# Canal: Scaling Social Network-Based Sybil Tolerance Schemes

Bimal Viswanath
MPI-SWS
bviswana@mpi-sws.org

Mainack Mondal
MPI-SWS
mainack@mpi-sws.org

Krishna P. Gummadi
MPI-SWS
gummadi@mpi-sws.org

Alan Mislove
Northeastern University
amislove@ccs.neu.edu

Ansley Post
MPI-SWS
abpost@mpi-sws.org

## Abstract

There has been a flurry of research on leveraging social networks to defend against multiple identity, or Sybil, attacks. A series of recent works does not try to explicitly identify Sybil identities and, instead, bounds the impact that Sybil identities can have. We call these approaches *Sybil tolerance*; they have shown to be effective in applications including reputation systems, spam protection, online auctions, and content rating systems. All of these approaches use a social network as a credit network, rendering multiple identities ineffective to an attacker without a commensurate increase in social links to honest users (which are assumed to be hard to obtain). Unfortunately, a hurdle to practical adoption is that Sybil tolerance relies on computationally expensive network analysis, thereby limiting widespread deployment.

To address this problem, we first demonstrate that despite their differences, all proposed Sybil tolerance systems work by conducting payments over credit networks. These payments require max flow computations on a social network graph, and lead to poor scalability. We then present *Canal*, a system that uses landmark routing-based techniques to efficiently approximate credit payments over large networks. Through an evaluation on real-world data, we show that Canal provides up to a three-order-of-magnitude speedup while maintaining safety and accuracy, even when applied to social networks with millions of nodes and hundreds of millions of edges. Finally, we demonstrate that Canal can be easily plugged into existing Sybil tolerance schemes, enabling them to be deployed in an online fashion in real-world systems.


*EuroSys'12,* April 10–13, 2012, Bern, Switzerland.


***Categories and Subject Descriptors*** C.4 [*Performance of Systems*]: Design studies; C.2.0 [*Computer-Communication Networks*]: General—Security and protection

***General Terms*** Algorithms, Design, Performance, Security

***Keywords*** Sybil attacks; social networks; sybil tolerance; social network-based Sybil defense; credit networks

## 1. Introduction

Multiple identity attacks—commonly known as Sybil attacks [10]—are known to be a fundamental problem in many distributed systems. In a Sybil attack, a malicious user creates multiple identities and takes advantage of these identities to attack the system. For example, in social networking sites like Digg or YouTube, where content is rated based on user feedback, an attacker can create multiple identities and cast multiple votes, thereby manipulating content popularity. Recent studies have shown that Sybil attacks are becoming more widespread [42], affecting news aggregators like Digg [35], microblogs like Twitter [46], and review sites like TripAdvisor [33].

Recently, a series of schemes have been proposed that defend against Sybil attacks by leveraging social networks [24, 29, 34, 35]. These schemes are based on the assumption that, although an attacker can create an arbitrary number of Sybil identities, she cannot establish an arbitrarily large number of social connections to non-Sybil identities. In contrast to previous social network-based Sybil detection, e.g. [6, 44, 45], the schemes we consider do not explicitly identify Sybil identities in the network but, instead, bound the impact that Sybil identities can have. This approach is called *Sybil tolerance* [39]; these schemes have been shown to be effective in applications including reputation systems [9], spam protection [24], online auctions [29], and content rating systems [35].

We demonstrate that despite their differences, all of these schemes work by assigning credit values to links in a social network, and then conducting payments between nodes in the network. Effectively, these schemes are using *credit networks* [5, 12] as a basis for Sybil tolerance.[1] Unfortunately, these schemes do not scale well to large social networks. Finding routes for credit payments can be reduced to determining the maximum flow [11] between nodes in the network; doing this over large graphs is known to be expensive [13], and existing techniques for pre-calculating [14] the maximum flow are not directly applicable since the credit network is constantly changing. This serves as a practical deployment barrier, and to the best of our knowledge, none of these Sybil tolerance schemes have been deployed in a real-world system.

In this paper, we address this situation and scale Sybil tolerance schemes to extremely large graphs. We build Canal, a system that can efficiently approximate credit payments over large, dynamic networks. Canal trades accuracy for speed; we demonstrate that Canal's approximation rarely impacts users and does not change the Sybil tolerance properties of the application or benefit malicious users. We show that Canal can be directly plugged into existing Sybil tolerance schemes, and would reduce the credit payment latency from multiple seconds to a few hundred microseconds.

In more detail, Canal uses a novel landmark routing-based algorithm, routing credit payments via landmark nodes [36]. Canal consists of two components: a set of *universe creator* processes, which continually select new landmarks, and a set of *path stitcher* processes, which continually process incoming credit payment requests. Since the credit network is constantly changing (due to credit movements, as well as new identities and social links), Canal continually calculates new landmarks in parallel with making flow calculations. We design Canal to naturally take advantage of multiple cores and machines, enabling Canal to run over social networks that cannot be stored on a single machine.

We evaluate Canal on real-world networks at scale. We first demonstrate that Canal's approximation provides a dramatic speedup in the processing of credit payments, enabling Canal to be run in an online fashion. We then show that the approximation that Canal uses rarely impacts users, and that users eventually receive the same total available credit in Canal as they would in an exact system. Finally, we rerun the experimental setup of two previously proposed Sybil tolerance schemes, and demonstrate that using Canal would provide up to a 2,329-fold speedup in runtime while maintaining over 94% accuracy. This shows that existing schemes can naturally leverage Canal, and that Canal enables new schemes to be designed to inherit the benefits of credit networks.

The remainder of this paper is organized as follows. Section 2 provides background on the Sybil tolerance schemes we consider, and Section 3 provides background on the credit networks that underlie these schemes. Section 4 describes the design of Canal. Section 5 provides Canal microbenchmarks on real-world graph data, and Section 6 evaluates the performance of Canal when applied to real-world Sybil tolerance schemes. Section 7 concludes.

[1] Credit networks are a concept borrowed from the electronic commerce community. Credit networks provide a way to model trust between identities in a distributed system and leverage it as a payment infrastructure for transactions between arbitrary identities, even in the absence of a central trusted bank and common currency.

## 2. Background and related work

In this section, we give a brief overview of the prior work on social network-based Sybil defenses, with the goal of placing the contributions of this paper into context. A more extensive background is provided in [39]; we review the details relevant to Canal here. For the remainder of this paper, we consider *identity-based* systems where each user is intended to have a single identity and is expected to use the identity when interacting with other users in the system. In such systems, we call a user with multiple identities a *Sybil user* and each identity she uses a *Sybil identity*.

We divide the related work on Sybil defense into three classes, discussed in the sections below: Sybil prevention, Sybil detection, and Sybil tolerance.

### 2.1 Sybil prevention

Traditional defenses against Sybil attacks rely on either trusting central authorities or tying identities to resources that are hard to forge or obtain in abundance, preventing a user from creating many Sybil identities in the first place. We term these approaches *Sybil prevention*. For instance, systems like Cyworld [4] require users to present verified identities, such as passports or social security numbers, when creating new accounts. Other approaches include solving memory or CPU-intensive crypto-puzzles before granting access to system services [1–3, 41].

### 2.2 Sybil detection

Researchers have also explored allowing Sybil identities to be created but later detecting the identities and preventing them from interacting with other users (e.g., banning those identities) [32]. We term these approaches *Sybil detection*.

Recently, researchers have explored analyzing the structure of the social network as a mechanism for Sybil detection [6, 20, 30, 34, 44, 45]. To identify Sybils, all social network-based Sybil detection schemes make two common assumptions [43]:

1. Although an attacker can create an arbitrary number of Sybil identities in the social network, she cannot establish an arbitrary number of social connections to non-Sybil identities.

2. The non-Sybil region of the network is densely connected (or fast-mixing [25]), meaning random walks in the non-Sybil region quickly reach a stationary distribution.

