Friend-to-friend data sharing with OneSwarm

Tomas Isdal* Michael Piatek* Arvind Krishnamurthy* Thomas Anderson*

Abstract

Peer-to-peer data distribution is a useful primitive for building scalable network services. But, the public sharing typical of currently popular P2P networks exposes user behavior to systematic monitoring, raising privacy concerns that restrict use.

This paper presents the design and implementation of OneSwarm, a P2P data sharing protocol that uses strong identities and friend-to-friend data transfer to inhibit systematic monitoring of its users. To guide our design, we report measurements of the social network and sharing workload from a popular music web service with more than one million users. Trace playback and measurements of real-world deployments confirm the practicality of our approach. OneSwarm effectively controls overlay overhead, stress, and stretch while restricting sharing to a social graph.

1 Introduction

Peer-to-peer (P2P) networks are among the most popular data distribution systems on the Internet today. Well-known file sharing systems such as BitTorrent have millions of users transferring terabytes of data in aggregate. For content producers, P2P networks provide a way to externalize the storage and distribution costs of building Internet scale services. For users, P2P networks provide scalable data delivery even during flash crowd scenarios.

Although popular, P2P networks have a significant drawback; indiscriminate data sharing exposes user behavior to systematic monitoring by third parties. Unlike traditional communication networks that require legal oversight for monitoring, P2P network use can be recorded without the permission of anyone. For example, a BitTorrent tracker by design returns a random subset of the users potentially interested in downloading a particular file; by posing as a user, a third party can crawl this data to identify anyone's interest set [24]. As a result, P2P networks can only be safely be used by those comfortable with wholly public knowledge of their activity. And even then, the interfaces are often vulnerable to spoofing, allowing a miscreant to plant evidence to implicate completely innocent non-users [25].

One partial solution would be to launder P2P traffic through an anonymizing multihop overlay such as Tor [10]. However, this faces several practical problems. First, in an anonymizing overlay, the public activity is attributable to the overlay exit node, creating a severe disincentive to host a node. As a result, such networks are woefully underprovisioned compared to the amount of data transmitted over P2P networks; P2P file sharing is strongly discouraged over Tor, for example. Secondly, because of the need for multiple layers of encryption, and non-geographic routing for security, the performance of such a system would be very poor compared to normal P2P network used for file sharing, for example. And finally, the privacy of such a solution is uncertain; since anyone can become a Tor node, the network is vulnerable to attacks by well-provisioned monitoring agents [19, 31].

In this paper, we present a P2P data sharing protocol that leverages social relationships to give each user explicit control over their privacy. Data can be made public, it can be shared with specific friends for sharing with their friends, shared with family but not friends, and so forth. We call this friend-to-friend (F2F) data sharing, and we have built this into a new P2P client called OneSwarm. In essence, we build several of the trappings of an operating system—persistent identities, access control, and resource sharing—into a P2P data sharing system. Our hypothesis is that we can significantly improve user privacy without significantly impairing robustness or performance.

To guide our design, we have measured and analyzed the social network and usage behavior of more than 1 million users of the last.fm music website. What is unique about our data set is that it allows us to study both social relationships and per-user sharing at the same time; previous characterizations of social networks have examined graph structure alone, while previous studies of file sharing omitted social relationships. To preview a few of our findings, surprisingly the last.fm social graph does not have a highly connected core group of users, and the object popularity distribution of last.fm is more skewed than that of BitTorrent. We plan to make an anonymized version of our data available on request.

The remainder of this paper is organized as follows. Section 2 provides an overview of our measured data of user behavior and relationships in the last.fm community. This data motivates the design of OneSwarm, described in Section 3. We evaluate OneSwarm in Section 4 using trace replay and measurements of real deployments. In Section 5, we discuss related work and conclude in Section 6.

*Dept. of Computer Science and Engineering, Univ. of Washington

2 Measuring a social sharing workload

Currently, systems builders have limited data with which to evaluate new protocol designs layered on top of social networks. Without measurements of real social graphs and associated workloads, the relevant constraints and properties of the environment are unclear. Recently, significant progress has been made towards measuring and understanding the properties of online social networks [3, 20, 21]. But, these existing studies have focused primarily on graph properties. While essential, graph properties alone are not sufficient. To design and evaluate file sharing protocols built on top of a social graph, we need to understand the sharing behavior of users as well.

In this section, we report measurements of both a social network and a sharing workload. The popular music website last.fm[1] provides both. last.fm builds music related services on top of a database of user listening habits and a social network. These services include artist recommendations, Internet radio, popularity charts, interest groups, etc. To build the collective database, last.fm users run custom software that reports the playback history of each user's local music library.

Maintaining a list of friends is not required to use any of last.fm's core functionality. The social network is used to provide communication, notifications, and convenient access to the listening histories of friends. All friend links are symmetric and must be approved by both parties. Also, each user's list of friends and listening history is public.

2.1 Methodology

To measure both the social graph and each user's listening behavior, we used last.fm's public XML-RPC API.

