FaceTrust: Collaborative Threat Mitigation Using Social Networks

Michael Sirivianos
Duke University
msirivia@cs.duke.edu

Xiaowei Yang
Duke University
xwy@cs.duke.edu

Kyungbaek Kim
University of California, Irvine
kyungbaek@uci.edu

Abstract

Unwanted traffic mitigation can be broadly classified into two main approaches: a) centralized security infrastructures that rely on a limited number of trusted monitors to detect and report malicious traffic; and b) highly distributed systems that leverage the experiences of multiple nodes within distinct trust domains. The first approach offers limited threat coverage and slow response times. The second approach is not widely adopted, partly due to the lack of guarantees regarding the trustworthiness of nodes that comprise the system.

Our proposal, FaceTrust, aims to achieve the trustworthiness of centralized security services and the wide coverage and responsiveness of large-scale collaborative threat mitigation. FaceTrust is a large-scale peer-to-peer system designed to rapidly propagate behavioral reports concerning Internet entities (e.g., hosts, email signatures etc.). A FaceTrust node builds trust for its peers by auditing their behavioral reports and by leveraging the social network of FaceTrust administrators. A FaceTrust node combines the confidence its peers have in their own reports and the trust it places on its peers to derive the likelihood that the entity is malicious (e.g. being a spam bot).

The simulation-based evaluation of our approach indicates its potential under a real-world deployment: during a simulated spam campaign, FaceTrust nodes characterized 71% of spam bot connections as such with confidence greater than 75%.

1 Introduction

The majority of the currently deployed Internet threat mitigation techniques rely on centralized infrastructures and place trust on a small number of security authorities. For instance, email systems and browsers rely heavily on a few centralized IP [7] and web site [4,5] reputation services. End-users rely on software vendors to update their anti-virus/malware tools or to release updates that patch their systems' vulnerabilities.

Unfortunately, centralized services are often found to maintain out-dated blacklists [42] or respond slower than the worm infection rate [36, 37], offering a rather large window of opportunity to attackers. Moreover, the vantage points of centralized services are limited in numbers, but attacks launched using large botnets are becoming increasingly surreptitious. In those attacks, one malicious host may attack multiple domains, each for a short period of time [27,41], reducing the effectiveness of malicious traffic detection with a small number of vantage points.

Motivated by this problem, researchers have proposed collaborative worm containment systems [17, 61] and peer-to-peer spam filtering platforms [58, 59] to achieve rapid and reliable detection and suppression of unwanted traffic. These early systems assume compliant behavior from all participating peers, which is hardly true given the heterogeneity of the Internet. Compromised hosts controlled by attackers may join the system, polluting the detection mechanisms. In addition, honest peers may become compromised after they join the system.

We propose a collaborative peer-to-peer threat mitigation system (FaceTrust) that uses social trust embedded in online social networks (OSN) to defend against malicious peers. Each FaceTrust node disseminates *behavioral reports*, i.e. security alerts regarding Internet entities, such as IP addresses, packet fingerprints, and email signatures that it observes to its peers (Section 3). The goal of the system is to ensure that the behavioral reports reach the destination nodes faster than the threat itself, and that they are credible enough to warrant action by their receivers. Each FaceTrust node associates a trust score to another FaceTrust node and uses this score to assess the trustworthiness of the behavioral reports originated by the other node.

FaceTrust is designed to be a peer-to-peer system comprising of nodes across different trust domains. The large scale of the Internet inhibits a node from directly assessing the trustworthiness of all participating nodes. However, in order for the system to be effective in mitigating numerous, heterogeneous, and fast-spreading threats, it must enable a FaceTrust node to interact with a large number of other FaceTrust participants, while being aware of their trustworthiness.

To address this challenge, FaceTrust uses social trust to bootstrap direct trust assessments, and then employs a lightweight reputation system to evaluate the trustworthiness of nodes and their behavioral reports (Section 3.3). Our insight is that each FaceTrust node will be admin-

istered by human administrators (admins), and nodes maintained by trusted admins are likely to disseminate trustworthy reports. Therefore, a FaceTrust node may obtain a direct trust assessment with a number of nodes with whom its admin has social relationships. Social relationships between admins can be obtained from massive Online Social Network (OSN) providers, such as Facebook and MySpace. FaceTrust then uses a reputation management system [23, 34] to aggregate the nodes' collective experiences in order to enable one node to form opinions of another node's trustworthiness despite the lack of past history of interactions.

Reputation systems are known to be vulnerable to the Sybil attack [21]. Sybil attacks subvert distributed systems by introducing numerous malicious identities under the control of an adversary. Through the use of these identities the adversary acquires disproportional influence over the system. To mitigate this attack, FaceTrust again uses the social network to assess the probability that a node is a Sybil attacker, i.e. its *identity trust* (Section 3.2) Each FaceTrust node's identity is associated with its admin's identity. The latter is verified through the social network using a SybilLimit-like technique [55], which is shown to be effective in identifying sybils among social network users.

Our design enables a FaceTrust node to use both the identity trust and the reputation of another node to assess the overall trustworthiness of the other node's behavioral report. The originator of a behavioral report itself also assigns a confidence level to the report, as traffic classification has a level of uncertainty. The trustworthiness of a node and its confidence level in a report determines whether a node should trust a report or ignore it. Trusted reports can be used for diverse purposes, depending on the FaceTrust node's function. For example, email servers can use them to automatically filter out email messages that originate from IPs that have been designated as spammers with high confidence. IDS systems can use them to block packets that originate from suspicious IPs or have signatures that have been designated as malicious.

We evaluate our design (Section 4) using a $100K$-node sample of the Facebook social network. We first evaluate the effectiveness of our SybilLimit-like technique in mitigating Sybil attackers. Our results show that honest nodes have ~ 0.89 average identity trust, whereas Sybil nodes in clusters with more than 200 nodes have less than 0.05 average identity trust. We also show that it is practical for a typical centralized social network provider to support the proposed identity trust primitive. In addition, we demonstrate through simulation that collaborating FaceTrust nodes are able to suppress common types of unwanted traffic in a reliable and responsive manner. Our simulations show that in a $100K$-node FaceTrust network with only 10% of nodes having spam classification capability, nodes with no local spam detection capability are able to identify 71% of connections from spammers with greater than 75% confidence. In a similar setting, where only 10% of the nodes are able to classify worms, behavioral reports that carry worm signatures are disseminated faster than the rate at which the worm spreads.

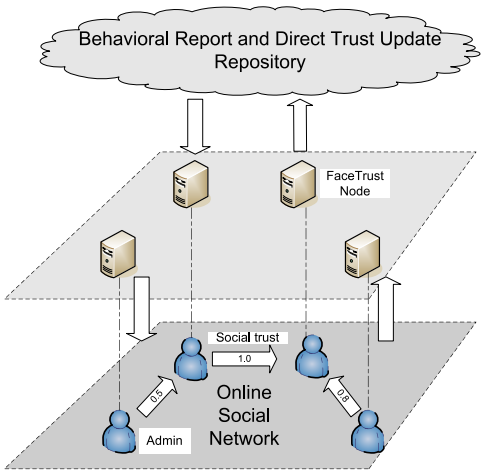

Figure 1: FaceTrust architecture.

2 System Overview

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

2.1 FaceTrust Components

Figure 1 depicts FaceTrust's architecture. At a high-level, the FaceTrust system comprises the following components: 1) human users that administer networked devices/networks (*admins*) and join a social network; 2) end systems that are administered by specific admins and participate in monitoring and reporting the behavior of entities (*FaceTrust nodes*); 3) behavioral reports submitted by FaceTrust nodes concerning entities they observe; 4) direct trust updates made available by FaceTrust nodes reporting their perceived trustworthiness of their peers; and 5) a distributed repository that receives and stores behavioral reports and trust updates.

The same administrator that administers a FaceTrust node also administers a group of applications that interface with the node. These applications may be equipped with Internet traffic monitoring and characterization functionality, i.e., they are able to detect email spam, port scanning traffic, DoS activity etc. Examples of applications with such monitoring and charac-

terization functionality are honeypots [47, 51], backscatter/darknet traffic detection infrastructure [13, 47] and worm detection and containment systems [19, 45, 50]. They may also be applications that utilize concrete descriptions of unwanted traffic to filter it out, such as email filters or IDS systems.

2.2 Behavioral Reports

A traffic characterization application uses the `Report(entityID, action, confidence)` call of the FaceTrust node RPC API to feedback its observed behavior of an entity. The first argument identifies the source of the threat (e.g. an IP address, a packet fingerprint etc.) The second argument determines the type of security threat the report concerns (e.g. "is a worm", "is a spam bot" etc), and the third argument is the confidence with which the application is reporting that the specified entity is performing the specified action. The latter takes values in 0% to 100% and reflects the fact that in many occasions traffic classification has a level of uncertainty. For example, a mail server that sends both spam and legitimate email may or may not be a spamming host.

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

$$[\text{behavioral report}]\ I,\ is\ spam\ bot,\ 50\%$$

In addition, FaceTrust nodes are able to revoke behavioral reports by updating them. If for example at a later time, i determines that no spam originates from I, it sends a new behavioral report in which it updates the confidence value from 50% to 0%. Each report is signed by the reporting FaceTrust node's public key (Section 3.1) for authentication and integrity.