The first assumption concerns how the Sybil and non-Sybil identities are connected and is necessary in order for the schemes to be able to leverage the social network; if it were not assumed, the attacker could establish social network links at will. While this assumption may not hold on all online social networks, recent work suggests that there are social networks where this assumption holds true [28]. The second assumption concerns the internal structure of the non-Sybil region and is necessary for these schemes to locate the boundary between the non-Sybil region and the Sybil region. If the second assumption did not to hold (implying small cuts existed within the non-Sybil region), the honest identities on either sides of cuts are likely be blocked from interacting with each other [40].[2]

### 2.3 Sybil tolerance

More recently, a series of schemes has taken an alternate approach to defend against Sybils. Instead of trying to explicitly label identities as Sybil or non-Sybil, these schemes are designed to limit the impact that a malicious user can have on others, regardless of the number of identities the malicious user possesses. We refer to these schemes as *Sybil tolerance* [39]. Sybil tolerance schemes make the same assumption 1 from Section 2.2, but avoid making assumption 2. Instead, they require more information about the system to which they are applied: In addition to the social network, these schemes also take as input the interactions between users. By doing so, they are able to allow or deny individual interactions, and reason about the impact (in terms of interactions) that identities have on one another.

The result is that the guarantees of Sybil tolerance are expressed in terms of interactions that are allowed. To compare, Sybil detection schemes reason only about identities (i.e., they reason about identities being *admitted*, and express their guarantees in terms of the number of Sybil identities admitted), while Sybil tolerance schemes reason about interactions (i.e., they decide whether certain interactions are allowed or denied). Thus, in a Sybil tolerance scheme, a certain pair of identities may be allowed to participate in certain interactions and not others, and may be allowed to interact at certain times and not others (all depending on the state of the system).

We now provide a brief overview of three example Sybil tolerance schemes. It is important to note that other Sybil tolerance schemes exist [9, 27], but we only discuss the three below for brevity.

***Ostra*** [24] is targeted at countering unwanted communication (i.e., spam). Ostra assumes the existence of a social network, and assigns credit values to the links. When a user wishes to send a message to another user, Ostra locates a path with available credit from the source to the destination. If such a path is found, credit is "paid" from each user to the next along the path, and the credit is refunded if the message is not marked as spam. If no path can be found, the message is blocked from being sent.

***SumUp*** [35] is designed to prevent users with multiple identities from manipulating object ratings in content sharing systems like Digg. SumUp assumes the existence of a social network and selects a trusted vote collector. SumUp then assigns weights to the links around the vote collector by handing out "tokens" (causing the links around the vote collector to be more highly weighted; links farther away are assigned weight 1). Finally, to vote, each voting identity must find a path with credit between himself and the vote collector and consume a credit along that path; if no such path can be found, the vote is discarded.

***Bazaar*** [29] provides stronger user reputations in online marketplaces like eBay. To do so, Bazaar creates a transaction network (akin to a social network) by linking pairs of identities that have successfully completed a transaction; the weight of each link is the dollar value of the transaction. When a new transaction is about to take place, Bazaar compares the value of the new transaction to the max flow between the buyer and seller. If sufficient flow is found, credit totaling the value of the transaction is removed between the buyer and seller, and is added back if the transaction is later reported to not be fraudulent. Otherwise, if sufficient flow is not found, the new transaction is denied.

Unfortunately, these schemes all tend to require significant computational resources in order to locate paths for credit payments. For example, on average, Bazaar takes over 6 seconds of CPU time to determine whether sufficient flow exists on a network with 5.5 M links [29], and Ostra requires over 35 milliseconds on a network with 3.4 M links [24]. Given that both of these are intended to be run in an online fashion, Bazaar would require an average of 6 seconds of CPU time for every bid on a marketplace like eBay, and Ostra would require an average of 35 milliseconds of additional CPU time for every message sent. Both of these represent substantial computational resource investments. With ever-larger and denser networks being created, it is unsurprising that—to the best of our knowledge—none of these schemes have been deployed in a real-world system.

The underlying reason for this high computation time is that these systems are required to check for credits on all possible paths, requiring one or more breadth-first-searches (BFSs) over the graph, with $O(E)$ cost per BFS.

[2] Researchers have begun to explore whether the second assumption holds in practice. Unfortunately, recent work [26] demonstrates that the mixing time for many real-world networks is substantially higher than was previously thought, suggesting that the networks are actually *not* fast mixing. Additionally, another study [19] demonstrates that in real-world social networks, identities in the periphery are often organized into densely connected clusters that connect to the rest of the network via a small cut.

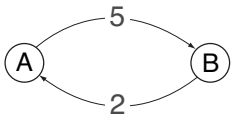


**Figure 1.** Simple credit network between two nodes $A$ and $B$, with credit available $c_{ab}$ and $c_{ba}$ shown. In this example, $A$ has 5 credits available from $B$, and $B$ has 2 credits available from $A$.

## 3. Sybil tolerance and credit networks

We now show that Sybil tolerance schemes are all effectively using *credit networks*. We first give a brief overview of credit networks before describing how existing systems are leveraging them internally.

### 3.1 Credit networks

Credit networks [5, 12] were first introduced in the electronic commerce community in order to build transitive trust protocols in an environment where there is only pairwise trust between accounts and there are no central trusted entities. In a credit network, identities (nodes) trust each other by offering pairwise credit (links) up to a certain limit. Nodes can use the credit to pay for services they receive from each other. The credit network could also be used as a payment infrastructure between nodes that do not directly extend credit to each other. Nodes can route credit payments to a remote node via network paths that traverse over links where credits are available (see Figures 1 and 2).

Formally, a *credit network* is a directed graph $G = (V, E)$ where $V$ is the set of nodes and $E$ is the set of labeled edges. Each directed edge $(a, b) \in E$ is labeled with a dynamic scalar value $c_{ab}$, called the *available credit*, and is initialized to $C_{ab}$. Intuitively, $C_{ab}$ represents the initial credit allocation that $b$ gives to $a$, and $c_{ab}$ represents the amount of unconsumed credit that $b$ has extended to $a$. Note that $c_{ab} \geq 0$ at all times.

Payments between two nodes in a credit network are contingent upon the availability of credit along network paths connecting the nodes. If a node $a$ wishes to pay node $b$ a total of $c$ credits, then a path

$$a \rightarrow u_1 \rightarrow ... \rightarrow u_n \rightarrow b$$

(which could just be $a \rightarrow b$) must exist where $c$ credits are available on each $(i, j)$ link (i.e., $c_{ij} \geq c$). If so, the credit available on each directed edge $c_{ij}$ on the path from $a$ to $b$ is decreased by $c$. As a result of this action, each node "pays" $c$ credits to its successor on the path to $b$.

It is not necessary to find a single path with available credit along each edge; instead, the payment could be split across multiple paths. For example, consider the network shown in Figure 3. In this scenario, node $A$ could pay 4 credits to node $E$ by paying 2 credits along $A \rightarrow B \rightarrow C \rightarrow E$ and 2 credits along $A \rightarrow B \rightarrow D \rightarrow E$.

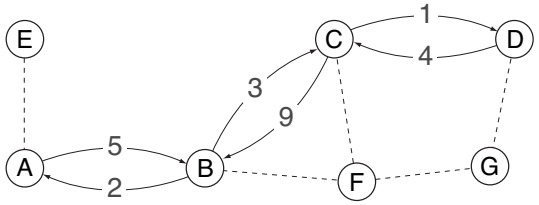


**Figure 2.** More complex credit network, with credit available ($c_{ij}$) shown for each link. In this example, $A$ can pay 1 credit to $D$ along the path $A \rightarrow B \rightarrow C \rightarrow D$. After paying the credit, the values on these links would be 4, 2, and 0, respectively. Note that, for simplicity, the links not on this path are only shown as dashed lines.

### 3.2 Sybil tolerant nature of available credit

Credit networks have been shown to be naturally tolerant to Sybil attacks [31]. In brief, we assume that an attacker is allowed to create as many identities as she wishes and manipulate the available credit on links between identities she owns. However, the attacker is able to establish only a limited number of links to non-malicious users (by assumption 1 in Section 2.2), and she can not manipulate the credit available to him on these links.