Crawling the social graph: Given a user name, the last.fm API provides a list of that user's friends. We crawled the social graph using a breath first search starting from several manually chosen seed users. Because last.fm acceptable use policy imposes a rate limit on API calls, this process took one month to complete (August 25th–September 25th, 2008).

Crawling the sharing workload: The last.fm API provides a list of song identifiers and play counts for users at a per-week granularity. Along with social links, we collected a trace of these week-long histories for each crawled user for the two weeks prior to the start of our crawl. Although these week-long histories provide a coarse-grained view of usage per user, they do not translate directly to fine-grained trace replay, i.e., at the granularity of minutes. last.fm provides fine-grained information with song identifiers and timestamps as an RSS feed for each user, but monitoring these RSS feeds for all last.fm users exceeds crawl rate-limit. Instead, we sampled 1,000 random users from our crawl and our analysis of short-term activity uses this trace.

[1]http://www.last.fm/

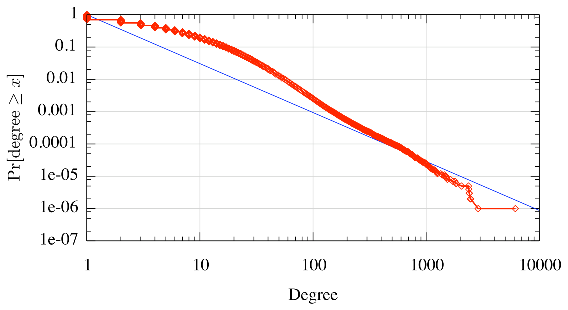


Figure 1: CCDF of user degrees in the last.fm trace. A best-fit power law distribution is shown ($\alpha = 1.51$).

2.2 Social network

Estimating coverage: Our crawl discovered 1,768,197 users and 6,325,306 social links. Most users that had social links were in a single large connected component. Because last.fm does not provide a count of all active users, we estimate coverage by sampling users and computing the fraction of these that were observed during our crawl. last.fm provides lists of users per country, and our samples were drawn randomly from the set of all users providing country information.[2] We sampled 8,081 such users of which 4,263 occur in our crawl (53%). Of the remaining users, 92% have no social links. The remaining 8% of users are grouped into small, disconnected clusters. These results suggest that our crawl covers the largest connected component in the social network and that the overwhelming majority of remaining users have no social links.

Path properties: The average path length in the last.fm social graph is 7.1, and the diameter is 14.[3] Paths between last.fm users tend be longer than those of other social networks, e.g., Mislove, et al. report average path lengths between 4 and 6 for popular social networks [21]. We attribute this difference to two factors. First, unlike services which provide a social network only, e.g., Orkut, social networking is not the primary purpose of last.fm. Second, the degree distribution of crawled users in last.fm includes a comparatively high number of low degree nodes. We examine this next.

Degree distribution: Figure 1 shows the complementary cumulative distribution function (CCDF) of degrees for all users observed in our trace. Our crawl reveals that the majority of users have very low degree. 30% of users

[2]Although we could *sample* such users by screen scraping web pages, *enumerating* all users in this manner violates the API acceptable use policy.

[3]Because computing all shortest paths in such a large graph is not computationally feasible, we consider a sample of 50,000 randomly selected user pairs.

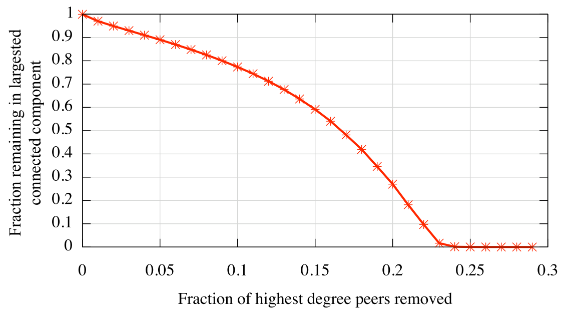


Figure 2: The fraction of nodes in the largest connected component of the last.fm social graph (y-axis) as an increasing fraction of high degree nodes are removed (x-axis).

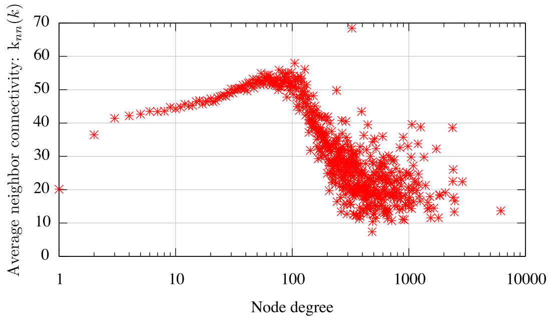


Figure 3: The average degree of directly connected neighbors (k_{nn}, y-axis) for nodes with increasing degree (x-axis). High degree nodes tend to connect to many nodes of low degree.

have just one social link, the median degree is 3, and 81% of users have 10 or fewer friends. The majority of low degree nodes inflate average path length since reaching the majority of fringe users requires an extra hop. Also shown is a best-fit power law distribution ($\alpha = 1.51$) obtained using the maximum likelihood method [8]. The Kolmogorov-Smirnov goodness-of-fit metric for the fit is 0.137. Unlike other networks, the last.fm degree distribution does not strongly follow a power-law.

