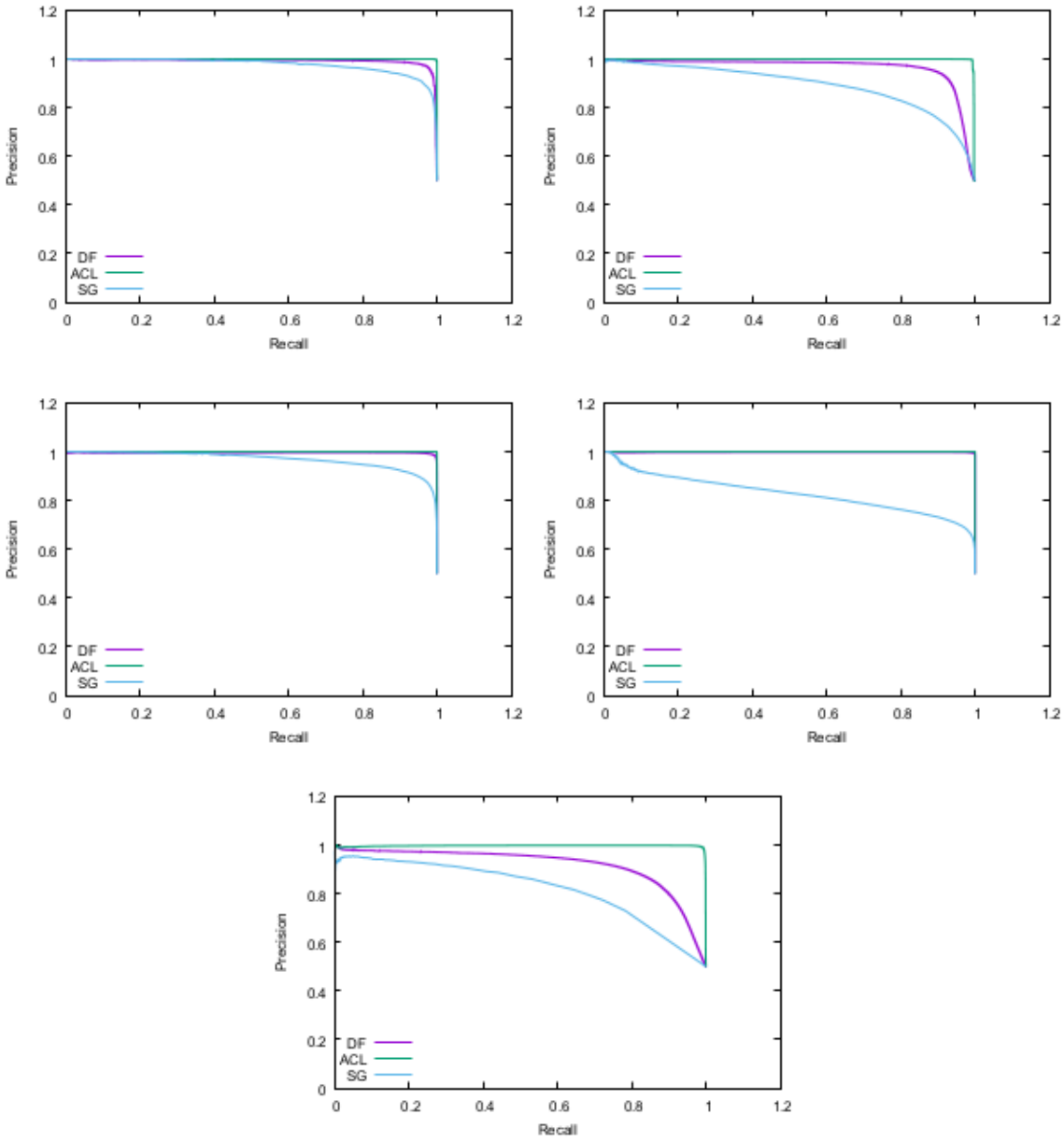
5) SybilGuard, random model