2.3 Trusting an Entity

Each FaceTrust node assigns a *peer trust* value to each of its peers in the network. This trust value determines the trustworthiness of the peer's behavioral reports. The *peer trust* is determined using two dimensions of trust: a) *reporter trust*; and b) *identity trust*. We describe these two dimensions below. The receiver of a behavioral report derives its confidence in the correctness of the received report from the behavioral report's confidence and the peer trust.

FaceTrust nodes collectively compute *reporter trust* values by employing a reputation management mechanism. This mechanism relies on FaceTrust nodes verifying each other's behavioral reports to derive individual *direct trust* values (Section 3.3). If a node i is able to verify the behavioral reports of node j it can determine a direct trust value d_{ij}. The FaceTrust nodes share these values with other FaceTrust nodes by exchanging *direct trust updates*. For reasons of scalability and efficiency, a node considers the behavioral reports of only a (possibly random) subset V (including itself) of all the nodes in the FaceTrust network. Consequently, nodes submit and retrieve direct trust updates only for nodes in V. We refer to V as a node's *view*.

FaceTrust also relies on the fact that nodes comprising Internet systems such as email servers, honeypots, IDS etc are administered by human admins. These human users maintain accounts in online social networks (OSN). FaceTrust leverages OSNs in the following two ways: a) it defends against Sybil attacks [21] by exploiting the fact that OSNs can be used for resource testing, where the test in question is a Sybil attacker's ability to create and sustain acquaintances. Depending on the result of the test, the OSN provider assigns an *identity trust* value to the admin; and b) It initializes the *direct trust* values in the absence of prior interactions between FaceTrust nodes, by considering the trust that is inferred by associations in the social network of administrators.

Finally, an application can use the `GetTrust(entityID, action)` call of the FaceTrust node RPC API to obtain a trust metric that corresponds to the likelihood of an entityID performing the specified malicious action. The FaceTrust node derives this metric by aggregating behavioral reports regarding entityID. These behavioral reports are weighted by the reporting node's *peer trust*.

2.4 Security Challenges

FaceTrust is a collaborative platform aiming at suppressing malicious traffic. As such, it is reasonable to assume that FaceTrust itself will be targeted in attempts to disrupt its operation. FaceTrust is an open system, meaning that any admin with a social network account and a device can join. Due to its highly distributed and open nature, it faces the following challenges:

False Behavioral Reports. Malicious FaceTrust nodes may issue false or forged reports aiming at reducing the system's ability to detect unwanted traffic or disrupting legitimate Internet traffic.

Behavioral Report Suppression or Alteration. We implement the distributed repository for behavioral reports as a Distributed Hash Table (Section 3.5). Consequently, the system is vulnerable to attacks on DHT routing [18] aiming at preventing legitimate nodes from retrieving important behavioral reports. In addition, since the reports may be stored by malicious nodes such nodes may attempt to alter their content.

False Direct Trust Updates. To address false behavioral reports, FaceTrust employs a reporter reputation system to determine the amount of trust that should be placed on each user's reports. However, the reputation system itself is vulnerable to false reporting as malicious nodes may send false or forged direct trust updates.

Sybil Attack. An adversary may attempt to create multiple FaceTrust identities aiming at increasing its ability to subvert the system using false behavioral reports and direct trust updates. Defending against Sybil attacks without a trusted central authority is hard. Many decentralized systems try to cope with Sybil attacks by binding an identity to an IP address. However, malicious users can readily harvest IP addresses through BGP hijacking [41] or by commanding a large botnet.

3 FaceTrust Design

In this section we describe the design of our system.

3.1 OSN Providers as Certification Authorities

For an open system such as FaceTrust to operate reliably, node accountability in the form of node authentication and prevention of Sybil attacks is of the utmost importance.

Existing certification authorities are not suitable for open, large scale P2P architectures because they require users to pay a certification fee ($\sim$\$20 for class 1 certificate). In addition to CAs being expensive, they currently represent a monopoly for a very important Internet primitive, and we consider breaking this monopoly beneficial.

We propose to leverage existing OSN repositories as inexpensive, Sybil-mitigating certification authorities. OSNs are ideally positioned to perform such function because using SybilLimit-like techniques (see Section 3.2) they can determine the amount of confidence one can place on a node's identity. We refer to this confidence as *identity trust*. They are able to perform inexpensive resource tests by analyzing the centrally maintained social graph.

Each node that participates in FaceTrust is administered by human users that have accounts with Online Social Network providers. The system needs to ensure that each user's social network identity is closely coupled with its FaceTrust node. To this end, FaceTrust uses an OSN application as a front-end to the social network that the human user has joined.

Every FaceTrust admin obtains a public/private key pair that is associated with its social network identity. It obtains its public key in the form of a public key certificate that is signed by the trusted OSN provider (e.g. signed with Facebook's public key). This certification mechanism closely resembles a typical X.509 PKI infrastructure, in which the hierarchy of certificates is always a top-down tree, with a root certificate at the top, representing a CA that is so central to the scheme that it does not need to be authenticated by some trusted third party.

The certifying OSN provider can either be the provider of existing popular OSN (e.g. Facebook) or a third-party OSN application (e.g. Facebook application [3]) provider. In the first case, the popular OSN provider has access to the complete social network, and augments the OSN application API to allow applications to query the identity trust of the OSN's users. OSN providers are incented to provide this service because it adds value to their service, making the service more attractive to subscribers. In the second case, a third-party deploys FaceTrust as an OSN application and has access only to the social network of users using FaceTrust. Although the FaceTrust-only social network is not as complete as the OSN provider's one, the fact that it is smaller makes its analysis less computationally expensive. In addition, it does not require the adoption of the service by the OSN provider. For ease of exposition, for the rest of the paper we refer to both the popular OSN provider and the third-party OSN application provider as *OSN provider*.

To defend against forged or falsifiedbehavioral reports and direct trust updates, FaceTrust nodes sign messages that originate from them using their private key. When a node i exchanges direct trust updates with a node j, or receives behavioral reports that are claimed to originate from j, i acts as follows. If i does not have j in its view, it obtains j's public key certificate from j and verifies it using the OSN provider's certificate. After verifying j's public key, i verifies the signed direct trust update or behavioral report that is claimed to originate from j.

3.2 Determining the Identity Trust

There is typically one-to-one correspondence between a real user's social network identity and its real identity. Although, malicious users can create many identities, they can establish only a limited number of trust relationships with real humans. Thus, groups of Sybil attackers are connected to the rest of the social graph with a disproportionally small number of edges. The first works to exploit this property was SybilGuard and SybilLimit [55, 56], which bound the number of Sybil identities using a fully distributed protocol.

Based on a similar concept, FaceTrust's Sybil detection algorithm determines the strength of a FaceTrust user's identity. This algorithm is executed solely by the OSN provider over its centrally maintained social graph $\mathcal{I}$. An admin's i identity is considered weak if it has not established a sufficient number of real relationship's

over the social network. Upon being queried by an admin v, the OSN provider returns a value in $[0.0, 1.0]$, which specifies the confidence of the provider that a specific node s is not participating in a Sybil attack, i.e. the probability that s is not part of a network of Sybils.

First, we provide some informal background on the theoretical justification of SybilGuard and SybilLimit. It is known that randomly-grown topologies such as social networks and the web are fast mixing small-world topologies [11, 25, 53]. Thus in the social graph $\mathcal{I}$ with n nodes, a walk of $\Theta(\sqrt{n}\log n)$ steps contains $\Theta(\sqrt{n})$ independent samples approximately drawn from the stationary distribution. When we draw random walks from a verifier node v and the suspect s, if these walks remain in a region of the network that honest nodes reside, both walks draw $\Theta(\sqrt{n})$ independent samples from roughly the same distribution. It follows from the generalized Birthday Paradox [56] that they intersect with high probability. The opposite holds if the suspect resides in a region of Sybil attackers that is not well-connected to the region of honest nodes.

SybilGuard replaces random walks with "random routes" and a verifier node accepts the suspect if random routes originating from both nodes intersect. In random routes, each node uses a pre-computed random permutation as a one-to-one mapping from incoming edges to outgoing edges. We refer to this random permutation as routing table. As a result, two random routes entering an honest node along the same edge will always exit along the same edge ("convergence property"). This property guarantees that random routes from a Sybil region that is connected to the honest region through a single edge will traverse only one distinct path, further reducing the probability that a Sybil's random routes will intersect with a verifier's random routes. SybilLimit [55] is a near-optimal improvement over the SybilGuard algorithm. In SybilLimit, a node accepts another node only if random routes originating from both nodes intersect at their last edge. For two honest nodes to have at least one intersected last edge with high probability, the required number of the random routes from each node should be approximately $r = \Theta(\sqrt{m})$, where m is the number of edges in $\mathcal{I}$. The length of the random routes should be $w = O(\log n)$.