Highly connected core: Social networks with power-law degree distributions tend to have a highly connected core of nodes whose removal quickly partitions the graph. For protocols built on social networks, this may hinder both performance and robustness. When available, core nodes may become bottlenecks. When unavailable, path lengths increase, raising overhead and reducing capacity.

Unlike several previously studied social networks, last.fm does not appear to have a strongly connected core. To analyze this, we remove an increasing fraction of the highest degree nodes from the graph and compute the resulting connectivity. The results are summarized in Figure 2. Connectivity degrades slowly, suggesting the existence of redundant paths among highly connected nodes. An identical prior analysis shows that for other online social networks, the largest connected components fractures after the removal of just 1-10% of high degree nodes [21]. In contrast, breaking up the largest connected component of the last.fm social graph requires removing 24% of high degree nodes. After these removals, the largest connected component contained fewer than 1% of remaining nodes.

Additional structural properties of the social graph are revealed by the degree correlation function, k_{nn} [18]. This measures the average degree of directly connected neighbors as a function of node degree and is computed using the joint degree distribution, i.e., the probability of connecting to a node with a particular degree conditioned on having a particular degree. The degree correlation function provides a measure of how often high degree nodes tend to connect to other high degree nodes, a property typical of power-law graphs with a highly connected core.

The degree correlation function for the last.fm graph is shown in Figure 3. In networks where high degree nodes tend to connect to other high degree nodes, k_{nn} increases with degree. For last.fm, we observe the opposite. Many relatively low-degree nodes are directly connected to nodes with high degree. This is consistent with our analysis of the robustness of the social graph (Figure 2). While the largest connected component did not fracture suddenly as high degree nodes were removed, the low degree nodes that are reachable through only a single high degree node are disconnected, leading to gradual decline in the size of the largest connected component.

Synthesizing these results, we observe that limited path redundancy is expected for those users with extremely low degree. But, for the set of nodes with even modest connectivity, redundant paths exist, even after targeted removal of high degree nodes.

From the perspective of building OneSwarm, our analysis of the last.fm social graph points to promising structural properties but potentially large overhead. Robust connectivity even after targeted removals suggests the potential for routing around capacity constrained nodes or those that are unavailable. But, large average path length may require significant control traffic to discover data sources and lengthy overlay forwarding paths, hurting performance. Of course, we cannot separate the design and evaluation of a file sharing protocol built on a social graph from the properties of that graph. As a result, our quantitative results are specific to the workload we measured. It remains an open question whether these results will generalize to the different social networks that might arise in practice.

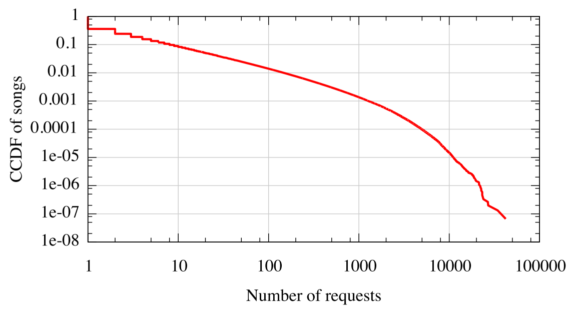


Figure 4: CCDF of the number of unique users listening to a given song for all observed songs. 64% of songs are listened to just once.

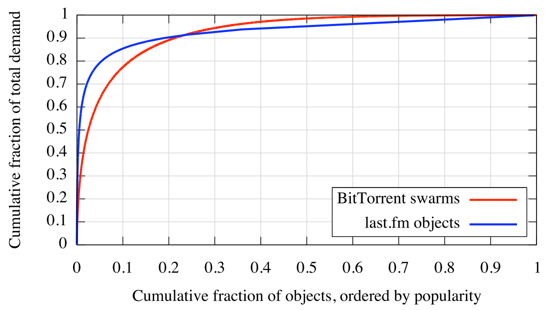


Figure 5: Distributions of demand in the last.fm and BitTorrent workloads.

2.3 Listening habits

This section reports measurements of the listening behavior of last.fm users. We focus on the workload properties most relevant to the design of OneSwarm. These are: 1) the popularity of objects, 2) the variation in demand among users, and 3) the total and peak demand. We discuss each of these in turn.

Object popularity: For file sharing systems layered on social networks, path lengths depend on both the connectivity of users and the object popularity. Even if paths between users are typically lengthy, paths to popular objects may be short because of high replication. We first consider object popularity in terms of requests per object. This is shown in Figure 4. Most objects receive few requests from unique users; 64% of songs are listened to by just one user.