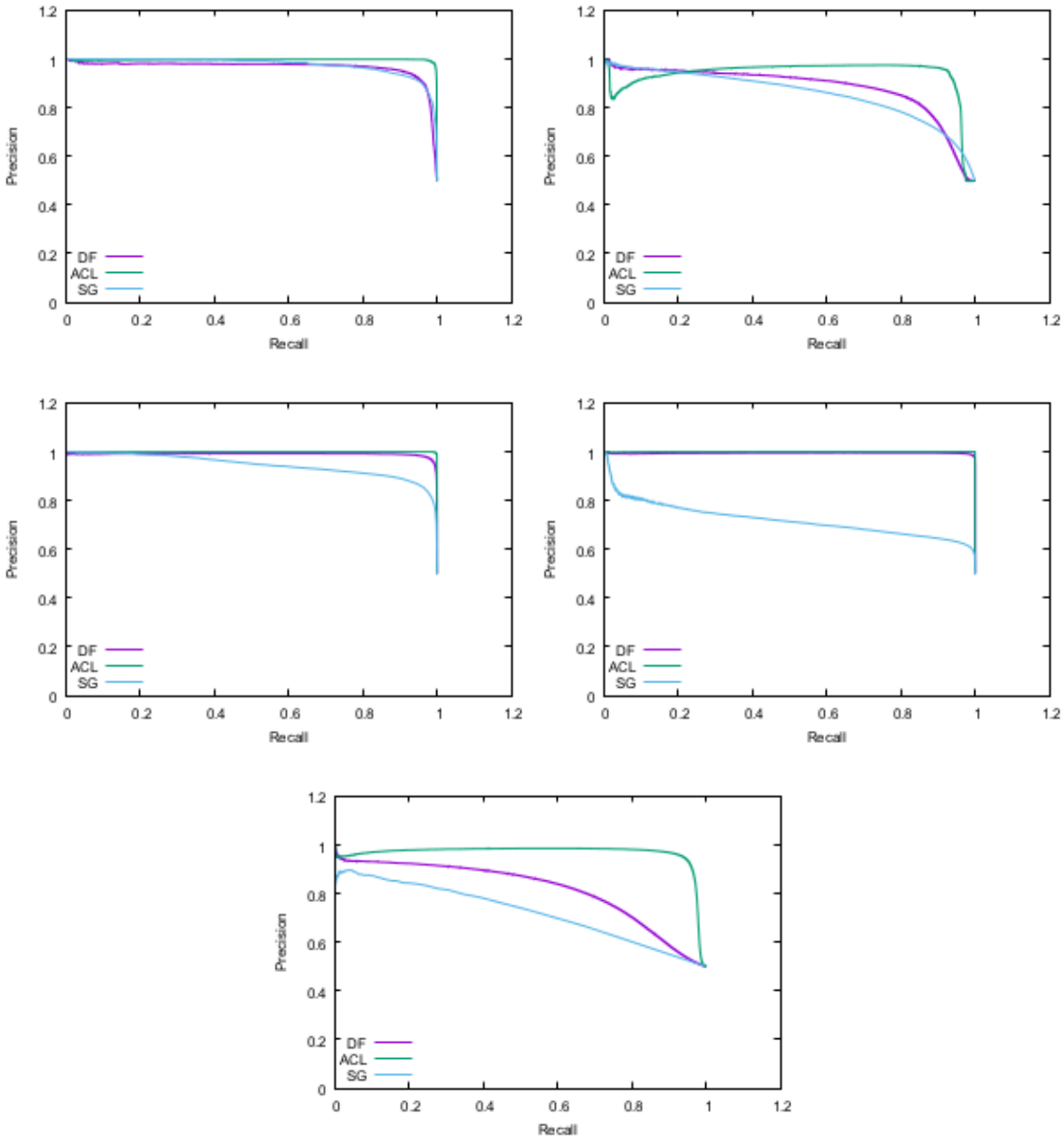
6) SybilGuard, fixed model



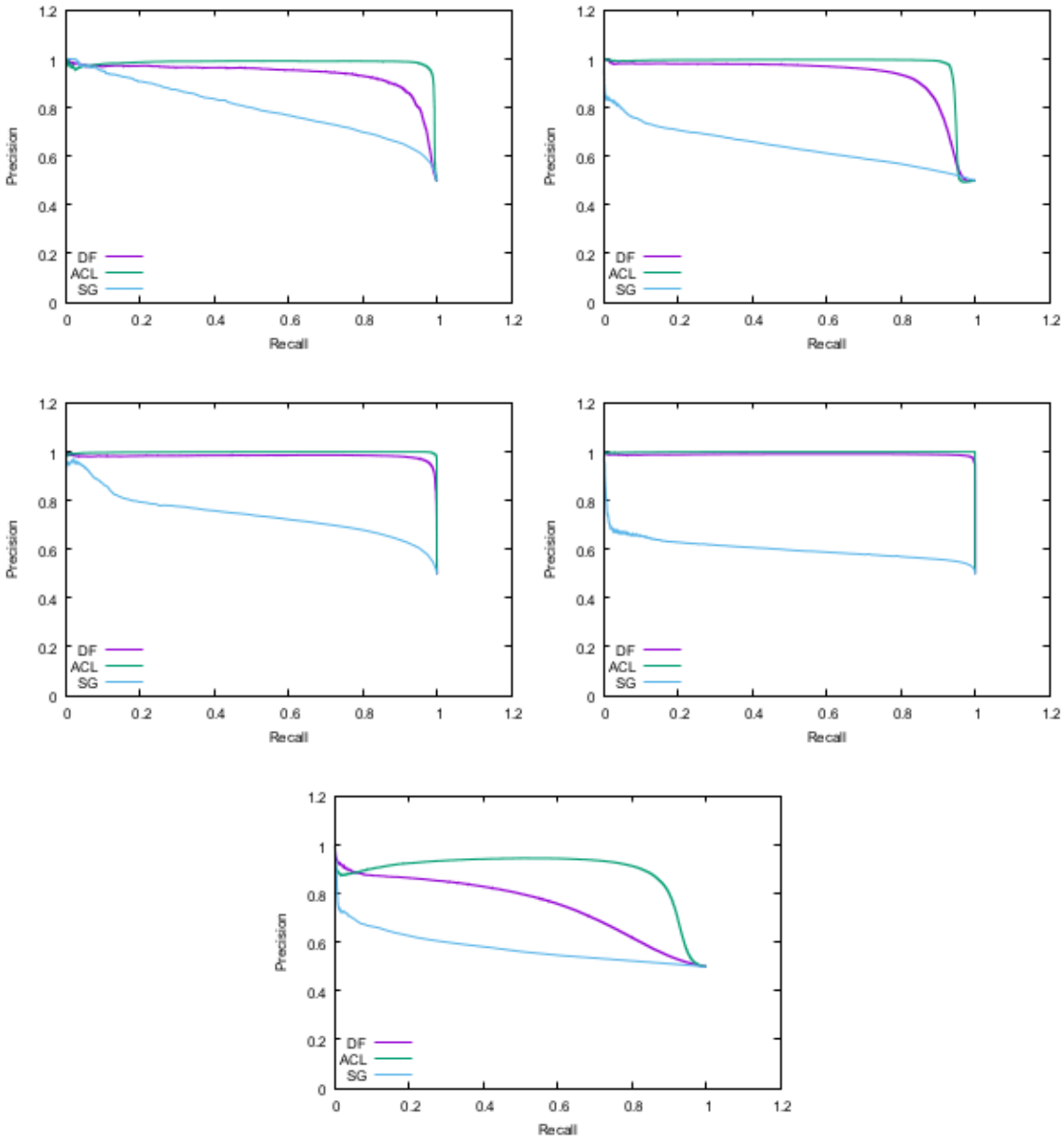