With FaceTrust's SybilLimit-like technique the OSN provider computes an identity trust score for each node s in the social graph $\mathcal{I}$. At initialization time, the OSN provider selects l random verifier nodes. It also creates $2r$ independent instances of pre-computed random permutation as a one-to-one mapping from incoming edges to outgoing edges (routing table). The first r routing tables are used to draw random routes from suspect nodes s and the rest r routing tables are used to draw random routes from the verifier nodes v. SybilLimit uses distinct routing tables for verifiers and suspects in order to avoid undesirable correlation between the verifiers' random routes and the suspects' random routes. For each s, the OSN provider runs the SybilLimit-like algorithm is as follows:

1. For each of the l verifiers v, it picks a random neighbor of v. It draws along the random neighbors r random routes of length $w = O(\log n)$, for each instance of the r routing tables, where n is the number of nodes in $\mathcal{I}$. It stores the last edge (tail) of each verifier random route.

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

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

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

Nodes query the OSN provider for the identity trust of their peers. The OSN provider performs the above computations periodically and off-line to accommodate for topology changes. The OSN provider stores the result of this computation for each node as a separate attribute.

3.3 Determining the Reporter Trust

Malicious nodes may issue false behavioral reports to manipulate the trust towards entities. In addition, misconfigured nodes may also issue erroneous reports. FaceTrust can mitigate the negative impact of incorrect behavioral reports by assigning higher weights to reports obtained from more trustworthy FaceTrust nodes. Conceptually, each FaceTrust node i maintains a reporter trust value rt_{ij} to every other FaceTrust node j in its view, $j \in V_i$. This trust score corresponds to node i's estimation of the probability that node j's reports are accurate. It is obtained from three sources: trust attainable from online social networks, direct behavioral report verification, and transitive trust.

First, FaceTrust relies on the fact that FaceTrust nodes are administered by human users. Competent and benign users are likely to maintain their nodes secure, and provide honest and truthful reports. The trust on the competency and honesty of human users could be obtained via social networks. FaceTrust admins maintain accounts in

online social networks. An admin i tags her acquaintance admin j with a social trust score st_{ij} in $[0.0, 1.0]$ based on her belief on j's ability to manage her FaceTrust node(s). This value is used to initialize a direct trust score between two FaceTrust nodes i and j: $d_{ij} = st_{ij}$.

Second, a FaceTrust node i dynamically updates the direct trust d_{ij} by comparing behavioral reports submitted by the node j with its own reports. A node i may verify a report from a node j for an entity e, if i has also generated a recent report with respect to the same entity. i may also probabilistically choose to observe e solely for the purpose of verifying reports of another node j. The portion of the received behavioral reports that the FaceTrust nodes verify is a tunable parameter. Intuitively, if i and j share similar opinions on e, i should have a high trust in j's reports. Let v_{ij}^k be a measure of similarity in $[0,\ 1.0]$ between i and j's k_{th} report. A node i may updates its direct trust to j using an exponential moving average:

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

As i verifies a large number of reports from j, the direct trust metric d_{ij}^k gradually converges to the similarity of reports from i and j.

By updating d_{ij} and making it available for retrieval to other nodes, i enables its peers $j \in V_i$ to build their FaceTrust reporter trust graph $T_j(V_j, E_j)$. The reporter trust graph of a node i consists of only the nodes in its view V_i, and its directed edge set E_i consists of the direct trust d_{uv} for each $u, v \in V_i$. If a node u has not released a direct trust update for a node v, d_{uv} is treated as being equal to 0.0.

Third, a FaceTrust node i incorporates direct trust and transitive trust to obtain i's overall trust to j: rt_{ij}. Due to the large number of FaceTrust nodes, the admin of a FaceTrust node i may not tag a social trust s_{ij} to the admin of a node j. Moreover, due to the variety and large number of observed entities, nodes i and j may not have encountered the same entities and are therefore unable to directly verify each other's reports. Furthermore, i can further improve the accuracy of its trust metric for j by learning the opinions of other FaceTrust nodes about j. The overall reporter trust rt_{ij} can be obtained as the maximum trust path in node i's reporter trust graph $T_i(V_i, E_i)$, in which each edge $u \rightarrow v$ is annotated by the direct trust d_{uv}. That is, for each path $p \in P$, where P is the set of all paths between nodes i and j:

$$rt_{ij} = max_{p \in P}(\Pi_{u \rightarrow v \in p} d_{uv}) \quad (2)$$

We use the maximum trust path because it can be efficiently computed with Dijkstra's shortest path algorithm in $O(|E| \log |V|)$ time for a sparse T. In addition, it yields larger trust values than the minimum or average trust path, resulting in faster convergence to high confidence regarding the actions entities perform. Last, it mitigates the effect of misbehaving nodes under-reporting their trust towards honest nodes.

If T_i is very sparse (e.g. $|E_i|$ being smaller or slightly larger than $|V_i|$), a node i may not be able to derive meaningful trust values for its peers in V_i using the report trust graph $T_i(V_i, E_i)$. To remedy this situation, V_i includes the nodes with which i is socially acquainted and has assigned social trust values to. In addition, the node augments T_i with directed edges from itself towards a pre-trusted set N of reliable nodes. The size of the pre-trusted and is almost the same for all nodes in the FaceTrust network. Nodes become aware of those reliable nodes either via word of mouth over the social network or by querying the OSN provider. Nodes do not send direct trust updates for these pre-trusted nodes.

3.4 Determining the Trust for an Entity

As mentioned above, a FaceTrust node i may receive multiple behavioral reports originating from multiple nodes $j \in V$ and concerning the same entity e for the same action a. Each report is marked with a level of confidence c_j of the reporter j. Thus, i needs to aggregate the behavioral reports to determine an overall confidence `GetTrust(`e`,`a`)` that e will perform the action a.

When an admin i receives multiple reports concerning the same entity and action pair (e, a), it derives the overall trust `GetTrust(`e`,`a`)` weighing the behavioral reports' confidence by the peer trust of their reporters.

$$GetTrust(e, a) = \frac{\Sigma_{j \in V_i}\ rt_{ij}\ id_j\ c_j(e, a)}{\Sigma_{j \in V_i}\ rt_{ij}\ id_j} \quad (3)$$

3.5 FaceTrust Repository

A node can exchange behavioral reports and direct trust updates with any other node in the FaceTrust network regardless of whether the admins of the nodes are acquaintances in the social network. With this design choice, we ensure that behavioral reports and direct trust updates reach the interested nodes on time, improving the threat coverage of our system. We also enable users that are not well-connected in the social network to peer with other trustworthy nodes.

Under typical deployment scenarios and for varying types of unwanted traffic, nodes often exchange behavioral reports about many malicious entities. The frequency with which behavioral reports and trust updates are submitted and retrieved would impose a significant scalability challenge to a centralized FaceTrust repository. Therefore, although we maintain the FaceTrust social network in a centralized manner, we design a dis-

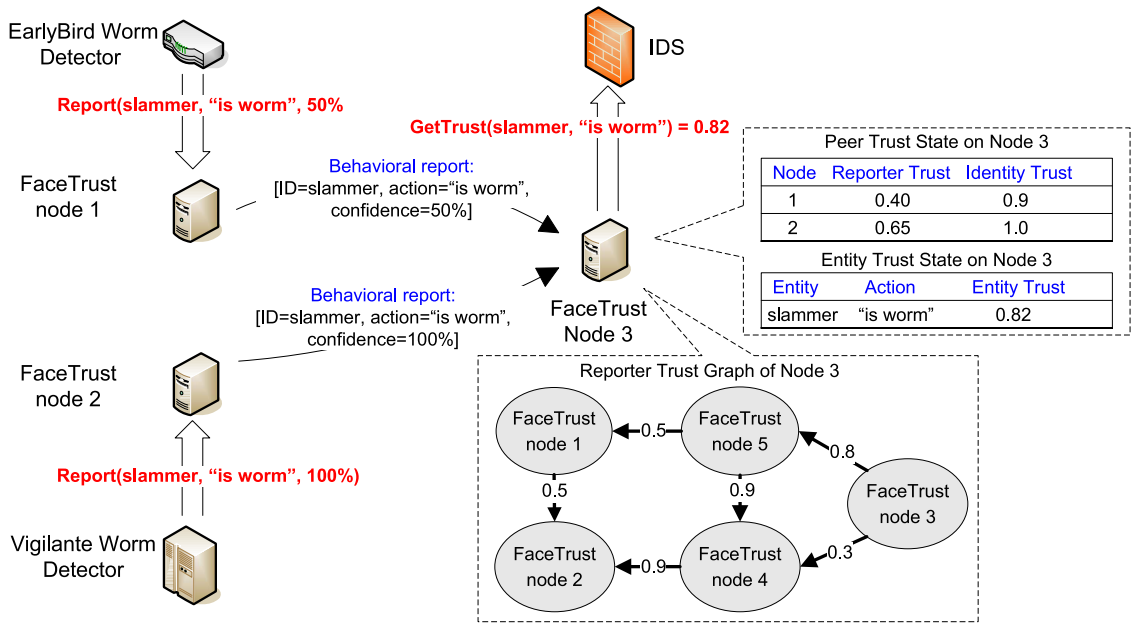

Figure 2: Example of the operation of a small FaceTrust network.

tributed repository to which nodes submit to and retrieve from behavioral reports and direct trust updates.