Although the majority of objects are unpopular, popular objects account for the majority of total demand. Figure 5 shows the cumulative fraction of total system demand attributed to objects ordered by decreasing popularity. We reproduce an identical accounting for demand in the BitTorrent P2P file sharing system for comparison [26]. Demand is skewed in both BitTorrent and last.fm but both the heads and tails of the distributions differ. Unpopular objects contribute significantly more to total demand in last.fm than in BitTorrent. Songs listened to by three or fewer unique users account for 10% of total demand. Also, popular last.fm objects account for a larger fraction of total demand than do popular BitTorrent objects. The top 5% of objects account for 79% of total demand in last.fm and 63% in BitTorrent.

The comparatively large fraction of total demand attributable to unpopular objects may stem from last.fm's approach to data collection. Existing P2P workload measurements are influenced by the properties of the distribution system. For example, if unpopular objects have poor availability in a particular P2P network, an object request trace is likely to underrepresent the true demand for those objects. Since last.fm simply records user behavior when interacting with their own libraries, it does not exhibit this bias.

For the design of OneSwarm, popular objects reduce average path lengths since requests will be for objects with many replicas. But, to discover unpopular objects, lengthy paths will be required.

Demand per user: Figures 4 and 5 show demand from the perspective of objects. We next turn to demand per user. For last.fm, demand per user is the distribution of songs played, shown in Figure 6. Demand varies by orders of magnitude; some user histories include 10s of songs while others include 1000s. While this type of skew in demand is typical of object request workloads, one might expect heavy users of last.fm to also have many friends, but the length of play histories and number of friends are weakly correlated ($\rho = 0.14$). From the perspective of file sharing, this implies that a significant fraction of requests will come from users with only limited connectivity.

The measurements of usage in Figure 6 describe only active users, i.e., those that listen to at least one song. Surprisingly, these users are in the minority; 52% of measured last.fm users did not listen to any songs during our two week trace. If availability correlates with activity, protocol designers building on social networks should expect a large fraction of the social links to be unavailable even over lengthy time scales. Over shorter time scales, the last.fm usage exhibits a typical diurnal pattern with peak activity of 7.3% of users and a typical daily minimum of 2%, obtained using our fine-grained measurements of the listening behavior of 1,000 users.

Total demand: Over the two weeks of our activity trace, we observed 799,953 users that listened to at least one song with 156,295,286 total songs played. Of these, 15,120,192 were unique song requests per user (i.e., allowing duplicates only for the logs of distinct users). Multiplying this value by the the average song length in bits (weighted by popularity) gives an estimate for the total demand. Assuming an audio bitrate of 128 kilobits, total demand for measured last.fm users over two weeks

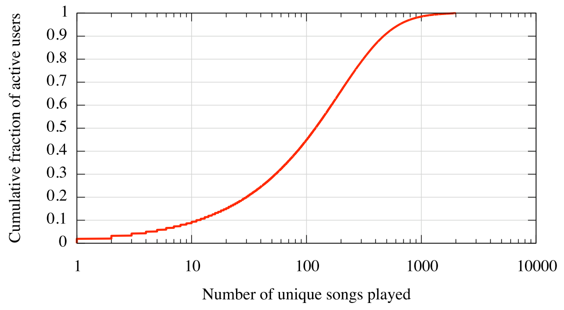


Figure 6: The cumulative fraction of users (y-axis) playing a given number of unique songs (x-axis) or fewer in our two week trace.

is 44.6 TB.

Our measurements suggest that, at least for a music sharing workload, multihop overlay forwarding is practical given current broadband capacities. Distributing 44.6 TB in two weeks requires just 4.2 MB of data per user per day. Even when forwarded over multiple hops, this meager amount of traffic is still well under the gigabytes of total capacity of even a modest 1 Mbit home broadband connection. Further, because our trace accounts for only two weeks of usage, we overestimate the steady-state demand of the last.fm workload where repeated object requests are frequent. This is supported by our data; the number of unique songs added by the second week of our trace was roughly half the unique songs discovered during the first week.

3 F2F sharing in OneSwarm

For the last.fm workload, our measurements show that existing P2P networks are likely to have far more aggregate capacity than aggregate demands. In this section, we describe OneSwarm, a file sharing protocol puts that puts this excess capacity to productive use by improving the privacy of P2P file sharing users. Instead of sharing all data publicly with ephemeral peers, OneSwarm users define permissions for data sharing among trusted, verifiable friends. Instead of centralizing information about which peers have which data objects, e.g., at a coordinating tracker as in BitTorrent, OneSwarm peers locate distant data sources by flooding object lookups through the social overlay. Instead of sources sending data directly to receivers, data transfers occur over the reverse search path in the overlay, using redundant transfers with address rewriting to obscure sender and receiver identity.

In the remainder of this section, we describe the protocol details of OneSwarm, separating our discussion into four parts: 1) how users name and find one another when identities are known, 2) how users learn the identities of their friends, 3) how users discover data sources in the overlay, and 4) how data is exchanged.