7) Random attack, p = 0.01



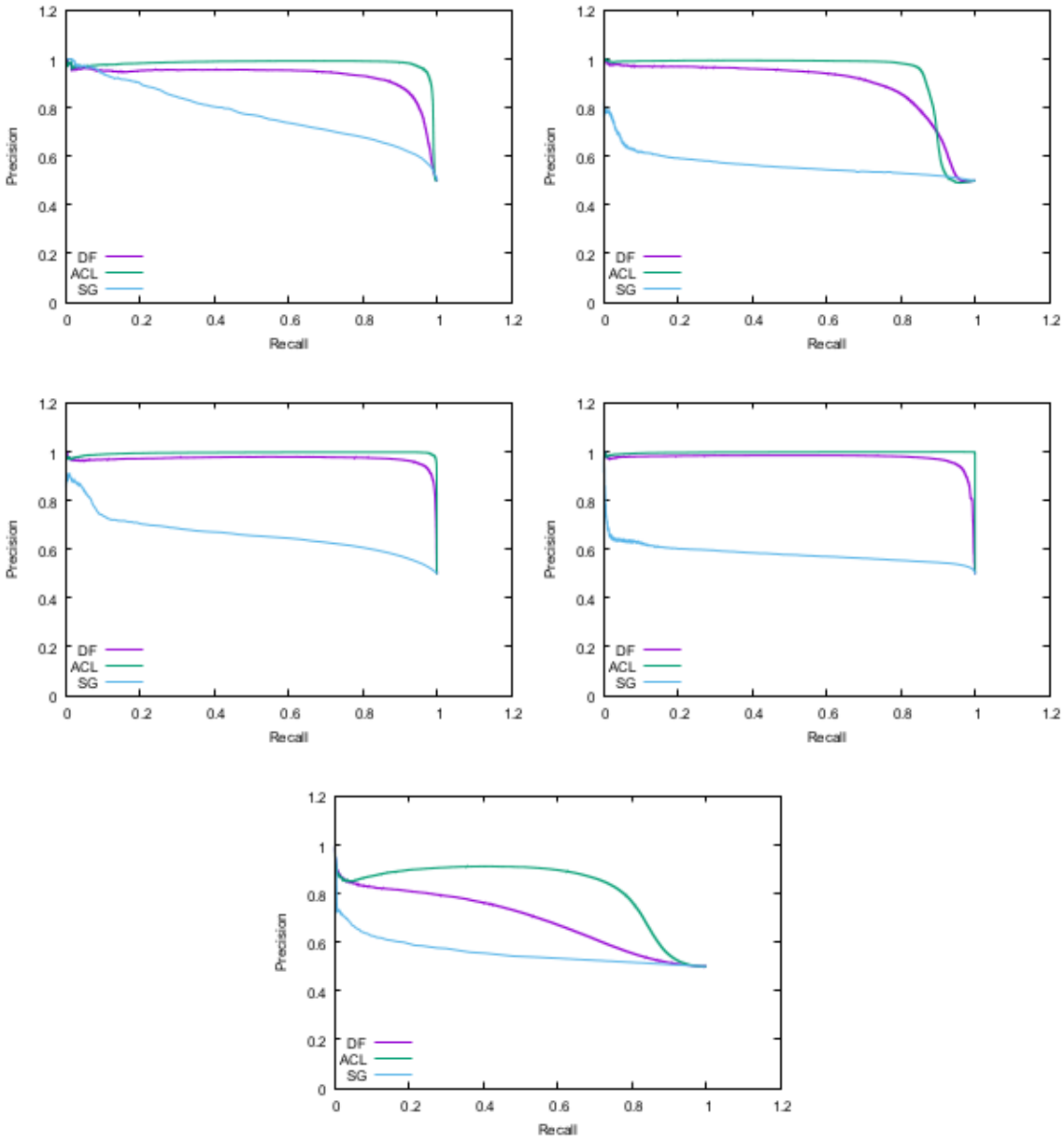