Our distributed repository consists of two portions, one for behavioral reports and one for direct trust updates. The portion of the repository tasked with maintaining behavioral reports is implemented as a a Distributed Hash Table, e.g Chord [48]. Nodes form a DHT to store and retrieve behavioral reports updates concerning nodes in their view. When a node queries for Behavioral reports it is interested on the reports for a single entity/action pair. These reports are sent by multiple nodes, thus for efficiency it is reasonable to index(key) them based on the hash of the concatenation of the entity's ID (e.g IP) and the action description. When a FaceTrust node i encounters a specific entity, it queries the DHT for all the behavioral reports that involve the entity and the action. Once it locates the node that stores those behavioral reports it asks the node for those reports that originate from nodes in V_i.

On the other hand a node needs to retrieve all the direct trust updates involving all the nodes in its view. Thus it is reasonable to index the updates by the node that issues it and store them at the issuing node. A node i needs to explicitly query for the existence of an update involving all node pairs in its view. Thus every time interval D, a node i directly requests from each node $j \in V_i$ to reply with its current non-zero direct trust values d_{jv} towards other nodes in the network. If the difference between the current direct trust metric d_{jv} and the last d_{jv} i retrieved from j is greater than ϵ, j includes this update in his reply to i's request for direct trust updates. The constant ϵ is used to ensure that the nodes do not incur the overhead of communicating the update if it is not sufficiently large.

Employing a DHT in an adversarial environment poses several security challenges [18, 46]. One of these challenges is the "secure node ID assignment", which is addressed in our system by the randomized generation of public/key pairs. This results in nodes not being able to decide what their Chord identifier is. The Sybil attack is addressed through the use of our OSN Certification Authorities; nodes with very low identity trust are barred from participating in the DHT. Nodes with high identity trust exclude nodes with low one from their finger tables, e.g. nodes with identity trust below 0.1 cannot join the ring.

Chord's inherently constraint routing table and secure node ID assignment ensures secure routing table maintenance [18]. This means that each constrained routing table (and neighbor set) has an average fraction of only f random entries that point to nodes controlled by the attacker, where f is the fraction of compromised nodes. To further counter attacks on DHT routing we employ a form of redundant routing [18]. We use multiple DHT map functions (e.g. multiple Chord rings using distinct hash function seeds). A node stores behavioral reports that correspond to its ID under all map functions. It also maintains a distinct forwarding(finger) table for each map function. This technique defeats behavioral report suppression attacks (Section 2.4) through redundancy. The attacker is unlikely to control all the nodes that store the behavioral reports or at least one node in the forwarding path for all rings.

A node i can retrieve the random subset of nodes in its view V_i either from the OSN provider which may act as a tracker of online FaceTrust nodes, or by exchanging contact lists with other nodes.

3.6 FaceTrust Operation Example

Figure 2 depicts an example of the operation of a small FaceTrust network. The network includes a node tasked with checking incoming packets for malicious code, FaceTrust node 3. That node has no inherent packet classification functionality, thus it relies on the other two nodes, 1 and 2, for early warning about malicious traffic coming his way. Node 1 is an EarlyBird [45] network-layer worm detector, which in this example is able to identify and generate the fingerprint (sl) of observed Slammer worm packets and report that this fingerprint is worm (wm) with confidence $c_1(sl, wm) = 50\%$. Node 2 is a Vigilante [19] end-to-end worm detector, which is able to run the malicious code within a sandbox and analyze it. Thereby, node 2 reports with confidence $c_1(sl, is\ wm) = 100\%$ confidence that the fingerprint of the slammer packet is a worm.

Node 3 maintains the depicted reporter trust graph, derived from his 5-node view. This view includes nodes 1 and 2, which send the depicted behavioral reports. It also includes nodes 4 and 5, which do not send any reports in this example. The weighted directed edges in the graph correspond to the direct trust between the peers in node 3's view. From the reporter trust graph and Equation 2, the maximum trust path between nodes 3 and 1 traverses nodes 5 and 1 yielding $rt_{31} = 0.4$. The maximum trust path between 3 and 2 traverses nodes 5, 4 and 2 and yields reporter trust $rt_{32} =\sim 0.65$. The identity trust of nodes 1 and 2 has been computed by the OSN provider to be $id_1 = 0.9$ and $id_2 = 0.8$, respectively. We can now use Equation 3 to compute the trust $GetTrust(sl, wm)$ that the IDS beside node 1 places on FaceTrust to correctly identify the slammer packet as worm:

$$\frac{c_1(sl, wm)rt_{31}id_1 + c_2(sl, wm)rt_{32}id_2}{rt_{31}id_1 + rt_{32}id_2} = 0.82$$

4 Evaluation

In this section we evaluate FaceTrust's ability to mitigate unwanted traffic under varying threat scenarios. First, we evaluate our identity trust mechanism in terms of its ability to mitigate Sybil attacks and its computational cost. Second, we simulate the operation of a FaceTrust P2P network that aims at proactively identifying spam sources. Third we proceed with simulating the operation of a FaceTrust P2P network charged with the task of identifying a containing the Slammer worm. Last we demonstrate the significance of FaceTrust admins appropriately setting the social trust towards their acquaintances for rapid convergence of the system to correct trust values for reporters and entities.

4.1 Sybil-Resistance Evaluation

We experimentally evaluate our identity trust metric. We crawled Facebook [2] to gain an initial insight on how effective SybilLimits is in detecting Sybil attacks in social networks that are likely to resemble the network of FaceTrust admins. Unlike other online social networking sites, Facebook may more closely resemble the network of trust between FaceTrust admins: users do not tend to establish an excessive number of friend relationships and identities are stronger: a) initially it was open only to college and high school students with verifiable emails; and b) its EULA states clearly that they consider creating fake identities as violation of their terms of use.

Our crawled Facebook graph consists of more than 51 million nodes, and we scaled it down to a $100K$-node single connected component graph using the "forest fire" [29] sampling technique. The average hop count of this graph is 6.11 and the diameter of the graph is 19. The number of total undirected edges is 930680 and the average degree of node is 18. We assume that the current Facebook graph does not include a substantial number of Sybils, therefore we consider those 100K nodes to represent the honest region of the social graph.

4.1.1 Identity Trust Evaluation

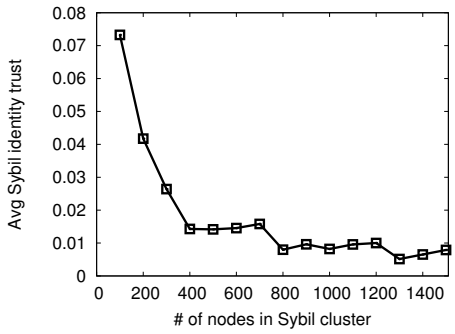

Figure 3: Average identity trust of all Sybil nodes as a function of the number of nodes in the Sybil cluster with one attack edge. In contrast, the average identity trust of honest nodes nodes is ~ 0.89.

We now evaluate the effectiveness of FaceTrust's SybilLimit-like technique in assigning low identity trust values to Sybil nodes. In our evaluation, Sybil attackers form a single well-connected cluster. This cluster has a random graph topology under which Sybil nodes have average degree 14. The Sybil cluster is connected to the

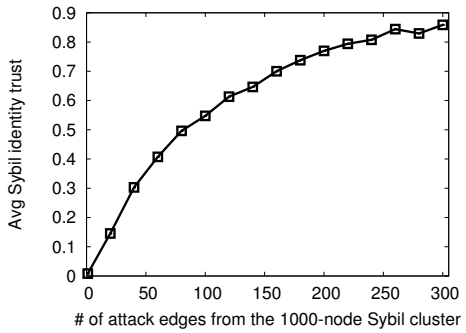

Figure 4: Average identity trust score of all Sybil nodes as a function of the number of attack edges in the Sybil cluster with 1000 Sybil nodes.In contrast, the average identity trust of honest nodes nodes is ~ 0.89.

honest region through *attack edges* between a Sybil node and an honest node. We vary the number of nodes in the Sybil cluster from 100 up to 1500, as well as the number of attack edges from 1 up to 300. We set the length w of random routes equal to 17, the number of routing tables r at each node equals 2600 and the number of verifiers l equal to 100. With these settings, the average identity trust of 1000 randomly chosen honest nodes is 0.89 with 0.10 standard deviation.

In Figure 3, we observe that as the number of nodes in a Sybil cluster increases, the average identity trust of Sybil nodes decreases and stabilizes at around 0.01. The reason is that as the Sybil cluster increases in size, the probability that random routes from Sybil nodes cross attack edges decreases. Figure 4 shows that when the number of attack edges increases, the average identity trust of Sybil nodes in the cluster increases logarithmically. The reason is that the probability that random routes from Sybil nodes and honest nodes cross attack edges increases.

The above results demonstrate that the SybilLimit-like technique assigns much lower identity trust to Sybil nodes than to honest nodes, when the number of attack edges is not too high. The low identity trust for Sybil nodes renders substantially less effective Sybil attacks in which the misbehaving nodes are not well-connected to the social graph.