3.1 Naming and finding friends

Each OneSwarm user is named using a cryptographic key that persistently identifies that user among friends and bootstraps secure communication. Associated with this key is short-term connectivity information, i.e., an IP address and port. In OneSwarm, each user's persistent identity is the public key of a 1024 bit public/private RSA key pair that is generated when installing the OneSwarm client. This long-term identity is linked to short-term connectivity information via a distributed hash table (DHT) maintained among all OneSwarm peers. On startup, each client P inserts a copy of its current IP address and port into the DHT. This value is inserted multiple times—once for each friend. Each insert is signed by P, encrypted with the public key of a given friend, and indexed by the XOR of P's public key and the public key of the given friend. In other words, each OneSwarm client P maintains the following DHT mappings:

$$\forall F \in \text{Friends}_P, \text{Pub}_P \oplus \text{Pub}_F \rightarrow \{M_{\text{Pub}_F}, \text{H}(M)_{\text{Priv}_P}\}$$

where $M = \{\text{IP}_P, \text{Port}_P, \text{Timestamp}\}$.

Multiple insertions of connectivity information enable fine-grained control over sensitive network address information. A simple alternative is indexing connectivity information with the public key of P alone. But, in this case, any user that learned P's public key could monitor P's network location and availability as long as P maintained its identity. By encrypting updates and updating connectivity information for each friend individually, P can control visibility per-friend.

In our implementation, ID $\rightarrow$ {IP, Port} mappings are stored in a Kademlia-based DHT using twenty-fold replication for fault tolerance. This level of replication has been shown to provide high availability for DHTs running on end-hosts [12]. The DHT identifier of OneSwarm users is independent of their identity in the F2F overlay and is determined on startup by hashing the client's current IP address and DHT port. This inhibits systematic monitoring of targeted regions of the DHT key space since the region for which each client is responsible is determined by that client's network address and port. This is certified during DHT operations by other OneSwarm peers.

3.2 Bootstrapping the social overlay

The OneSwarm DHT tells a client how to connect to a given friend provided that friend's public key is known. But, this requires users to first obtain keys for their friends. In existing F2F designs, e.g., Freenet [7], key exchange is manual. We view this manual exchange as a hinderance to the adoption of protocols based on social networks. OneSwarm makes key maintenance among friends completely automatic by piggy-backing on popular existing social networks, e.g., Facebook and Google Talk. We observe that the explicit encoding of trust relationships, a longstanding stumbling block for public key infrastructures, has already been done by the users of existing social networks; we simply need to tweak the usage of these networks to distribute public keys in addition to photos, messages, etc. In this section, we describe how OneSwarm is bootstrapped using two existing networks, Facebook and Google Talk, although the techniques described apply generally.

Google Talk: Google Talk is an instant messaging client. Users add friends to their list of contacts to receive updates regarding their status, and friend additions are not finalized until approved by both parties. The set of contact lists of Google Talk users forms a social network among users distinguished by Google IDs.

To piggy-back on this network, OneSwarm clients optionally connect to the Google Talk on startup and periodically during runtime to exchange key information with a pool of ten Google Talk bots running at the University of Washington. Each client sends a bot their public key, a nickname, and the SHA-1 hash of the Google Talk IDs of their friends. For each friend hash, the bot checks an internal database to determine whether another OneSwarm user has registered a public key corresponding to that hash. If so, the bot verifies that the relationship is symmetric, i.e., both have registered the hashes of one another, and then returns the public key of the remote peer to the user registering their friend list. During the remote peer's next update, the situation is reversed, and both peers will have learned the other's key from the bot.

Users can choose whether to choose to accept new and updated keys received from the bot manually or automatically. Manual updating allows users to maintain separate lists of OneSwarm and Google Talk friends while still avoiding the inconvenience of manually exchanging keys with friends out-of-band.

Facebook: Facebook is a well known social networking website that provides messaging, photo sharing, interest groups, and other services. Most important for bootstrapping OneSwarm is the public Facebook API, which allows third party applications to query Facebook's database on behalf of users via the Facebook query language, which uses a restricted, SQL like syntax.

A OneSwarm client shares its public key through Facebook by piggy-backing on the photo sharing service provided by the API. On its initial connection, OneSwarm uploads a small image to a newly created 'OneSwarm' photo album with a caption containing a well-known prefix and a text encoded version of its public key. This album is made viewable by directly connected Facebook friends. OneSwarm then searches the user's list of Facebook friends for those who are also sharing a OneSwarm album and imports public keys from properly formatted captions. As with Google Talk, periodic updates synchronizes keys over time and users can approve imported keys manually or automatically.

3.3 Naming and locating data

At this point, we have described how OneSwarm peers join and maintain connections in the social graph and updated connectivity information. We next turn to the wire-level protocol used to connect to friends and search for data. OneSwarm peers connect to one another using secure sockets (SSLv3) bootstrapped by the RSA key pairs each client maintains as its identity. When two friends connect, they exchange file list messages. file list messages are compressed XML including attributes describing the name, size, date shared, and other meta-data for files for which a particular friend has permissions. After the initial file list is received, subsequent lists are simple diffs determined by the timestamp of the previously received file list.