As shown in Figure 4, the total amount of available credit to the malicious user is the sum of the available credit on her links to other users. An attacker with an arbitrary number of Sybil identities has exactly the same available credit as the attacker with just one identity; in this case, the relevant set of edges is the cut between the subgraph consisting of the attacker's Sybil identities and the rest of the network. Any available credit on edges between the attacker's Sybil identities does not matter, because it does not enable ad-

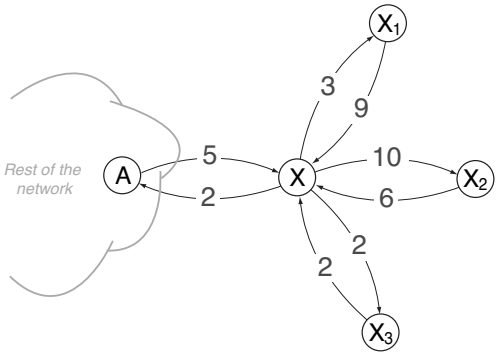


**Figure 4.** Credit networks leading to Sybil tolerance. User $X$ can create any number of identities ($X_1$, $X_2$, $X_3$) and arbitrarily assign the credit available between them. However, if $X$ wishes to pay credits from any of these identities to another identity in the rest of the network, the credits must be debited from $X$'s single valid link to $A$. Thus, the multiple identities do not enable any additional available credit with nodes in the rest of the network.


**Figure 3.** The (a) initial and (b) final state of the credit network with a credit payment along multiple paths. $A$ pays 4 credits to $E$: 2 credits are paid along the path $A \to B \to C \to E$ and 2 credits are paid along the path $A \to B \to D \to E$.

ditional payments to legitimate nodes. Moreover, collusion between malicious users does not enable the users to access more available credit together than they could separately.

Thus, available credit in a credit network is resilient to Sybil attacks.

### 3.3 Credit networks in existing systems

We now briefly discuss how the three example schemes discussed in Section 2.3 all work by essentially performing payments over credit networks. First, we note that each of the schemes internally uses a credit network: In Ostra and SumUp, the credit network is based on an externally provided social network, and in Bazaar, the credit network (called the *risk network*) is constructed from the feedback on prior transactions. Second, we observe that each scheme assigns available credit to links: In Ostra, the initial available credit is statically defined by the system operator, in SumUp, the credit is assigned by a token distribution mechanism, and in Bazaar, the credit is increased after each successful transaction.

Third, we observe that these schemes work by *paying credits* along paths between identities. In Ostra, a sender must first pay a receiver one credit before sending a message; if the sender is out of credit, the message is not delivered.[3] Similarly, in SumUp, each voter must pay one credit to the vote collector; if no path exists between the voter and vote collector with available credit, the vote is not counted. Finally, in Bazaar, when a transaction is about to occur, the system insists that the buyer pay the seller a number of credits corresponding to the new transaction value; if the sufficient available credit does not exist, the transaction is blocked.[4]

### 3.4 Computation speed of credit payments

The high computation time that Sybil tolerance schemes experience is explained by the use of credit payments over social networks. First, we observe that performing a credit payment involves searching for one or more paths with available credit between two nodes; this is essentially the maximum flow problem [11], which is known to be a computationally expensive operation. The most efficient algorithms for the maximum flow problem run in $O(V^3)$ [13] or $O(V^2 \log(E))$ [8] time. Second, techniques that precalculate the all-pairs maximum flow (e.g., Gomory-Hu trees [14]) cannot be applied to Sybil tolerance schemes, as these techniques assume a static network and impose a large, upfront pre-calculation cost of $|V| - 1$ maximum flow computations. Credit networks are constantly changing due to credit manipulations as well as new users and links; performing $|V| - 1$ maximum flow computations for every change in the credit network is impractical. Additionally, algorithms [17] that dynamically maintain a Gomory-Hu tree when the edge capacities increase or decrease often end up being expensive as well, requiring several maximum flow computations for each edge capacity update.

## 4. Canal design

We now detail the design of Canal, first giving a high-level overview of the model and goals of Canal before detailing the internal design.

### 4.1 Model and goals

Canal is designed to run alongside an existing Sybil tolerance scheme, providing two services: (a) maintaining the state of the credit network and (b) conducting credit payments. Canal is built to provide these services in a much faster manner than current implementations. We assume that Canal is run by the same organization that runs the Sybil tolerance scheme (alleviating concerns about Canal having access to potentially private social network data).

In order for existing Sybil tolerance systems like Bazaar and SumUp to take advantage of Canal, they need only allow Canal to store the credit network and replace any internal logic for conducting payments with calls into Canal. To avoid having to rebuild existing systems from scratch, Canal exports an API which can be easily used by existing Sybil tolerant applications. The API includes methods to initialize and add links to the credit network, but we are primarily concerned with the method

```
boolean payment(a,b,c)
```

[3] In Ostra, when a sender pays a credit to a receiver, a credit is debited from the sender–receiver path and, at the same time, *added* to the reverse path. Doing so allows Ostra to ensure liveness (as there is always credit available in the system).

[4] In Bazaar, this credit payment is undone if the transaction turns out not to be fraudulent.

that attempts a payment of $c$ credits from identity $a$ to identity $b$, and returns whether or not the payment could be made.

Canal responds to payment requests using an approximation that only considers a subset of the paths that exist when handling payments. As a result, Canal may not be able to find paths with available credit between a pair of users even though such paths exist. However, Canal will never find paths that do not exist or paths that do exist but do not have any available credit. Thus, Canal can suffer from *false negatives* (i.e., a payment is denied even though paths with available credit exist) but does not suffer from *false positives* (i.e., a payment is allowed even though sufficient credit is not actually available).

### 4.2 Design challenges

In order for Canal to be used in a real-world application, it has to overcome several challenges:

- *Latency:* Sybil tolerance schemes make user-visible decisions based on whether credit payments can be made. Thus, they will be practically deployable only if the Canal processing time is very fast (preferably in the order of a few milliseconds).
- *Efficiency:* Sybil tolerance schemes often have to process large numbers of payments in a short period of time; Canal must be deployable with reasonable computational resources.
- *Scalability:* Sybil tolerance schemes are designed to be run on very large social networks. Canal should support credit networks with hundreds of millions of links or more.
- *Accuracy:* Canal trades off accuracy for speed. The error introduced should not impact the Sybil tolerance guarantees, and should rarely impact users.
- *Dynamicity:* The credit network is constantly changing, due to payments being processed and changes to the social network. Canal should be able to support such a rapidly changing credit network.

### 4.3 Using landmark routing

Canal speeds up payments using a landmark routing-based technique. Historically, landmark routing [36] has enabled paths to be found between any pair of nodes via certain specific nodes (called landmarks). To do so, each node determines its path to the landmark; to route between a pair of non-landmark nodes, we need only have each route to the landmark. This is effectively *stitching* a path together out of two paths, and the stitched path may be longer than the shortest path (this is particularly likely when the landmark is located far away from the two nodes).

Canal's selection of landmark nodes is driven by three concerns, discussed in detail in the subsections below: First, we wish to be able to find short paths between nodes, but we are not required to use the absolute shortest path (Sybil tolerance systems are designed so that *any* payment path will do, but shorter paths often result in greater efficiency). Second, landmark routing is not typically designed for dynamic graphs (the paths to the landmark are generally treated as static; if the credit network is changing, Canal needs to update the paths to the landmarks). Third, we may need to conduct payments that require multiple paths (the credit available on any single path may not be sufficient).

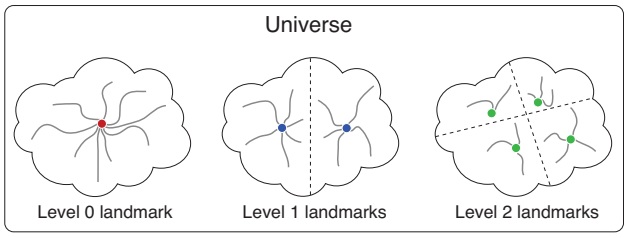


**Figure 5.** Diagram of a 2-level landmark universe, consisting of multiple levels of landmarks. Each level $i$ has $2^i$ landmarks. Paths can be found using landmark routing; all nodes share a level 0 landmark, and closer nodes share landmarks at multiple levels (resulting in shorter paths). The landmarks at lower levels induce partitions on the network (indicated by dashed lines).