4.1.2 Cost of Identity Trust Computation

The identity trust of nodes is computed centrally by the OSN provider, and it is critical for the scalability of the system that the cost of this computation is not prohibitively expensive. Table 1 shows the required computation time to derive the identity trust for each node in our $100K$-node sample of the Facebook social network. We profile the identity trust computation implemented in C++ on a Intel Pentium D, 3.40GHz CPU, 2048KB cache and 2GB RAM machine, running Linux 2.6.25-2-68. The identity trust computation consists of three parts, all of which are performed off-line: a) the cost of creating the routing tables at each node; b) the cost of drawing random routes from the suspect and the verifiers (random routing); c) the cost of determining intersections between the verifiers' and the suspects' tails(verification).

Creating one permutation table	0.815699
Random Routing from an honest suspect	0.049568
Random Routing from a Sybil suspect	0.027517
Random Routing from l verifiers	5.199869
Verification for an honest suspect	0.093293
Verification for a Sybil suspect	0.237826

Table 1: Times (sec) for computing the identity trust of a suspect in a 100K node social network with the SybilLimit-like technique.

The cost of creating one permutation table increases linearly with the number of nodes, because the system needs to explore all nodes generating permutation tables. The number of permutation tables for the SybilLimit-like technique is r which should be $\Theta(\sqrt{m})$, where m is the number of edges in the social graph. Consequently, the total cost of creating all permutation tables for the SybilLimit-like technique is $\Theta(n * \sqrt{m})$. Because of the linear increase in computation time, we need to precompute all required permutation tables for the target social network. If we assume that the size of the target social network is 40 million with average node degree 20, r has to be approximately 55KB and 8.8MB storage per node is required. Generally, the social network updates slowly, thus this type of precomputation is practical.

The number of traversed edges during random routing from a node is $\Theta(w * r)$ and its cost is $\Theta(log(n) * \sqrt{m})$, where n is the number of nodes in the social graph. In terms of scalability, as the size of social network increases to $R * n$ from n, this cost increases $\Theta(log(R * n)/log(n) * \sqrt{R})$ times. With a 40-million-node social network, the random routing from an honest suspect takes approximately 1.4 seconds. We can also precompute the random routing and store $2r$ tails, r for a suspect and another r for a verifier. The precomputation of random routing for a 40-million-node social network needs 440KB storage per node.

The verification compares r tails of a suspect with r tails of l verifiers and its cost in our implementation is $\Theta(l * log(\sqrt{m}) * \sqrt{m})$. Similar to random routing, the cost of the verification increases $\Theta(log(r * \sqrt{R})/log(r) * \sqrt{R})$ times, when the size of the social network increases R times. With a 40-million-node social network the cost of the verification for an honest node and a Sybil node would be approximately 2.6 seconds and 6.6 seconds, respectively. In the case of an honest node, there are many possible intersections and a verifier can find one of them

relatively promptly. But a Sybil node has very few intersections and the system typically exhausts all tails without finding any.

Since the computation of identity trust is performed off-line and consists of simple computing jobs, its cost can be greatly reduced by standard parallel computing techniques. From the above, we conclude that OSN providers that have the computational and hosting capability to host million of users are also capable of efficiently performing this computation using their highly scalable infrastructure.

4.2 Reporter Trust Evaluation

We now evaluate FaceTrust's effectiveness in managing the trustworthiness of reporters and their behavioral reports.

We use the SimPy 1.9.1 [38] simulation package to simulate FaceTrust's operation under spam email and worm attack scenarios. We simulate all the protocol operations described in Section 3 as well as an iterative Chord DHT with 3 rings for redundancy. We do not simulate any physical, network or transport layer events (e.g. congestion and packet loss). We repeat each simulation 3 times and we average results over the repetitions.

Nodes verify each other's behavioral reports and update their direct trust using the exponential moving average (Equation 1). We compute the similarity between reports in this evaluation as follows. Node i receives the k^{th} behavioral report from node j that involves an entity action pair (e, a) that both nodes have observed and to which node i and j have assigned confidence $c_i(e, a)$ and $c_j(e, a)$ respectively. Node i computes the similarity v_{ij}^k with node j as follows:

$$v_{ij}^k = \frac{min(c_i(e,a), c_j(e,a))}{max(c_i(e,a), c_j(e,a))} \quad (4)$$

E.g. if node i has sent a behavioral report concerning a packet signature being a worm with 50% confidence and node j has sent a behavioral report concerning the same signature being a worm with 60% confidence, $v_{ij} = 50/60$.

We evaluate our technique for varying view sizes $|V|$. The view of a node is a subset of the 100K-node sampled Facebook network described in Section 4.1. A node sets its direct trust towards nodes with which it is connected in the sampled Facebook network equal to a random value in $[0.0, 1.0]$. In addition, the size of the pre-trusted set $|N|$ (Section 3.3) is set equal to 20 and the social trust assigned to the pre-trusted nodes is equal to 0.9. The view of a node consists of its social acquaintances, the pre-trusted set and a random subset of the sampled network.

4.2.1 Spamming Bots Simulation

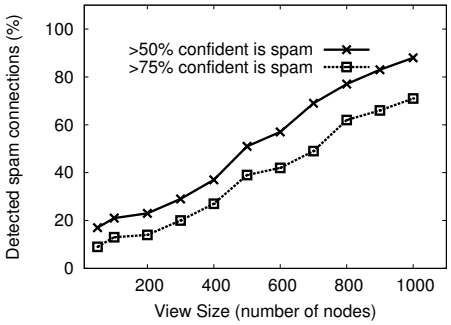

Figure 5: The percentage of spam connections that are detected in the absence of misbehaving FaceTrust nodes as a function of $|V|$. Connections for which a node is 50% or 75% confident that are spam are considered detected.

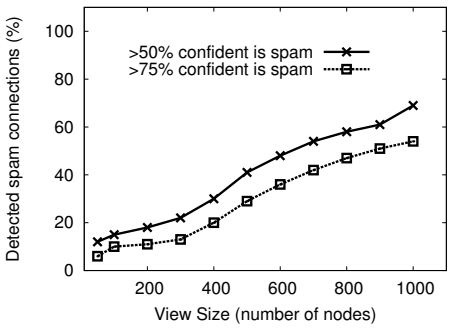

Figure 6: The percentage of spam connections that are detected in the presence of misbehaving FaceTrust nodes as a function of $|V|$. Connections for which a node is 50% or 75% confident that are spam are considered detected.

With this evaluation, we demonstrate FaceTrust's ability to suppress one of the most prevalent types of Internet unwanted traffic: spam email. We simulate the behavior of a spamming botnet as it spams $100K$ nodes (email servers) that belong in a FaceTrust network. The goal of the simulated FaceTrust network is to promptly identify spamming IPs and reject connections from them. $10K$ nodes in the FaceTrust network have the ability to characterize spam upon receiving it, e.g. by subscribing to a commercial email reputation service such as TrendMicro [8]. We refer to these $10K$ nodes as *honest nodes*. We refer to the rest $90K$ nodes as *regular nodes*. Regular nodes cannot detect spam and cannot verify the behavioral reports of other nodes. They rely on honest nodes to warn them about spamming bots.

We draw the simulated spamming botnet behavior partially from [41] and [54]. 1000 spamming bots send uniformly to all IP addresses over the course of 1200 sec. All spamming bots start to send emails at random instances within the first 10 minutes of the simu-

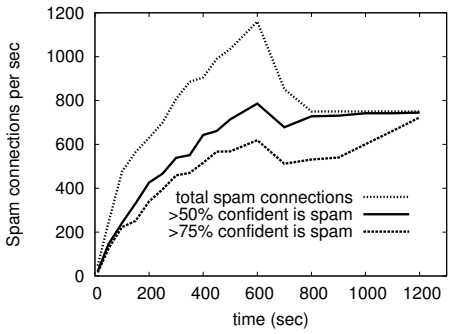

Figure 7: The number of spam connections per second that are characterized as such (true positives) as a function of time, in the presence of misbehaving FaceTrust nodes. We also depict the total number of spam connections. Connections for which a node is 50% or 75% confident that are spam are considered detected.

lation, since spam campaigns are found to be clustered in time [42]. Spamming bots establish 3 connections per second [54]. The parameter α of Equation 1 is set equal to 0.4. The parameter ϵ (Section 3.3) is set equal to 0.2. The size of the pre-trusted set is equal to 20. Lastly, FaceTrust nodes retrieve direct trust updates regarding nodes in their view every $D = 20$ secs. (Section 3.5).

750 of the spamming bots connect to randomly selected nodes for 120 sec. We call these spammers ephemeral.The rest of the bots persist for until the end of the simulation, and we refer to them as persistent. Spammers never attempt to connect to an already visited node.

A node that receives an SMTP connection issues a `GetTrust()` call, which entails the retrieval of relevant behavioral reports submitted by nodes in its view, and computing the trustworthiness of those nodes (Sections 3.3, 3.4). Honest nodes update the direct trust of their peers as soon as they verify their behavioral reports (Equation 1). They issue a `Report()` call to submit a behavioral report about a spamming host with 100% confidence as soon as they receive an SMTP connection from that host. Nodes use the maximum trust path to determine their transitive trust with other nodes in their view (Equation 2).