Shared files (or groups of files) are named in OneSwarm using the SHA-1 hash of their name and content. Users obtain content hashes 1) out-of-band, e.g., from an email or website, 2) from file list messages exchanged with friends, or 3) from keyword search the F2F network. We describe file transfer setup via network search since this subsumes the other cases. OneSwarm propagates searches via flooding.

Keyword search messages include a randomly generated search ID, timestamp, and list of keywords. Unlike flooding search in other P2P file sharing networks, OneSwarm search messages do not include a time-to-live. Rather than controlling search overhead by explicitly limiting propagation, OneSwarm forwards searches so long as 1) the forwarder has idle capacity, 2) the search timestamp is less than one minute old,[4] and 3) the search has not been forwarded previously. Clients maintain a time-limited history of previous search messages to avoid forwarding duplicates. To avoid the propagation of every search to every client in the overlay, each client waits 200 milliseconds before forwarding search messages to friends. The search source (or any forwarder) may terminate popular searches for which many data sources have already been discovered by sending a search cancel message to nodes to which they have sent or forwarded a search message. This message is also forwarded along the same paths as the corresponding search message but without any forwarding delay, allowing cancel messages to quickly reach the search frontier.

If a node is sharing a file that matches a search query,

[4]Synchronized clocks are not required. The forwarder's view of the current time is authoritative, and clients do not forward search messages with timestamps in the future.

it does not forward the search and instead responds with a search reply message. search reply messages include a search identifier, a list of content hashes which identify matching files, file metadata, and a path identifier for the reverse path. The path identifier allows clients to name multiple paths even if those paths partially overlap. We describe how the path identifier is used to enable multi-path and multi-source downloading next, but delay this to first describe how it is computed. Each OneSwarm peer maintains a randomly chosen *link ID* for each friend link.[5] The data source sets the initial value of the path ID to the lower 32 bits of the first matching file's hash. Next, the path ID of the search reply is updated before sending the message to each friend by computing the SHA-1 hash of the initial value XOR'd with the link ID of the given friend. This process of updating the path ID is repeated at each overlay hop, resulting in a unique ID for each path that a search reply message traverses on its way back to the sender. A simple example of path ID computation is shown in Figure 7.

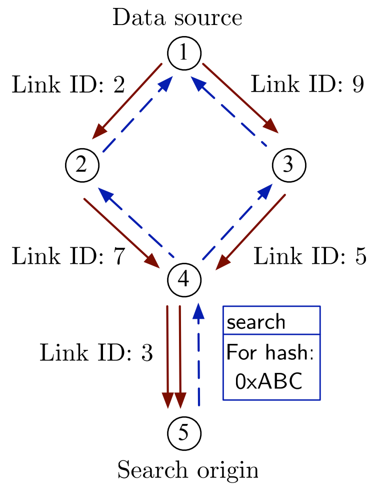
Non-keyword search messages, i.e., those specifying an exact file hash, are handled analogously, and are conducted periodically during data exchange to discover additional replicas for popular objects and to update or discover new paths to existing replicas.

3.4 Data transfer

A path identifier indexes routing tables at each overlay hop and effectively identifies a circuit from data source to receiver. Keep-alive messages refresh paths, which expire after thirty seconds of inactivity. OneSwarm uses the wire-level protocol from the BitTorrent file sharing system to transfer data, first obtaining a list of block hashes corresponding to the metadata stored in .torrent files [9]. But, rather than connecting directly to peers, OneSwarm tunnels BitTorrent traffic through F2F overlay paths. Rather than obtaining a list of peers from a centralized tracker, as in BitTorrent, OneSwarm discovers new peers and paths by periodically flooding search messages for objects being transferred.

Basing OneSwarm's wire-level protocol on BitTorrent draws on BitTorrent's strengths. Swarming file downloads minimize redundant data transfers in the overlay. If multiple users are downloading a popular file, OneSwarm will discover and use paths to those new, partial data sources. Tit-for-tat, BitTorrent's default request servicing policy, serves a second purpose in OneSwarm: load balancing among multiple overlay paths. Like unpredictable and heterogeneous end-hosts, multi-hop overlay paths have highly variable bandwidth and end-to-end latency. Scheduling block requests over unpredictable paths requires careful engineering to avoid wasting capacity or inducing lengthy data queues, but we inherit this feature for free by basing OneSwarm on the popular, widely used Azureus BitTorrent implementation.[6] Also, building OneSwarm on an existing P2P network and popular client helps in bootstrapping the F2F network; even if the social features are rarely used, OneSwarm can still serve as a vanilla BitTorrent client, and publicly shared files that have been downloaded by an F2F user can be shared with friends, bootstrapping content in the overlay.