#### 4.3.1 Finding short paths

Recent work has designed landmark routing techniques for accurately finding shortest paths in large networks [16]; we leverage this existing work to efficiently find short paths. Instead of using a single landmark, Canal uses a *landmark universe* with multiple *levels*. At each level $i$, there are $2^i$ landmarks selected randomly; each node finds a path to the closest landmark at each level. Thus, if there are $k$ levels, there are a total of $2^{k+1} - 1$ landmarks, and each node has paths to $k$ of these landmarks (one at each level). Furthermore, we refer to a universe with $k$ levels as a $k$-level landmark universe. A diagram of a landmark universe is shown in Figure 5.

Using a landmark universe enables Canal to find short paths. To see why, first consider a payment request between two nodes who are on opposite sides of the network (i.e., two nodes who have a relatively large shortest path length). For this pair of nodes, the only landmark they are likely to share is the level 0 landmark[5] (since they are far away, they are likely to have paths to two *different* level 1 landmarks, and two different level 2 landmarks, etc). Now consider the case of a payment request between two nodes who are close in the network. For this pair of nodes, there is likely to be a number of landmarks shared between them (since they are close in the network). By stitching a path between these nodes via one of the higher-level landmarks, we are likely to find a short path. A full explanation is available in [7, 16].

[5] Note that any pair of nodes is guaranteed to share at least one landmark in a given universe (the level 0 landmark), although the stitched path via that landmark is not guaranteed to have credit.

It is important to note that Canal's correctness guarantees are not affected if a Sybil node is selected as a landmark. On the other hand, Sybil landmarks may affect the liveness of paths stitched via that landmark, as links near the landmark may not have credit. As a result, a Sybil landmark may not be useful for routing credit payments. Since Canal typically uses hundreds of landmarks at any time, Sybil landmarks rarely impact Canal's ability to route credit payments.

In Section 5, we show how the deployer of Canal can select the level $k$ of each universe. Higher values of $k$ allow shorter paths to be located, but introduce an exponentially increasing number of landmarks (and corresponding overhead). In practice, setting $k$ to 5 works well on the social networks we use to evaluate Canal.

#### 4.3.2 Handling dynamic credit networks

We observe that, due to the rapidly changing nature of the credit network, any existing landmark data may quickly become stale. For example, as new links are introduced into the credit network, paths may exist that are not reflected in the landmark universe. Similarly, as credit payments are processed, paths near lower-level landmarks are likely to become congested and may run out of available credit (since a disproportionate number of credit payments will be routed via these landmarks); this would prevent Canal from finding available credit that may exist via other potential landmarks.

Canal addresses this issue by continually constructing landmark universes as it is running, replacing an old universe whenever a new one is created. This serves two purposes. First, continually constructing universes enables Canal to incorporate changes in the credit network into the landmark path data. Second, continually constructing universes ensures that any node is only a landmark for a short period of time, reducing the likelihood that the paths around the landmark would become congested with credit payments.

#### 4.3.3 Finding multiple paths

We noted above that Canal may need to conduct payments that require multiple paths (i.e., in the example in Figure 3, suppose $A$ wishes to pay 5 credits to $E$). In particular, we would like to be able to locate *disjoint* paths, as this increases our likelihood of finding the necessary available credit.

Canal addresses this issue by keeping a queue of recent landmark universes available. As new landmark universes are generated, they are added to the end of the queue and the oldest existing landmark universe is discarded. Keeping a set of universes available enables Canal to find paths between pairs of nodes via the landmarks that exist in different universes. By configuring the number of landmark universes that are stored in the queue, Canal can control the maximum number of paths that can be used at once for a credit payment.

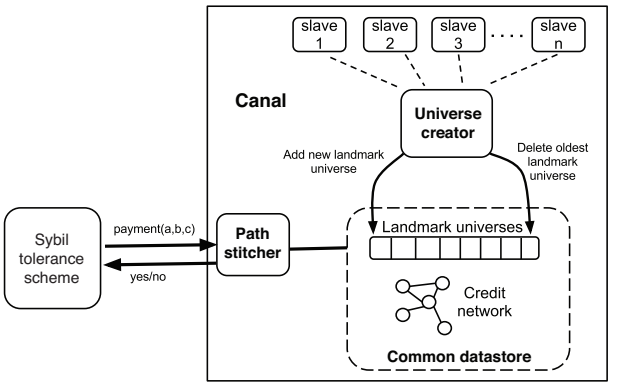


**Figure 6.** Canal system design.

### 4.4 Canal components

Figure 6 gives a high level view of Canal system design. There are three main components in Canal: a common datastore, *universe creator* processes, and *path stitcher* processes. The common datastore serves as a location to store the landmark universes and the credit network. The universe creator processes continually generate new landmark universes as described above, and the path stitcher processes respond to payment requests by the Sybil tolerance scheme. In the rest of this section, we will discuss in detail the design of each component.

#### 4.4.1 Common datastore

The common datastore serves as the repository for the state of the credit network as well as the landmark universes. The credit network is stored as a hash table of links, with each link stored with its current available credit. Because multiple processes will be manipulating the values on the credit network, each link also contains a shared/exclusive lock that processes must obtain before reading/writing the credit value. When a new link is added to the credit network, its lock and credit available are first initialized before it is inserted into the credit network hash table.

The landmark universes are stored using a linked list, with a global pointer to the head of the landmark universe list and each landmark universe pointing to the next. Since multiple processes will be scanning the landmark universe list, the pointers are also protected with read/write locks which processes obtain before following or changing a pointer.

The landmark universes themselves are represented as a series of *landmark maps* with one landmark map for each level in the landmark universe. Each landmark map contains tuples

```
(node, landmark, next_hop)
```

with one entry for each node in the network. The `landmark` represents the given node's landmark at this level, and the `next_hop` represents the node's next hop towards this landmark. By recursively following the `next_hop`s, each node can reconstruct its path to the landmark. Thus, in a $k$-level

landmark universe, there are $k$ landmark maps, each with an entry for all nodes in the network. Thus, each landmark universe requires $O(n \cdot k)$ space, where $n$ is the number of nodes in the credit network.

#### 4.4.2 Universe creator processes

We now describe the design of the universe creator processes, which construct new landmark universes. Assume that Canal is configured to construct $k$-level landmark universes. The universe creator processes all continually construct landmark universes using the following approach, taken from [16].

1. Randomly select $k$ random node sets of sizes $2^0, 2^1, 2^2, 2^3, \ldots 2^k$ respectively, from the network. Let the selected sets be denoted by $V_0, V_1, V_2, \ldots V_k$. These sets contain the new landmarks at each level.
2. For each set $V_i$, and every node $u \in V$, calculate the shortest path from $u$ to each of the landmark nodes in each set $V_i$. This is done by having the processes perform BFSs from each landmark in $V_i$.
3. Finally, using the BFSs, construct the landmark map for level $V_i$ by select the closest landmark node in $V_i$ and the next hop for all nodes.

In Canal, we speed up this universe creator process using three techniques: First, Canal exploits the fact that conducting the BFSs from the new landmarks can be parallelized. We configure the BFSs to be conducted in parallel by a set of slave threads. Second, to make sure that we only find paths with available credit, we design the BFS algorithm to only consider edges with available credit. This allows newly constructed landmark universes to "route around" links which have no available credit (and cannot be used for payments).

Third, the process of selecting the closest landmark at a given level for all nodes (step 3 above) can be broken down into a series of *merges* that can easily be parallelized as well.[6] Let us consider the $2^i$ BFSs that are conducted when constructing the landmark map for level $V_i$. Note that these BFSs are completed at different times by different processes. The landmark map is constructed by taking the first BFS and creating an entry for every node in the landmark map pointing to the landmark at the root of the BFS. Then, as subsequent BFSs complete, they are merged into the landmark map by scanning over the existing landmark map, and changing any tuples where the node is closer to the new landmark than to any landmark previously merged. In fact, multiple landmark maps can be merged together in the same manner, allowing all of the processes to contribute to constructing the landmark map.