8) Random attack, p = 0.03



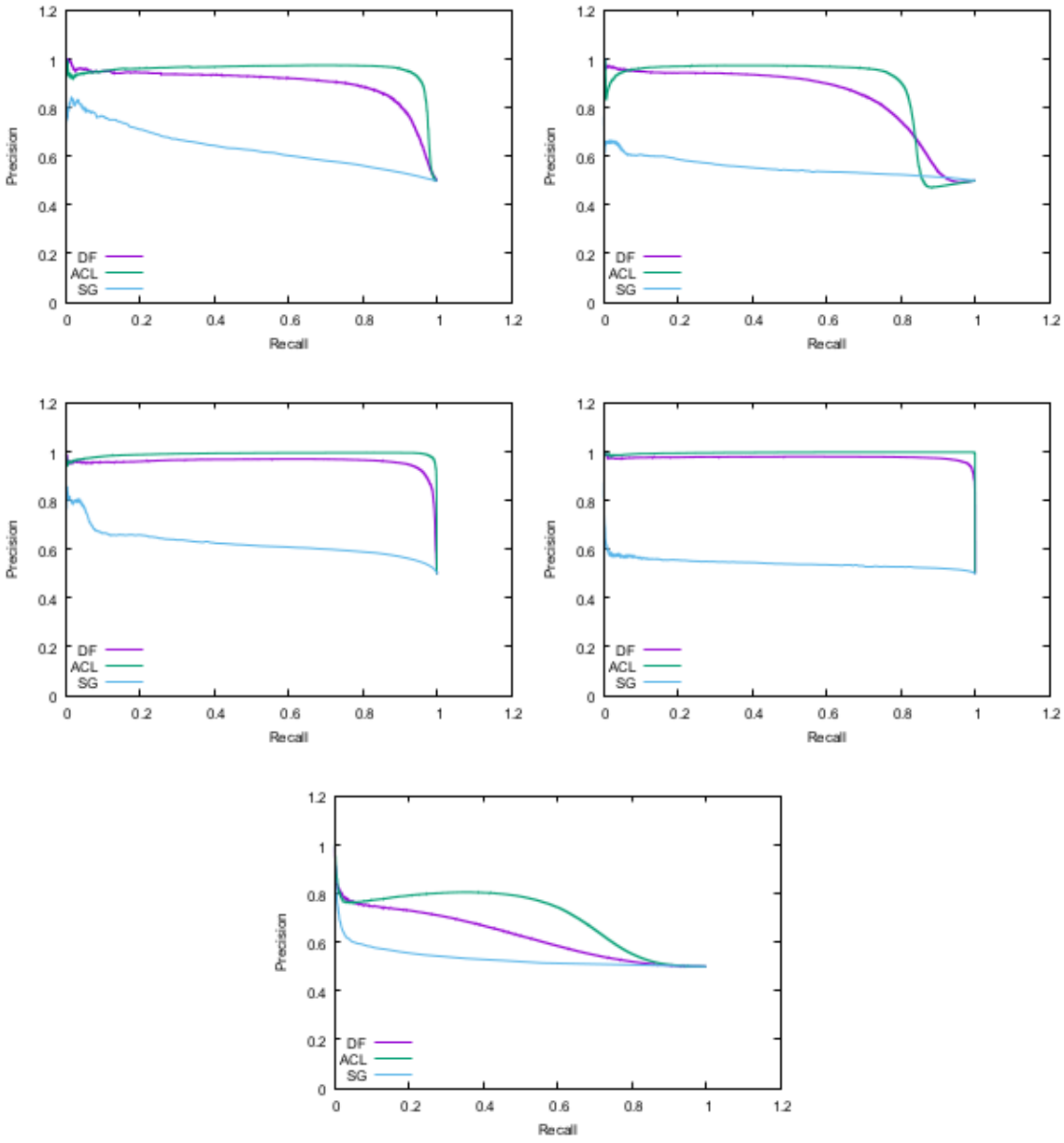
9) Random attack, $p = 0.05$



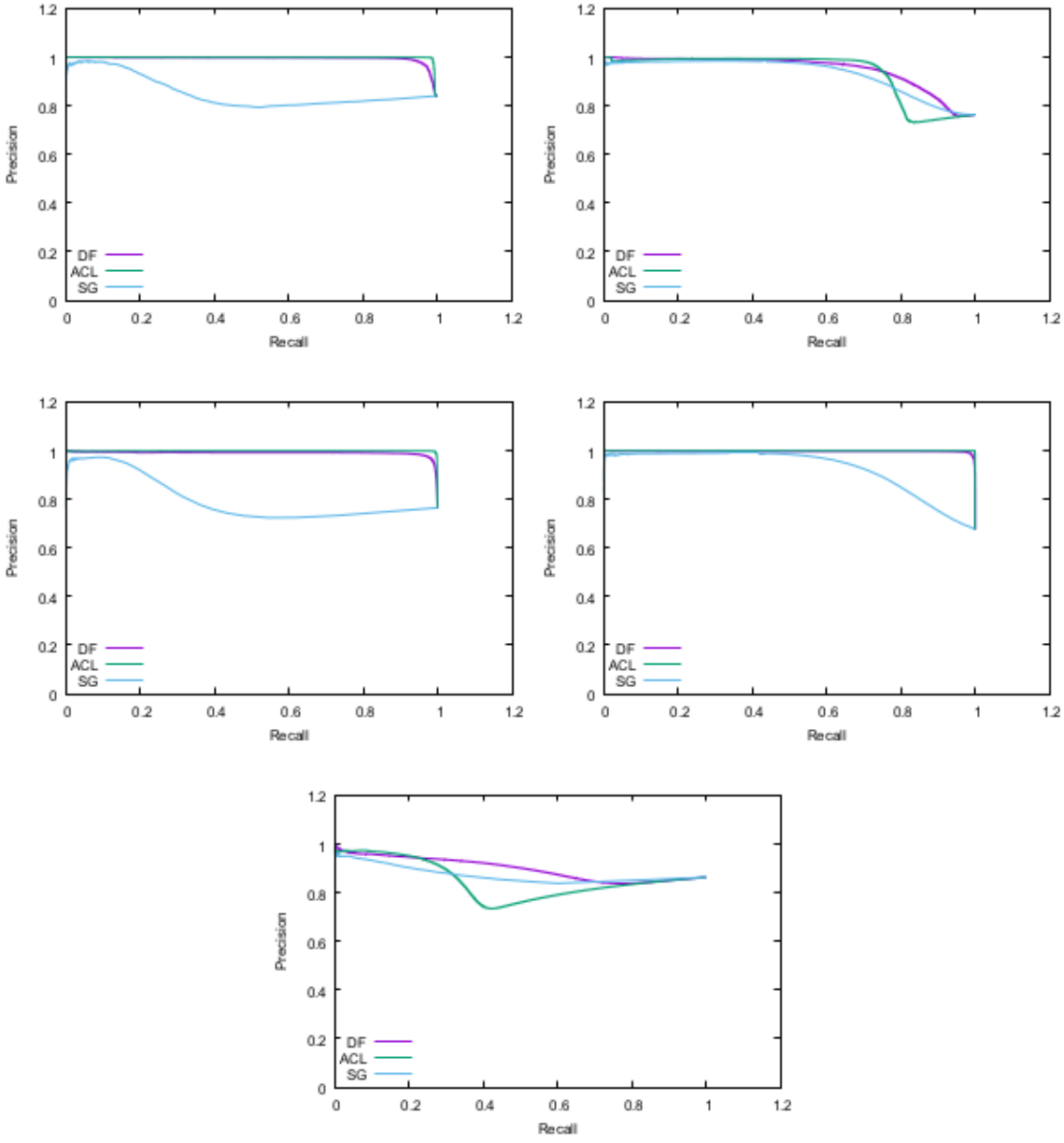
10) Random attack, $p = 0.07$



11) Random attack, $p = 0.09$



12) Fixed attack, weak



13) Fixed attack, strong