Unlike BitTorrent, however, OneSwarm benefits from the explicit resource sharing enabled by persistent identities. OneSwarm's default policy is fair queueing among friends, although priority can be assigned on a per friend and/or per transfer basis. Fair queuing of capacity applies to overlay messages, e.g., forwarding search requests and



$$\text{Path ID}_{1\to2} = \text{SHA-1}(\texttt{0xABC} \oplus 2)$$
$$\text{Path ID}_{1\to3} = \text{SHA-1}(\texttt{0xABC} \oplus 9)$$
$$\text{Path ID}_{2\to4} = \text{SHA-1}(\text{Path ID}_{1\to2} \oplus 7)$$
$$\text{Path ID}_{3\to4} = \text{SHA-1}(\text{Path ID}_{1\to3} \oplus 5)$$
$$\text{Path ID}_{4\to5} = \text{SHA-1}(\text{Path ID}_{2\to4} \oplus 3)$$
$$\text{Path ID}_{4\to5} = \text{SHA-1}(\text{Path ID}_{3\to4} \oplus 3)$$

Figure 7: An example of end-to-end path ID computation. Client 5 searches for peers with file ID 0xABC and queries are forwarded along blue, dashed social links. Client 1 has the data and forwards search reply messages along the reverse paths (red, solid lines). Path ID computations are shown for each hop. These vary per friend depending on link ID values chosen at random for each friend link by each client. Client 4 forwards both search reply messages to client 5 since each incoming reply has a different path ID representing a distinct end-to-end path.

[5]Though randomly chosen, this value is fixed for the lifetime of the link.

[6]http://azureus.sourceforge.net

replies, as well as data traffic, limiting the ability of a malicious user to disrupt the overlay by flooding useless control messages. Also, several well-known attacks on P2P resource sharing do not apply in the F2F environment. In the Sybil attack [11], for example, the multiple identities assumed by an attacker to acquire a disproportionate share of resources are unlikely to be approved by would-be friends. Connecting to a large number of peers to receive a disproportionate share of bootstrapping data, the so-called large-view exploit [30], is similarly frustrated.

3.5 Threat model

OneSwarm aims to be resistant to the systematic monitoring that is made trivial by today's public P2P network structures. We point out, however, that an explicit non-goal is to provide provably strong anonymity or to eliminate the possibility of monitoring by a highly capable monitoring agent, e.g., governments or law enforcement. In this section, we specify our threat model, considering several potential attacks on OneSwarm's implementation of F2F sharing.

With respect to privacy, our fundamental assumption is that of trusted friends. Several attacks we consider can be launched by a malicious user against their directly connected friends, but these attacks do not definitively expose the activity of indirectly connected users. With respect to attackers, we assume incomplete knowledge of the social graph and a limited perspective of the raw network traffic among peers in the overlay (with a complete view, the social graph could be inferred). We also assume computational limits that make brute force attacks against against cryptographic primitives infeasible.

- *Inferring social links:* Reasoning about about the actions of a targeted OneSwarm user becomes much easier if an attacker can learn about the complete set of that user's friends. Given our use of existing social networks to bootstrap OneSwarm social links, an attacker that obtained access to either a user's Google Talk contact list or Facebook account could learn about many potential OneSwarm friends. But, the potential for manual addition of friends hampers definitive reasoning using these sources alone. A determined attacker A can test if two users P and Q are friends as follows. Suppose A is friends with both P and Q (either by accidental addition or compromising an existing friend's machine). A can send P a search message and measure the time before receiving the forwarded search from Q. If this time is roughly 400 milliseconds (twice the search forwarding delay), P and Q are likely to be directly connected. An analogous timing attack can be conducted with two colluders: one friend of P and one friend of Q comparing message receive times.
- *Inferring the source of searches:* We consider two ways for a determined attacker to follow reverse paths back to the source of a search query. The order in which search messages are received from the friends of an attack node implies a likely next hop which is closer to the source node. If an attacker can gain access to successive hosts on the reverse path (e.g., law enforcement), the original sender can likely be eventually localized. The likelihood of a particular previous hop can be similarly estimated by counting the number of search reply messages routed to a particular friend; the number of search reply messages increases with proximity to the search source.
- *Inferring the source of data:* A timing attack similar to that described for iteratively localizing search sources applies to data sources as well (by measuring search reply receive times rather than those of search messages).
- *Disrupting network operations:* Persistent identities and content based naming greatly reduce the effectiveness of standard network disruption attacks. As described in Section 3.4, search flooding and several other attacks designed to overwhelm the overlay with control traffic are mitigated by fair queuing of traffic among friends. Spoofing file data is prevented by checking block hashes, which are themselves verified by the file hash (which includes per-block hashes and other metadata in its computation).

Although these attacks may enable very capable monitoring agents to deduce the activities of a targeted OneSwarm user, significant resources are required, e.g., monitoring network traffic and/or compromising the machines of multiple overlay hops. Compared to the relative ease of systematically monitoring P2P networks today, OneSwarm provides users with substantial gains in privacy.