Once the new landmark universe is constructed, it is added on to the end of landmark universe list in the common datastore. At the same time, the oldest landmark universe is removed from the front of the list and discarded. This ensures that landmarks are only "active" for a short period of time, reducing the likelihood that they will become hotspots in the network (Section 5 shows this happens rarely).

#### 4.4.3 Path stitcher processes

Finally, we describe the design of the path stitcher processes, which respond to `payment` requests from the Sybil tolerance scheme. Let us suppose that a path stitcher process has received a request to pay $c$ credits from node $a$ to $b$. At a high level, the path stitcher process walks down the landmark universe list, looking for paths with available credit between $a$ and $b$ using the landmarks in each universe. As soon as the path stitcher process has found paths with a total of $c$ available credits, it returns a successful result. Otherwise, if the path stitcher process reaches the end of the universe list without finding a total of $c$ credits, it returns an unsuccessful result.

To find paths with available credit in a single $k$-level landmark universe, the path stitcher process executes the following algorithm:

1. Scan the $k$ landmark maps and collect the set of common landmarks between $a$ and $b$. Note that there is guaranteed to be at least one common landmark (at level 0).
2. For each shared landmark, use the `next_hop` in the landmark map to "stitch" together a path via the landmark.
3. Refine the path by (a) eliminating any cycles and (b) performing path *short-cutting*. To perform short-cutting, we traverse the path up to the landmark node and see if there is a link from any of these nodes to a node lying in the path after the landmark node. If so, we short-circuit the path by using that link to create a shorter path between $a$ and $b$.

This process results in up to $L$ paths between $a$ and $b$, where $L$ is the number of common landmarks in this universe.[7]

Next, the path stitcher process pays as much credit as possible along each path. For each path, the path stitcher process walks the path, obtaining the lock on each link of the path, temporarily lowering the credit available to 0, and then releasing the lock. Once the end of the path has been reached, the path stitcher calculates the maximum credit available on the entire path (determined by the link with the minimal credit available); let this be $C$. Then, the path stitcher process walks the path once more, locking each link and resetting the credit available to be its previous value minus $C$. This effects a removal of $C$ credits along the entire path.

Once sufficient credit has been removed to meet the original `payment` request, the path stitcher process returns a suc-

[6] Our current implementation does not support the parallel BFS merge feature. We plan to incorporate this in the future.

[7] There could be potentially fewer resulting paths, as some of the paths may end up duplicated (e.g., if a path happens to cross two landmarks). This happens rarely in practice.

| Network | Nodes | Links | Avg. degree | Avg. max flow time (s) |
|---|---|---|---|---|
| Renren [18] | 33 K | 1.4 M | 21.1 | 0.352 |
| Facebook [38] | 63 K | 1.6 M | 25.7 | 0.445 |
| YouTube [23] | 1.1 M | 5.8 M | 5.2 | 2.91 |
| Flickr [22] | 1.6 M | 30 M | 18.8 | 15.2 |
| Orkut [23] | 3.1 M | 234 M | 76.3 | 220 |

**Table 1.** Statistics of the networks we evaluate Canal on. Also included is the average time for completing a max flow computation.

cessful result. However, if the end of the universe list is reached without enough credit being found, the path stitcher process first replaces any credit removed before returning an unsuccessful result.

The path stitching process is fast, since the landmark paths are all pre-computed; the path stitcher process simply walks the paths and removes available credit. Additionally, the path stitcher processes only ever hold a single link lock at a given time, ensuring that Canal is deadlock-free. Finally, the memory requirements of the path stitcher process is low, as it only needs to temporarily store the paths where it has removed credit.

### 4.5 Implementation

Our implementation of Canal is written in 2,269 lines of C++ (excluding publicly available libraries). The current implementation is designed to be run on a single machine, and uses Pthreads for the universe creator and path stitcher processes and Pthread locks to protect shared data. Our implementation is written so that the deployer can specify the number of universe creator and path stitcher processes to use; the tradeoff depends on the number of incoming `payment` requests that the deployer wishes to process (since more path stitcher processes allows a higher throughput, assuming CPU resources are available). We demonstrate in Section 5 that our single-machine implementation is able to support graphs with over 220 million edges. It is important to note while our current implementation runs on a single machine, we have designed Canal to be implementable across a cluster of machines. Doing so would allow Canal to be deployed on credit networks that are too large to fit in a single machine's memory. Canal can be implemented using graph parallel processing platforms [15, 21] that automatically distribute the graph state (our credit network) across multiple machines. This would mean that the Canal common datastore would be distributed across multiple machines and the universe creator processes would be using distributed graph processing algorithms. Existing graph parallel processing platforms follow the bulk synchronous parallel (BSP) model [37] and the challenge would be to minimize the global communication between the processes and the barrier synchronization cost associated with BSP algorithms. Additionally, a distributed implementation would likely require transactional updates to the common datastore to properly handle node failures. However, we leave a full implementation of a distributed version of Canal to future work.

| Renren | Facebook | YouTube | Flickr | Orkut |
|---|---|---|---|---|
| 225 | 292 | 4,131 | 13,296 | 41,787 |

**Table 2.** Time in milliseconds to calculate a level-0 landmark universe with one universe creator process for various datasets.

## 5. Canal microbenchmarks

In this section, we explore a number of Canal microbenchmarks before exploring how Canal performs alongside Sybil tolerance schemes in the following section. Here, we center our evaluation around five questions:

- What is the memory overhead of landmark universes?
- How expensive are landmark universes to compute?
- What is the response time for `payment` requests?
- Do nodes receive all of their available credits over time?
- Do "hotspots" in the network form around landmarks?

### 5.1 Experimental setup

We evaluate Canal on five real-world social networks of varying size, shown in Table 1. These networks are some of the largest publicly available social network data sets, and cover a wide number of nodes (33 K to 3.1 M) and edges (1.4 M to 234 M).

We run our experiments on machines with dual 12-core Intel Xeon X5650 2.66 GHz processors and 48 GB of RAM. In many of the experiments, we vary two key parameters of Canal: the universe level, and the size of the universe list (i.e., the number of universes that are cached). Unless otherwise stated, each experiment is the average of five random trials.

For reference, the final column in Table 1 shows the average time to compute max flow[8] between 50 random pairs of nodes in these networks; this further emphasizes the computational expense of conducting credit payments on large networks. Even for networks with only a few million links, the computation time can quickly become multiple seconds.

### 5.2 Memory and compute time of landmark universes

Canal has a certain fixed memory requirement to hold the credit network and edge locks in memory. For example, Orkut, the largest graph we consider, requires almost 20 GB of memory for storing the graph state. However, Canal also requires memory for storing the landmark universes; the amount depends on the universe level and size of the universe list. Figure 7 plots the memory size for a single

[8] We use the push-relabel algorithm [13] for computing max flow.

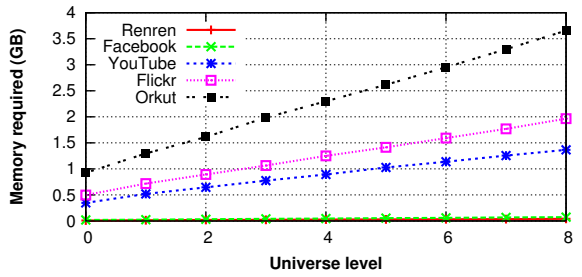


**Figure 7.** Memory requirements of different universe levels. The memory required increases linearly with the universe level.

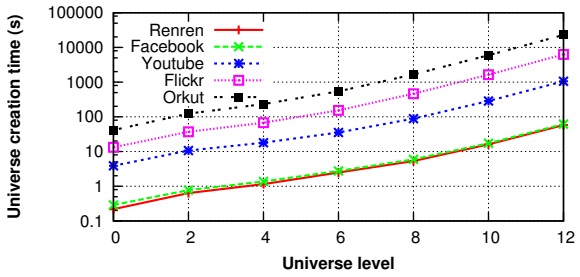


**Figure 9.** Graph showing the absolute landmark universe creation time as we increase the number of universe levels, for Canal configured with 22 universe creator processes.

landmark universe of different levels for each of the five networks we consider (multiple landmark universes simply require multiples of this size). We observe that the memory size increases linearly with the landmark universe level, and that the sizes allow multiple landmark universes to be kept in memory on our test machine.