Figure 5 reports the percentage of spam connections that are perceived as such by the regular FaceTrust nodes throughout the duration of the spam campaign. It is presented as a function of the FaceTrust nodes' view size V. A spamming connection is considered detected if the regular node that receives it considers the requesting node a spam bot with greater than 50 or 75% confidence.

In our simulation, on average a node is spammed by approximately 12 distinct spam hosts. We observe that for view size $|V| = 1k$, the network is able to reject 71% of the spam connections if nodes reject connections from nodes that are perceived as spam bots with 75% confidence. The speed with which confidence regarding spam bots propagates through the network depends on the size of the view: the more nodes a node i has in its view, the larger is the probability that a node that has detected and reported a spam bot earlier is included i's view. On average for $|V| = 1K$, at the end of the simulation a FaceTrust node has 0.81 trust towards honest nodes.

We also observe that for small-sized views, e.g. $|V| = 100$, the system is ineffective in rejecting spam because the probability that another node in the view has encountered the same threat (spamming bot) and is able to offer an early warning is substantially reduced. This observation motivates our design choice not to limit a node's view to comprise only its social acquaintances.

In Figure 6, spamming bots are also misbehaving FaceTrust nodes, which claim 1.0 direct trust for misbehaving nodes in their view. Misbehaving FaceTrust nodes discover honest nodes for which to report negative direct trust by receiving their behavioral reports on known spammers. In addition, misbehaving FaceTrust nodes send behavioral reports for all the spam sources, claiming 0% confidence that they are spammers. Furthermore, misbehaving FaceTrust nodes refuse to store negative behavioral reports regarding spamming hosts or forward DHT queries that involve them.

The $1K$ misbehaving nodes correspond to misconfigured honest nodes or nodes that were compromised after their users joined the social network and established trust relationships. Thus, they are well-connected in the social network, being trusted by their pre-existing acquaintances. They do not correspond to a botnet that joins the overlay and attempts to manipulate it, as such bots would be unable to establish friend links, resulting in them having very low identity trust. Thus, their influence would be substantially diminished (Section 4.1.1).

In this simulation, a misbehaving node is able to identify all honest nodes in its view, the number of which is on average ~ 100 for $|V| = 1K$. For $V = 1K$, a node retrieves on average ~ 3 behavioral reports that originate from honest nodes in its view and ~ 9 behavioral reports originating from misbehaving nodes. At the end the simulation, a FaceTrust node has 0.81 average trust towards honest nodes and 0.04 average trust towards misbehaving ones.

We observe that the effectiveness of the system is substantially reduced in the presence of misbehaving nodes. However, for $V = 1K$ it is still able to block 54% of the spam when nodes reject connections that they are 75% confident to be spamming.

In Figure 7 we report the number of spam connections per second that are characterized as such by the regular nodes (true positives), throughout the spam campaign. This is the same experiment as the one in Figure 6. Spamming hosts also act as misbehaving FaceTrust nodes reporting 1.0 trust for misbehaving nodes and 0%

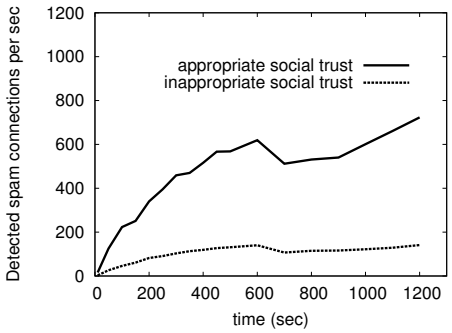

Figure 8: The rate with which spam connections are detected as a function for appropriate and inappropriate social trust values. Connections for which a node is 75% confident that are spam are considered detected by that node.

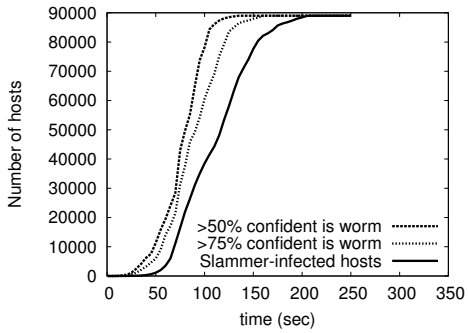

Figure 9: The number of nodes that are 50% or 75% confident of slammer packets being a worm upon encountering them. We also plot the number of nodes that are infected as a function of time in the absence of a containment mechanism.

confidence that spamming nodes are spammers. At each instance, the rate is derived as the number of detected connections over the last 10 seconds. We observe that as time progresses and the trust of the spam bots propagates in the network, the rate with which spam bots successfully establish connections decreases. In addition, as time progresses the trustworthiness of honest reporters increases resulting in nodes trusting their reports faster, resulting in more effectively blocking of spam connections. Furthermore as time progresses, the reputation of dishonest FaceTrust nodes decreases resulting in less effective manipulation of the trust, and enabling more effective spam detection. Persistent bots are the easiest to block as they get blacklisted early in their lifetime. Consequently, after the first 600 sec, as the ephemeral bots die out, the effectiveness of the system in blocking spam is drastically improved.

The above results demonstrate that FaceTrust offers a plausible spam blocking primitive. If it is used in conjunction with existing email classification mechanisms it has the potential for very wide threat coverage.

4.2.2 Significance of Social Trust

With this evaluation, we demonstrate the importance of using the social network of FaceTrust admins to initialize the direct trust between nodes to an appropriate value. The settings for this simulation are the same as for Figure 7, except that all the direct trust values between socially acquainted nodes are set equal to 0.1, and the direct trust is set to 0.9 for only 5 instead of 20 pre-trusted nodes.

Figure 8 illustrates that appropriately initializing the trust graph with proper social trust values yields more effective blocking of spammers than not initializing the trust graph. On average when the social trust is not appropriately used, at the end of the simulation a FaceTrust node has only 0.11 trust towards honest nodes and 0.01 trust towards misbehaving ones. Had regular FaceTrust nodes not leverage social trust, they would be disconnected from the rest of their trust graph and derive 0.0 trust towards their peers. Consequently, they would be unable to consider the behavioral reports of their peers and reject spam connections.

This result motivates our design choice to tap into the social trust between acquainted FaceTrust admins. Social trust is important because it allows new nodes to form a sufficiently connected reporter trust graph, which they can use to derive appropriate transitive trust values towards their peers. These trust values can be derived without prior report verifications. Social trust also contributes in trust values converging to correct ones faster (given that the social trust values are appropriate), even in the case report verifications are infrequent.

4.2.3 Internet Worm Simulation

To further demonstrate FaceTrust's utility, we now evaluate our system's ability to contain epidemically self-replicating malicious code. We simulate the behavior of the Slammer worm. We derive the Slammer epidemic parameters from [36], with the distribution of scan rate among infected hosts being normal with mean 4000 scans/sec, $N(4000, 2000^2)$. The parameter α of Equation 1 is set equal to 0.3, and the parameter ϵ (Section 3.3) is set equal to 0.1. Lastly, the parameter D (Section 3.5) is set equal to 3 sec.

Out of the $100K$ total simulated FaceTrust nodes, $10K$ nodes are able to detect worms with 100% certainty and not get infected (e.g. Vigilante [19] nodes). We refer to these 10K nodes as *honest nodes*. We refer to the rest $90K$ nodes as *regular nodes*. Regular nodes rely on honest nodes to warn them about worm-carrying packets. The rest of the nodes cannot detect worms and do not verify the behavioral reports of other nodes, therefore they do not generate behavioral reports and do not

send direct trust updates. Another $1K$ nodes are misbehaving FaceTrust nodes. These misbehaving nodes claim 0% confidence that slammer packets are worms. They also claim 1.0 direct trust for misbehaving nodes in their view. In addition, misbehaving nodes refuse to store or forward DHT queries regarding behavioral reports that would help regular nodes to contain the worm. Slammer randomly generates target IP addresses to scan, while the IPs FaceTrust nodes are picked uniformly randomly in the 32-bit IP space.

FaceTrust nodes retrieve behavioral reports for the Slammer packet when their IDS application calls `GetTrust(packet signature,‘‘is worm’’)`. They submit behavioral reports as soon as their application classifies a packet as being the slammer worm. They update their direct trust as soon as the node verifies the behavioral reports of another node.

Each node has a $1K$-node view of the FaceTrust network. Depending on the threshold of confidence for deciding whether a packet is worm, nodes that are $> 50\%$ or $> 75\%$ confident that a packet is worm do not get infected. Nodes that are not as confident about the packet being worm get infected upon being scanned and start scanning themselves. The epidemic originates from a single infected host at the start of the simulation. Infected nodes do not act as misbehaving FaceTrust nodes.

Figure 9 shows the number of regular nodes that are $> 50\%$ or $> 75\%$ confident of slammer packets being a worm as time progresses. In addition, it shows the number of hosts that become infected in the absence of a defense mechanism. Our results show that FaceTrust converges fast towards all honest FaceTrust nodes having greater than 50% confidence that the worm packet is malicious. The rate of convergence is faster than the rate of worm propagation. This result implies great potential for containment of self-replicating malicious code.

5 Related Work

We now discuss prior work pertinent to FaceTrust's design.