We now turn to examine how quickly landmark universes can be created. Table 2 presents the time required to construct a level-0 landmark universe with a single universe creator process; this follows the general trends of the max flow results in Table 1, but is sometimes higher due to Canal's use of per-link locks.

However, in Canal, we can take advantage of multiple cores to conduct landmark universe creation in parallel. Figures 8 and 9 present the speedup (relative to creating a landmark universe with a single universe creator process) and absolute time, respectively, when creating different levels of universes with 22 universe creator processes. We observe that higher level universes enable a greater level of parallelism, meaning Canal is more efficient at creating higher level landmark universes.

### 5.3 Latency of `payment` requests

Next, we examine the latency of processing `payment` requests in Canal. For this experiment, we select 5,000 random pairs of nodes in each network, and issue a `payment` request between each pair of nodes for one credit. We then record the latency of the response from Canal. To see how the latency scales with the payment size, we then repeat this experiment by pushing five units of credit. We use 8 level-5 landmark universes, and the credit network is initialized to have one available credit per link.

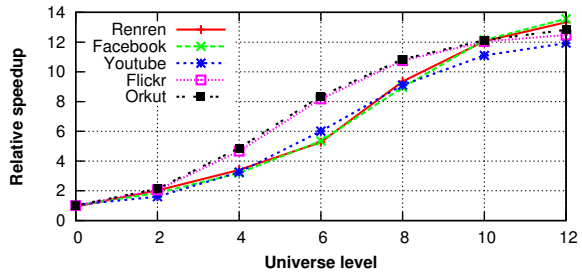


**Figure 8.** Landmark universe creation time speedup, relative to a single universe creator process, for Canal configured with 22 universe creator processes.

Table 3 presents the results for the five networks. We observe that both the median and 95th percentile latency for pushing one credit is below 1 millisecond for all networks, and the latency for pushing five credits is below 2.5 milliseconds for all networks. This represents a substantial speedup, as existing systems often take multiple seconds to determine if sufficient available credit is present in the credit network [24, 29].

### 5.4 Do nodes eventually receive all available credit?

Recall that, at any particular moment, Canal can only use a subset of the paths with available credit between two nodes (in particular, it can only use the paths via their common landmarks). However, if available credit along some of these paths is used up, Canal will disregard the exhausted links when constructing new landmark universes. Thus, even though a node only has access to a subset of its available credit at any one time, it will eventually receive more of its credit as new landmark universes are created. We now explore *how long* it takes for a node to access all of its available credit.

To do so, we pick 50 random pairs of nodes with degree greater than 10 (so that there is significant available credit between them) and the credit network is initialized to have

| | 1 credit | | 5 credits | |
|---|---|---|---|---|
| **Network** | **Median** | **95th P.** | **Median** | **95th P.** |
| Renren | 0.04 | 0.09 | 0.40 | 0.64 |
| Facebook | 0.04 | 0.13 | 0.52 | 0.74 |
| YouTube | 0.14 | 0.59 | 1.3 | 2.3 |
| Flickr | 0.17 | 0.51 | 1.3 | 2.0 |
| Orkut | 0.34 | 0.83 | 0.89 | 1.9 |

**Table 3.** Median and 95th percentile time in milliseconds taken by Canal to respond to a `payment` requests pushing a one unit and five units of credit.

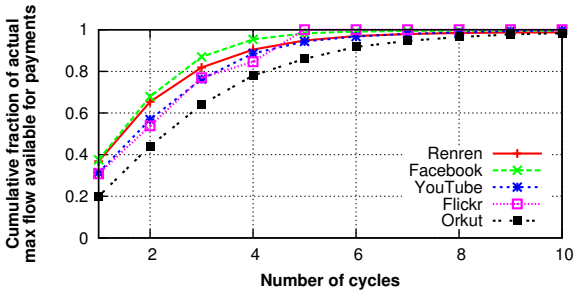


**Figure 10.** Cumulative fraction of actual max flow that is available for payments in Canal, for increasing cycles of landmark universe creation. We observe that nodes can quickly access all of their available credit.

one available credit per link. We then conduct a number of *cycles*, where each cycle consists of constructing a new set of 8 level-5 landmark universes and then making the largest payment possible between each pair of nodes (meaning we use up all of the available credit between the two nodes). Over time, we expect that the total payments will approach the actual max flow in the credit network between each pair of nodes.

Figure 10 presents the results of this experiment, showing the cumulative fraction of the actual max flow in the credit network that is used for payments. As expected, we see that the total payments asymptotically approach the actual max flow. More importantly, we observe that it does so very quickly: For example, node pairs in each dataset can achieve between 80% and 95% of their actual max flow in just 4 cycles. This indicates that even though Canal only has access to a subset of paths at any one time, nodes do eventually receive all of their available credit, even over short time windows.

### 5.5 Do landmarks lead to hotspots?

Our final microbenchmark concerns whether or not landmarks in Canal end up becoming "hotspots." To explore this effect, we again select 5,000 random pairs of nodes and have each pair of nodes pay one credit between each other. We

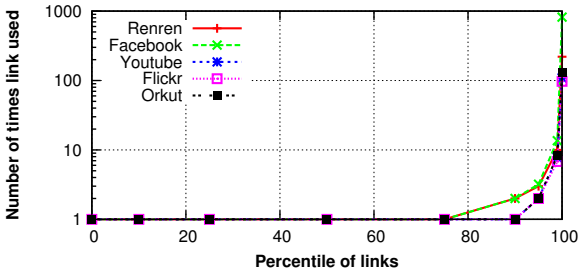


**Figure 11.** Distribution of the number of times links are used when processing 5,000 random credit payments. For all networks, the 99th percentile links are used fewer than 14 times.

| Category | Nodes | Links |
|---|---|---|
| Clothes | 1.3 M | 5.5 M |
| Home | 1.3 M | 4.5 M |
| Collectables | 419 K | 1.2 M |
| Electronics | 600 K | 1.5 M |
| Computing | 626 K | 1.7 M |

**Table 4.** Size statistics of the different categories of risk networks used in evaluating Canal implementation of Bazaar.

then count the total number of times each link in the network was used in transferring a credit. If hotspots form, we would expect to see a number of links used many times. For this experiment, we configure Canal to have 8 level-5 landmark universes.

The results of this experiment are shown in Figure 11. We plot the number of times a link is used versus percentile of links, and find that almost all links are used only once, and very few links are used many times. For example, the 90th percentile link is used no more than twice in all networks, and the 99th percentile link is used no more than 14 times.

## 6. Applying Canal to Sybil tolerance systems

The second half of our evaluation considers the impact that Canal would have on previously proposed Sybil tolerance schemes. In particular, we integrate Canal into both Bazaar [29] and Ostra [24]. We then recreate the original evaluation of these schemes and measure the speedup that these schemes observe when running with Canal, as well as the resulting impact on accuracy (in terms of false negatives).

### 6.1 Bazaar

Recall that Bazaar is designed to strengthen user reputations in online marketplaces like eBay. We replace the storage of the credit network (called the *risk network* in Bazaar) and max flow calculation components with Canal, and reperform the same evaluation as in the original paper. Bazaar was originally evaluated using a 90-day trace of five of the largest categories on the UK eBay site, and we use the same dataset to evaluate accuracy and speed up of our implementation compared to the original one. The five categories range in size from 419 K users to 1.3 M users, and from 1.2 M links to 5.5 M links as shown in Table 4.