5.1 Reputation Management Systems

FaceTrust is inspired by prior work on reputation and trust management systems [14, 23, 32, 34]. Well-known trust and reputation management systems include the rating scheme used by the eBay on-line auction site, object reputation systems for P2P file sharing networks [26, 52] or schemes to incent seeding in P2P content distribution systems [12, 39]. Some reputation systems claim resilience to false reports and collusions [16, 20], however the Sybil attack if successful enables an adversary to defeat these defenses. In contrast to these systems, FaceTrust applies to a broad range of entities, ranging from data objects to end systems. second, FaceTrust incorporates social trust to mitigate false reporting and Sybil attacks; other systems typically rely on a central certification authority, e.g. [52].

In EigenTrust [26] nodes converge to the same trust value because a node learns about all the interactions between peers in the system, while in FaceTrust nodes learn only about the interactions involving nodes in their view. FaceTrust aims at enabling nodes to have a reliable estimate of the probability that a peer is trusted and not in system-wide trust convergence. Therefore, FaceTrust is a more lightweight reputation management protocol, which does not require multiple iterations. EigenTrust reputation values are normalized so that the sum of the reputation values for all hosts is 1.0. The rationale is that no peer should be able to disproportionately affect the reputation ranking of a peer by sending a highly skewed reputation update. EigenTrust's normalized reputation values do not map to the probability that a host's report is valid, while FaceTrust trust values in $[0, 1.0]$ do. EigenTrust values can be used for ranking hosts, but cannot be used to derive the probability that a host's reports are trustworthy.

PageRank [15] leverages the link structure of the web to rank the popularity and significance of web search results. The PageRank of a web page corresponds to the probability that a web surfer will eventually visite this site by randomly following links. It is similar in principle to EigenTrust, with the main difference being that trust in PageRank is derived by incoming links, while in EigenTrust by explicit trust assignments between nodes derived by direct observations of each other's behavior.

5.2 Exploiting Social Networks to Derive Trust

Several works exploit the trusted relations that form social networks to reliably assess the trustworthiness of entities [22,24,33,40,43,44,60]. Unlike FaceTrust, they do not use tagged social links both to bootstrap trust values between socially acquainted nodes and to defend against Sybil attacks.

5.3 Collaborative Unwanted Traffic Mitigation

FaceTrust is a generic collaborative threat mitigation framework that can be used to defend against a large variety of attacks against the Internet infrastructure. Vigilante [19] and Sweeper [50] are also collaborative threat mitigation platforms. However, they are purpose-built for worm containment through malicious code detection

and distribution of antibodies. Vigilante employs dynamic taint analysis within an isolated virtual machine sandbox. Sweeper uses lightweight monitoring techniques and lightweight checkpointing to detect and recover from attacks during normal execution. Vigilante requires that a host is equipped with a sandbox in order to verify the antibody-carrying Self Certifying Alerts sent by other Vigilante nodes. In Sweeper, it is assumed that the consumers of antibodies produced by other nodes in the system fully trust the producers. Since FaceTrust employs reliable trust metrics, nodes in our system are able to determine the validity of alerts and antibodies without having to verify them themselves.

In [17] distributed monitors use heuristics to detect worms and report there fingerprints using a DHT. Zou et al. propse a system [61] in which distributed monitors report worm signatures to a centralized server.

Prior work also includes proposals for collaborative spam filtering [1, 58, 59]. Unlike these systems, FaceTrust explicitly addresses the issue of trustworthiness of the collaborating spam reporters through a distributed reporter reputation management system. Kong et al. [28] consider non-compliant behavior, using Eigentrust for reporter reputation. Nodes exchange email signatures,filter email based on content Besides being threat-specific, the aforementioned solutions only enable classifying the contents of emails and not the source of spam. This requires email servers to waste resources on email reception and filtering. FaceTrust can assign trust metrics to sources, thereby rejecting unwanted email traffic on the outset.

TrustedSource [6] employs a centralized repository to which more than 7000 globally distributed submit behavior reports. Applications query the repository which returns the reputation for the specified IPs,domain names or URLs. The reputation can be one of the five trusted, neutral, unverified, suspicious, and untrusted. A similar infrastructure is used by the SpamHaus [7] and Trend Micro [8], which rely on worldwide sensors to report the IP address of spam senders. Based on the sensor's feedback SpamHaus publishes blacklists of IPs known to send spam. Unlike FaceTrust, the reputation for IPs/domains/URLs or the blacklists depend only on the reported values of the sensors/honeypots and not on the trustworthiness of the sensors themselves. These sensors are deployed by TrustedSource, SpamHaus and TrendMicro or are assumed to be trusted. We envision FaceTrust to consist of a much larger number of sensors/reporters as it is going to be a P2P system installed by bot system administrators and casual Internet users within large numbers of distinct trust domains. In addition, centralized reputation services incur a delay between the first reception of report of misbehavior and the time the reputation of the entity is publicly available, offering a rather large window of opportunity to attackers. This delay can be attributed to the fact that these services determine the reputation of an entity once the number of reports exceeds a predetermined threshold in order to avoid incorrect classifications. FaceTrust does not need to incur this delay because it leverages the history of past interactions to readily determine trust values.

Zang et al.'s Highly Predictive Blacklisting [57] compares blacklists submitted by distinct distributed contributors. They rank blacklist reports based on similarity. The intuition is that if two contributors have been attacked by the same attackers in the past they are more likely to be attacked by the same attackers in the future. Thus, reports from a contributor that had similar experiences in the past should weigh more.

Behavioral blacklisting [42] combines spamming activity across multiple spam target domains. It identifies spamming IP addresses based on the observation that spamming botnets tend to send large amounts of emails from many IP addresses, to relatively small number of domains over a short period of time. The system is able to capture this behavior and cluster offending IP addresses based on spam target domains and time frames. FaceTrust can be used to assign trust values to email sender behaviors based on spam content, sending frequency, time patterns etc as reported in [42, 54].

Ostra [35] leverages social networks to tackle unwanted communication and is inherently resilient to the Sybil attack. It bounds the total amount of unwanted communication a user can produce based on the number of trust relationships the user has and the amount of communication that has been flagged as wanted by its receivers. In our setting, Ostra could be used to regulate the amount of behavioral reports that each FaceTrust node can send. However, Ostra assumes that all communication is verifiable and can be flagged as wanted. However, this is not always the case in FaceTrust where nodes do not always have the ability to verify behavioral reports. Thus, instead of limiting the number of behavioral reports a node can produce, we chose to assigns levels of trustworthiness to each report that depend both on the node's reporting history and the social relationships of the node's administrators.

Sybil Attack Defenses The most common strategy against Sybil attacks is perhaps relying on a central authority that establishes a Sybil-free identity domain by binding each entity to a single cryptographic identifier (e.g. public key certificate). Douceur [21] states that this approach is the only one fully capable of preventing Sybil attacks. The challenges of this approach include the cost of deployment, privacy and anonymity issues, cryptographic identifier revocation etc. In addition, this approach relies on the fact that the trusted authority is

fully capable to distinguish unique identities, which may not always be true.

Another common defense against Sybils is resource testing of computing or storage capability. The underlying assumption is that a Sybil attacker does not possess enough resources to perform the additional tests imposed on each Sybil node. Some drawbacks with resource testing are listed in [21], such as the fact that attackers subvert this defense by tricking humans into solving CAPTCHAS [10] posted on their website or presented by malware on infected machines.

FaceTrust's design is inspired by SybilGuard and SybilLimit [55, 56]. These systems defend against Sybil attacks [21] by exploiting the fact that OSNs can be used for resource testing, where the test in question is a Sybil attackers ability to create and sustain acquaintances.

SybilGuard and SybilLimit are decentralized protocols that operate over a distributed social network. Given a request for connection, the protocol decides whether the request is accepted or rejected. SybilLimit restricts a Sybil attacker that managed to socially associate with $O(\sqrt(n)/\log(n)$ nodes in the social network to performing collaborative tasks with only $O(\sqrt(n)$ legitimate network nodes, regardless of how many Sybils it deploys. In SybilGuard/Limit, a verifier either accepts a suspect as a node with which to establish a social edge with or not. However, in real social network large clusters of legitimate social network nodes are frequently not well-connected with other clusters, meaning that nodes from one cluster may reject collaboration requests from legitimate nodes in other clusters. FaceTrust Sybil defenses are designed for centrally maintained online social networks (e.g. Facebook). They are conceptually simpler because they leverage information available to the OSN provider (i.e. the complete social graph topology), and intend to not completely ignore input from not well-connected nodes, but instead assign low weight (identity trust) to it.

A SybilGuard-like technique is used by Leshniewski et al. [30] to limit the Sybil attack in one-hop DHTs.

5.4 Social-network-based Applications

Instant messaging networks such as Google Talk, Yahoo IM, and MSN are prime examples of Internet applications that form online social networks. The research community and the industry has also created social-networking-based applications for file-sharing and backup [9, 31, 49]. FaceTrust extends the utility of social networks to providing a trust layer for Internet entities, aimed at mitigating unwanted traffic.