Table 5 presents the latency for credit network `payment` transactions for the original implementation of Bazaar and for Bazaar augmented with Canal. We make a number of interesting observations. First, we observe that the median latency for transactions is below 200 microseconds for all categories, and the 95th percentile latency is below 4 milliseconds. This low latency enables Bazaar to be used in an online fashion. Second, when compared to the latency of the original Bazaar implementation, we observe speedups of be-

| Category | Orig. Avg. | Canal Med | Canal 95th P. | Relative Speedup |
|---|---|---|---|---|
| Clothes | 6,290 | 0.2 | 3.4 | 2,329 × |
| Home | 5,340 | 0.1 | 3.4 | 785 × |
| Collectables | 1,180 | 0.08 | 2.0 | 1,404 × |
| Electronics | 1,660 | 0.09 | 2.70 | 1,522 × |
| Computing | 1,410 | 0.1 | 2.56 | 1,084 × |

**Table 5.** Time in milliseconds required to process credit network transactions in Bazaar with Canal with 30 level-2 landmark universes. Also included is the original processing time from the Bazaar paper and the relative speedup. We observe speedups between 785-fold and 2,329-fold.

tween 785-fold and 2,329-fold. This underscores the impact of Canal on Sybil tolerance systems like Bazaar.

However, this reduction in latency comes at the cost of accuracy. Since Canal can only look for credit on a subset of paths, it may be unable to find sufficient available credit between a buyer–seller pair, thereby wrongly flagging a transaction as fraudulent. To determine how often this occurs, we calculate the fraction of the transactions for which the original Bazaar implementation found sufficient available credit, but Canal was unable to.

The results of this experiment are presented in Table 6, for a configuration with 30 level-3 landmark universes. We observe that Canal provides between 94% and 98% accuracy in all categories, meaning that at least 94% of the time, Canal is able to find sufficient available credit when the original Bazaar implementation did as well. We further explore the sensitivity of Canal's accuracy to configuration parameters in Figure 12, where we vary the number of landmark universes and the level of each universe for the Home category. We observe that accuracy over 95% can be achieved with 20 level-3 landmark universes, suggesting that even a modest number of landmark universes is likely to be sufficient to deploy Canal in practice.

### 6.2 Ostra

Next, we explore integrating Canal into Ostra [24], a system designed to prevent unwanted communication. Ostra was

| Category | Accuracy |
|---|---|
| Clothes | 94.2% |
| Home | 97.0% |
| Collectables | 97.6% |
| Electronics | 95.4% |
| Computing | 95.9% |

**Table 6.** Accuracy of Bazaar implementation using Canal in each category, relative to the original Bazaar implementation. Canal provides high accuracy for Bazaar, implying that users are rarely impacted by the approximate available credit that Canal finds.

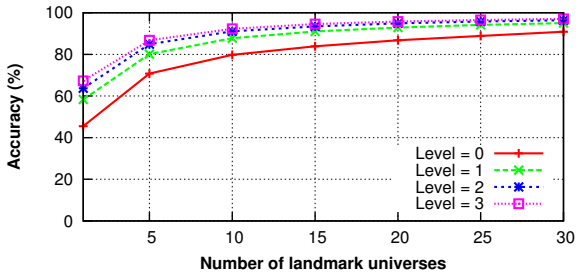


**Figure 12.** Accuracy of Bazaar with Canal for the Home category, for varying numbers of landmark universes and universe levels. Over 95% accuracy can be achieved with 20 level-3 landmark universes.

originally evaluated on a social network derived from largest strongly connected component of YouTube network [23]; this network consists of 446 K nodes and 3.4 M links [24]. A synthetic communication trace was generated using statistics of a real email trace. We re-run the original Ostra experiments using the same input data, and evaluate accuracy and speed up of our implementation compared to the original one. We use a configuration of Ostra with 128 randomly selected nodes as spammers in the system (each of whom tries to send 500 spam messages), with a credit limit of 3 on every link, and with a 1% false email classification probability by good users.

We first examine the speedup that is observed with Canal deployed to Ostra. Presented in Table 7, the results show that if Ostra were to use Canal, the average time taken to find a path with available credit would drop from 35.4 milliseconds to 190 microseconds (a relative speedup of over 186 times). We observe a lower speedup when Canal is applied to Ostra, compared to Bazaar, for two reasons: First, Ostra requires only a single path for every transaction, while Bazaar generally requires the use of multiple paths, and second, the attacker strength is lower for Ostra (just 128 attackers each attempting to send 500 messages).

We now examine the accuracy that Canal provides when deployed with Ostra. Similar to the evaluation with Bazaar, we calculate the fraction of transactions for which the original version of Ostra was able to find a path with available credit, but Canal is not. The results of this experiment for different configurations of the number of landmark universes and the universe level is presented in Figure 13. We see that

| Original Avg. | Canal Median | Canal 95th Percentile | Relative Speedup |
|---|---|---|---|
| 35.4 | 0.05 | 1.4 | 186 × |

**Table 7.** Time in milliseconds required to process a credit network transaction in Ostra in Canal with 30 level-3 landmark universes. Also included is the original processing time from the Ostra paper and the relative speedup.

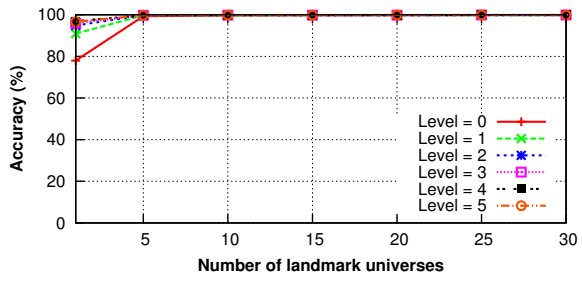


**Figure 13.** Accuracy of the Ostra implementation using Canal, for varying numbers of landmark universes and universe levels. Over 99% accuracy can be achieved once 5 landmark universes are used.

Canal provides over 99% accuracy once at least five landmark universes are used.

### 6.3 Summary

Overall, the results in this section demonstrate that Canal can be easily integrated into existing Sybil tolerance schemes. Moreover, the results show that implementations of both Ostra and Bazaar with Canal achieve significant speedup while providing an approximation that only rarely impacts the credit available to users. Given the previous high computation cost of these systems, Canal opens the door for schemes like Ostra, Bazaar, and SumUp to be deployed in real systems, with on-demand computations done over highly-dynamic credit networks.

## 7. Conclusion

We have presented Canal, a system that can efficiently and accurately transfer credit payments over large credit networks. Canal is designed to complement existing Sybil tolerance schemes such as Ostra [24], SumUp [35], TrustDavis [9], and Bazaar [29]. We argued that these schemes are all based on computing payments over credit networks, a computation that requires computing max-flow over a graph, that leads to significant computational complexity and makes them impractical to deploy on real-world sites. With Canal, these schemes see a dramatic speedup, making them practical for real-world use.

An evaluation demonstrated that Canal's approximation rarely impacts honest users and does not allow malicious users to obtain any additional credit. Furthermore, Canal is able to perform payment calculations in under a few millisecond on graphs with hundreds of millions of links, a massive speedup when compared to existing approaches. Finally, we demonstrated that, were existing Sybil tolerance schemes to use Canal, the time necessary to process credit payments would be reduced by multiple orders of magnitude while achieving over 94% accuracy.

## Acknowledgements

We thank the anonymous reviewers, Allen Clement, Peter Druschel, and our shepherd, Emin Gün Sirer, for their helpful comments. This research was supported in part by NSF grant IIS-0964465, a Google Research Award, and an Amazon Web Services in Education Grant.

## References

[1] M. Abadi, M. Burrows, M. Manasse, and T. Wobber. Moderately hard, memory-bound functions. In *ACM Transactions on Internet Technology*, volume 5, pages 299–314, 2005.

[2] N. Borisov. Computational puzzles as Sybil defenses. In *Proceedings of the 6th IEEE International Conference on Peer-to-Peer Computing (IEEE P2P'06)*, 2006.

[3] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. S. Wallach. Secure routing for structured peer-to-peer overlay networks. In *SIGOPS Operating Systems Review*, volume 36, pages 299–314, 2002.

[4] H. Chun, H. Kwak, Y.-H. Eom, Y.-Y. Ahn, S. Moon, and H. Jeong. Comparison of online social relations in volume vs interaction: A case study of cyworld. In *Proceedings of the 8th ACM/USENIX Internet Measurement Conference (IMC'08)*, 2008.

[5] P. Dandekar, A. Goel, R. Govindan, and I. Post. Liquidity in credit networks: A little trust goes a long way. In *Proceedings of the 12th ACM Conference on Electronic Commerce (EC'11)*, 2011.