6 Conclusion

We have presented FaceTrust, a scalable peer-to-peer system for the rapid propagation of reports concerning the behavior of Internet entities (hosts, email signatures, packet fingerprints etc). FaceTrust nodes use each other's reports and the social network of their human users to provide to applications a quantitative measure of an entity's trustworthiness: the likelihood that an entity is associated with a specified malicious action. Applications can in turn use this measure to make informed decisions on how to handle traffic associated with the entity in question.

Our simulation-based evaluation demonstrated our design's potential for the suppression of two common types of unwanted traffic. FaceTrust nodes were able to identify 71% of spam connections with greater than 75% confidence. In addition, FaceTrust nodes became aware of a worm fingerprint at a faster rate than the worm propagated.

References

[1] Cloudmark. `http://www.cloudmark.com/en/home.html`.
[2] Facebook. `http://www.facebook.com`.
[3] Facebook developers. `http://developers.facebook.com/`.
[4] Google Safe Browsing API. `code.google.com/apis/safebrowsing`.
[5] McAfee SiteAdvisor. `http://www.siteadvisor.com`.
[6] Secure Computing, TrustedSource. `http://www.securecomputing.com/index.cfm?skey=1671`.
[7] The spamhaus project. `http://www.spamhaus.org/`.
[8] TrendMicro Email Reputation Services. `http://us.trendmicro.com/us/products/enterprise/network-reputation-services`.
[9] Wuala, the social online storage. `http://wua.la/`.
[10] L. V. Ahn, M. Blum, N. Hopper, and J. Langford. Captcha: Using Hard AI Problems for Security. In *Eurocrypt*, 2003.
[11] R. Albert, H. Jeong, and A.-L. Barabasi. Internet: Diameter of the world-wide web. In *Nature*, 1999.
[12] N. Andrade, M. Mowbray, W. Cirne, and F. Brasileiro. When can an Autonomous Reputation Scheme Discourage Free-riding in a Peer-to-peer System? In *CCGRID*, 2004.
[13] M. Bailey, E. Cooke, F. Jahanian, J. Nazario, and D. Watson. The Internet Motion Sensor: A Distributed Blackhole Monitoring System. 2005.
[14] M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized Trust Management. In *IEEE S&P*. Oakland, CA.
[15] S. Brin and L. Page. The Anatomy of a Large-scale Hypertextual Web Search Engine. 1998.
[16] S. Buchegger and J. L. Boudec. A Robust Reputation System for Mobile ad hoc Networks. 2003.
[17] M. Cai, K. Hwang, Y. Kwok, S. Song, and Y. Chen. Collaborative Internet Worm Containment. In *IEEE Security and Privacy Magazine*, 2005.
[18] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. Wallach. Secure Routing for Structured Peer-to-peer Overlay Networks. In *USENIX OSDI*, 2002.
[19] M. Costa, J. Crowcroft, M. Castro, A. Rowstron, L. Zhou, L. Zhang, and P. Barham. Vigilante: End-to-end Containment of Internet Worms. In *ACM SOSP*, 2005.
[20] C. Dellarocas. Immunizing Online Reputation Reporting Systems Against Unfair Ratings and Discriminatory behavior. In *ACM conference on Electronic commerce*, 2000.
[21] J. R. Douceur. The Sybil Attack. In *IPTPS*, 2002.
[22] S. Garriss, M. Kaminsky, M. Freedman, B. Karp, D. Mazieres, and H. Yu. Re:Reliable Email. In *NSDI*, 2006.
[23] K. Hoffman, D. Zage, and C. Nita-Rotaru. A Survey of Attack and Defense Techniques for Reputation Systems. In *ACM Computing Surveys*, 2008.
[24] T. Hogg and L. Adamic. Enhancing Reputation Mechanisms via Online Social networks. In *ACM conference on Electronic Commerce*, 2004.
[25] J. Kaiser. It's a Small Web After All. In *Science*, 1999.
[26] S. D. Kamvar, M. Schlosser, and H. Garcia-Molina. The EigenTrust Algorithm for Reputation Management in P2P Networks. In *WWW*, 2003.

[27] S. Katti, B. Krishnamurthy, and D. Katabi. Collaborating Against Common Enemies. In *ACM IMC*, 2005.

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

[29] J. Leskovec and C. Faloutsos. Sampling from large graphs. In *ACM SIGKDD*, 2006.

[30] C. Lesniewski-Laas. A Sybil-proof One-hop DHT. In *Workshop on Social Network Systems*, 2008.

[31] J. Li and F. Dabek. F2F: Reliable Storage in Open Networks. In *IPTPS*, 2006.

[32] J. Li, N. Li, X. Wang, and T. Yu. Denial of Service Attacks and Defenses in Decentralized Trust Management. In *ACM CCS*, 2006.

[33] J. Li, Y. Sovran, and A. Libonati. Pass it on: Social Networks Stymie Censors. In *IPTPS*, 2008.

[34] S. Marti and H. Garcia-Molina. Taxonomy of Trust: Categorizing P2P Reputation Systems. 2006.

[35] A. Mislove, A. Post, P. Druschel, and K. P. Gummadi. Ostra: Leveraging Social Networks to Thwart Unwanted Traffic. In *USENIX NSDI*, 2008.

[36] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver. Inside the slammer worm. In *IEEE S&P*, 2003.

[37] D. Moore, C. Shannon, G. Voelker, and S. Savage. Internet Quarantine: Requirements for Containing self-propagating Code. In *IEEE INFOCOM*, 2003.

[38] K. Mller and T. Vignaux. Simpy (simulation in python). `http://simpy.sourceforge.net/`.

[39] M. Piatek, T. Isdal, A. Krishnamurthy, and T. Anderson.

[40] J. M. Pujol and R. S. J. Delgado. Extracting Reputation in Multi Agent Systems by Means of Social Network Topology. In *International Conference on Autonomous agents and Multiagent systems*, 2002.

[41] A. Ramachandran and N. Feamster. Understanding the Network-level Behavior of Spammers. In *ACM SIGCOMM*, 2006.

[42] A. Ramachandran, N. Feamster, and S. Vempala. Filtering spam with Behavioral Blacklisting. In *ACM CCS*, 2007.

[43] J. Sabater and C. Sierra. Reputation and Social Network Analysis in Multiagent Systems. In *International Conference on Autonomous agents and Multiagent systems*, 2002.

[44] J. Sabater and C. Sierra. Social Regret, a Reputation Model Based on Social Relations. 2002.

[45] S. Singh, C. Estan, G. Varghese, and S. Savage. Automated worm fingerprinting. In *USENIX OSDI*, 2004.

[46] E. Sit and R. Morris. Security Considerations for Peer-to-peer Distributed Hash Tables. In *IPTPS*, 2002.

[47] L. Spitzner. The Honeynet Project: Trapping the Hackers. In *Security & Privacy Magazine, IEEE*, 2003.

[48] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service. In *ACM SIGCOMM*, 2001.

[49] D. N. Tran, F. Chiang, and J. Li. Friendstore: Cooperative Online Backup Using Trusted Nodes. In *SocialNets*, 2008.

[50] J. Tucek, S. Lu, C. Huang, S. Xanthos, Y. Zhou, J. Newsome, D. Brumley, and D. Song. Sweeper: a Lightweight End-to-end System for Defending Against Fast Worms. In *EuroSys*, 2007.

[51] M. Vrable, J. Ma, J. Chen, D. Moore, E. Vandekieft, A. C. Snoeren, G. M. Voelker, and S. Savage. Scalability, Fidelity, and Containment in the Potemkin Virtual Honeyfarm. In *ACM SOSP*, 2005.

[52] K. Walsh and E. G. Sirer. Experience with an Object Reputation System for Peer-to-Peer Filesharing. In *USENIX NSDI*, 2006.

[53] D. J. Watts and S. H. Strogatz. Collective dynamics of 'small-world' networks. In *Nature*, 1998.

[54] Y. Xie, F. Yu, K. Achan, R. Panigrahy, G. Hulten, and I. Osipko. Spamming Botnets: Signatures and Characteristics. In *ACM SIGCOMM*, 2008.

[55] H. Yu, P. Gibbons, M. Kaminsky, and F. Xiao. A near-optimal social network defense against sybil attacks. In *IEEE S&P*, 2008.

[56] H. Yu, M. Kaminsky, P. B. Gibbons, and A. Flaxman. SybilGuard: Defending Against Sybil Attacks via Social Networks. In *ACM SIGCOMM*, 2006.

[57] J. Zhang, P. Porras, and J. Ullrich. Highly Predictive Blacklists. In *USENIX Security*, 2008.

[58] Z. Zhong, L. Ramaswamy, and K. Li. ALPACAS: A Large-scale Privacy-Aware Collaborative Anti-spam System. In *IEEE INFOCOM*, 2008.

[59] F. Zhou, L. Zhuang, B. Zhao, L. Huang, A. Joseph, and J. Kubiatowicz. Approximate Object Location and Spam Filtering on Peer-to-Peer Systems. In *ACM/IFIP/USENIX Middleware*, 2003.

[60] P. Zimmerman. PGP Users Guide. December 1992.

[61] C. Zou, L. Gao, W. Gong, and D. Towsley. Monitoring and Early Warning for Internet Worms. In *ACM CCS*, 2003.