[6] G. Danezis and P. Mittal. SybilInfer: Detecting Sybil nodes using social networks. In *Proceedings of the 16th Network and Distributed System Security Symposium (NDSS'09)*, 2009.

[7] A. Das Sarma, S. Gollapudi, M. Najork, and R. Panigrahy. A sketch-based distance oracle for web-scale graphs. In *Proceedings of the 3rd ACM International Conference of Web Search and Data Mining (WSDM'10)*, 2010.

[8] E. A. Dinic. An algorithm for the solution of the max-flow problem with the polynomial estimation. *Doklady Akademii Nauk SSSR*, 194(4), 1970.

[9] D. do B. DeFigueiredo and E. T. Barr. Trustdavis: A non-exploitable online reputation system. In *Proceedings of the 7th IEEE International Conference on E-Commerce Technology (IEEE E-Commerce)*, 2005.

[10] J. Douceur. The Sybil Attack. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS'02)*, 2002.

[11] L. R. Ford and D. R. Fulkerson. Maximal flow through a network. In *Canadian Journal of Mathematics*, volume 8, pages 399–404.

[12] A. Ghosh, M. Mahdian, D. M. Reeves, D. M. Pennock, and R. Fugger. Mechanism design on trust networks. In *Proceedings of the 3rd International Conference on Internet and Network Economics (WINE'07)*, 2007.

[13] A. V. Goldberg and R. E. Tarjan. A new approach to the maximum flow problem. In *Proceedings of the 18th annual ACM Symposium on Theory of Computing (STOC'86)*, 1986.

[14] R. E. Gomory and T. Hu. Multi-terminal network flows. In *Journal of the Society for Industrial and Applied Mathematics (SIAM)*, volume 9, pages 551–570, 1961.

[15] D. Gregor and A. Lumsdaine. The parallel BGL: A generic library for distributed graph computations. In *Proceedings of*

*the Parallel Object-Oriented Scientific Computing (POOSC)*, 2005.

[16] A. Gubichev, S. Bedathur, S. Seufert, and G. Weikum. Fast and accurate estimation of shortest paths in large graphs. In *Proceedings of the 19th ACM international conference on Information and knowledge management (CIKM'10)*, 2010.

[17] T. Hartmann and D. Wagner. Fully-dynamic cut tree construction. Technical Report 2011.25, Karlsruhe Institute of Technology, 2011.

[18] J. Jiang, C. Wilson, X. Wang, P. Huang, W. Sha, Y. Dai, and B. Y. Zhao. Understanding latent interactions in online social networks. In *Proceedings of the 10th ACM/USENIX Internet Measurement Conference (IMC'10)*, 2010.

[19] J. Leskovec, K. Lang, and M. Mahoney. Empirical comparison of algorithms for network community detection. In *Proceedings of the 19th International World Wide Web Conference (WWW'10)*, 2010.

[20] C. Lesniewski-Laas and M. F. Kaashoek. Whānau: A Sybil-proof distributed hash table. In *Proceedings of the 7th Symposium on Networked Systems Design and Implementation (NSDI'10)*, 2010.

[21] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. Pregel: A system for large-scale graph processing. In *Proceedings of the International Conference on Management of Data (SIGMOD'10)*, 2010.

[22] A. Mislove, H. S. Koppula, K. P. Gummadi, P. Druschel, and B. Bhattacharjee. Growth of the Flickr social network. In *Proceedings of the 1st ACM SIGCOMM Workshop on Social Networks (WOSN'08)*, 2008.

[23] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee. Measurement and analysis of online social networks. In *Proceedings of the 7th ACM/USENIX Internet Measurement Conference (IMC'07)*, 2007.

[24] A. Mislove, A. Post, K. P. Gummadi, and P. Druschel. Ostra: Leveraging trust to thwart unwanted communication. In *Proceedings of the 5th Symposium on Networked Systems Design and Implementation (NSDI'08)*, 2008.

[25] M. Mitzenmacher and E. Upfal. *Probability and Computing*. Cambridge University Press, Cambridge, UK, 2005.

[26] A. Mohaisen, A. Yun, and Y. Kim. Measuring the mixing time of social graphs. In *Proceedings of the 10th ACM/USENIX Internet Measurement Conference (IMC'10)*, 2010.

[27] M. Mondal, B. Viswanath, A. Clement, P. Druschel, K. P. Gummadi, A. Mislove, and A. Post. Limiting large-scale crawls of social networking sites. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication (SIGCOMM'11, poster session)*, 2011.

[28] M. Motoyama, D. McCoy, K. Levchenko, S. Savage, and G. M. Voelker. Dirty jobs: The role of freelance labor in web service abuse. In *Proceedings of the 20th USENIX conference on Security (SEC'11)*, 2011.

[29] A. Post, V. Shah, and A. Mislove. Bazaar: Strengthening user reputations in online marketplaces. In *Proceedings of the 8th Symposium on Networked Systems Design and Implementation (NSDI'11)*, 2011.

[30] D. Quercia and S. Hailes. Sybil attacks against mobile users: Friends and foes to the rescue. In *Proceedings of the 29th Conference on Information Communications (INFOCOM'10)*, 2010.

[31] S. Seuken and D. C. Parkes. On the Sybil-proofness of accounting mechanisms. In *Proceedings of the 6 th Workshop on the Economics of Networks, Systems and Computation (NetEcon'11)*, 2011.

[32] T. Stein, E. Chen, and K. Mangla. Facebook immune system. In *Proceedings of the 4th Workshop on Social Network Systems (SNS'11)*, 2011.

[33] D. Streitfeld. Ferreting out fake reviews online. `http://nytimes.com/2011/08/20/technology/finding-fake-reviews-online.html`.

[34] N. Tran, J. Li, L. Subramanian, and S. S. Chow. Optimal Sybil-resilient node admission control. In *Proceedings of the 30th Conference on Information Communications (INFOCOM'11)*, 2011.

[35] N. Tran, B. Min, J. Li, and L. Subramanian. Sybil-resilient online content voting. In *Proceedings of the 6th Symposium on Networked Systems Design and Implementation (NSDI'09)*, 2009.

[36] P. Tsuchiya. The landmark hierarchy: A new hierarchy for routing in very large networks. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication (SIGCOMM'88)*, 1988.

[37] L. G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 1990.

[38] B. Viswanath, A. Mislove, M. Cha, and K. P. Gummadi. On the evolution of user interaction in Facebook. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Social Networks (WOSN'09)*, 2009.

[39] B. Viswanath, M. Mondal, A. Clement, P. Druschel, K. P. Gummadi, A. Mislove, and A. Post. Exploring the design space of social network-based Sybil defense. In *Proceedings of the 4th International Conference on Communication Systems and Network (COMSNETS'12)*, 2012.

[40] B. Viswanath, A. Post, K. P. Gummadi, and A. Mislove. An analysis of social network-based Sybil defenses. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication (SIGCOMM'10)*, 2010.

[41] K. Walsh and E. G. Sirer. Experience with a distributed object reputation system for peer-to-peer filesharing. In *Proceedings of the 3rd Symposium on Networked Systems Design and Implementation (NSDI'06)*, 2006.

[42] Z. Yang, C. Wilson, X. Wang, T. Gao, B. Y. Zhao, and Y. Dai. Uncovering social network Sybils in the wild. In *Proceedings of the 11th ACM/USENIX Internet Measurement Conference (IMC'11)*, 2011.

[43] H. Yu. Sybil defenses via social networks: A tutorial and survey. *SIGACT News*, 42(3), 2011.

[44] H. Yu, P. B. Gibbons, M. Kaminsky, and F. Xiao. SybilLimit: A near-optimal social network defense against Sybil attacks. In *Proceedings of the IEEE Symposium on Security and Privacy (IEEE S&P'08)*, 2008.

[45] H. Yu, M. Kaminsky, P. B. Gibbons, and A. Flaxman. SybilGuard: Defending against Sybil attacks via social networks. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication (SIGCOMM'06)*, 2006.

[46] C. M. Zhang and V. Paxson. Detecting and analyzing automated activity on twitter. In *Proceedings of the 12th International Conference on Passive and Active Measurement (PAM'11)*, 2011.