4 Evaluation

Although the OneSwarm protocol is designed to protect users from systematic monitoring, its performance and robustness strongly depend on the properties of the social graph and sharing workload. In this section, we describe two evaluations of OneSwarm. First, we present the results of trace replay using our last.fm measurements. These demonstrate that OneSwarm can meet the demands of a real-world sharing workload on a large-scale social graph. Next, we report our experience with an ongoing, small-scale deployment of OneSwarm among 21 of our own friends (and friends of friends). Over the five weeks that we recorded usage, each user transferred an average of 800 megabytes of data apiece.

4.1 Trace replay in the last.fm social graph

To evaluate OneSwarm's operation at scale, we use the last.fm trace data described in Section 2 to drive a large-scale simulation of F2F file sharing. Our main result is that F2F sharing is feasible for the measured last.fm workload; 94% of requests that can be fulfilled by the overlay (i.e., a replica exists) result in successful downloads, and the majority of transfers use the shortest paths available. In support of this main conclusion, we provide additional analysis of the details of OneSwarm's operation including search overhead, utilization, sensitivity to node lifetimes, and variation in contention across overlay paths. We first describe our simulation methodology before reporting these results.

Methodology: Our last.fm trace data drives a discrete event simulator for OneSwarm with twenty second timesteps. Each last.fm user is interpreted as a OneSwarm user, friend links in the last.fm social graph correspond to OneSwarm friends, and each unique song request made by a user is interpreted as an object request in the overlay network. Searches are cancelled if 10 distinct paths are discovered.

We assume that all users have unconstrained download capacity, and each user is assigned an upload capacity limit drawn from a measured distribution of BitTorrent capacities [16]. As in our implementation, during periods of contention, capacity is split equally among each user's friends and search messages are not forwarded.

Initially, each user serves as a replica for songs listened to during the first week of our trace, and we begin each trace playback at the outset of the second week. For all users in our trace, we have a record of playback histories at the granularity of weeks. We translate these coarse-grained playback histories to the shorter time-scales required for simulation using measurements of 1,000 randomly sampled users from our trace. At each point during trace playback, clients are added so as to approximate the measured diurnal pattern exhibited by the sampled users. Users are randomly selected to join the system. Each user is assigned a session time (in songs to be played) drawn from the distribution of sampled users. We consider a session to be active until back-to-back playback of songs is separated by more than 30 minutes.

Object sizes are derived from the measured lengths of 81,805 songs listened to by the set of 1,000 sampled users, and we assume a constant data rate of 128 Kbps. To exercise capacity constraints, we increase this data rate to 1 Mbps for indicated trials; this rate is consistent with high quality streaming web video.

To examine the impact of unavailable nodes in the social graph on object availability, we compare trace playback 1) when all users observed in the last.fm trace are active (we refer to this as "always on"), and 2) when users persist in the overlay for 8 hours after playback of the final song of their session. This approximates typical peer lifetimes in BitTorrent [15].

Each trace playback is primed with eight hours of simulation time, and we report results for three subsequent hours of simulation time spanning peak load during the first day of the second week of the trace. Simulation during peak load exercises capacity constraints. We performed a similar analysis of OneSwarm's operation during minimum load, exercising availability, and this produced similar results.

Object availability: A simple metric that distills the feasibility of F2F overlay forwarding is the fraction of objects requests satisfied, i.e., those that discover at least one replica in the overlay. During trace replay, 9% of searches fail for the last.fm workload with both always on and 8 hour lifetimes during peak load. During simulations spanning the time period of minimum load, the fraction of failed searches increases to 12%.

OneSwarm searches can fail for any of three reasons: 1) the song being requested occurred only once in our trace (no replicas exist), 2) all available replicas are offline, or 3) no path exists to the query source from available replicas due to either overloaded or unavailable nodes along the path. Object requests of the first type (no replicas exist) account for 6% of total demand in our trace. These searches are certain to fail and correspond to the songs listened to by just one last.fm user in our trace. This implies that the remaining cases (capacity overload and/or replica unavailability) cause search failures in just 3% of cases when users are always on and 6% of cases with 8 hour lifetimes. This high level of robustness is surprising given that most nodes have few friends, most songs have few replicas, and average path length between random users is high. Several workload properties ameliorate these challenges. First, although users have few friends, even these users are frequently connected to high degree nodes (Figure 3), providing high path diversity. And, although most songs have few replicas, most requests are for popular objects for which many replicas exist (along many paths). This also compensates for typically lengthy path lengths among users.

Overhead: OneSwarm discovers paths to replicas by flooding search messages among friends. Although the majority of data transfer is attributable to the popular minority of objects, the majority of control traffic stems from the unpopular minority of object requests for which search messages are forwarded to nearly every active node in the overlay (during periods of low contention). This is an explicit design choice to improve availability in OneSwarm, but this results in high overhead.

We compute search overhead as the fraction of control messages messages making up overall traffic. For the last.fm workload with always on lifetimes, overhead is 27% of total data traffic. The increased data rate during

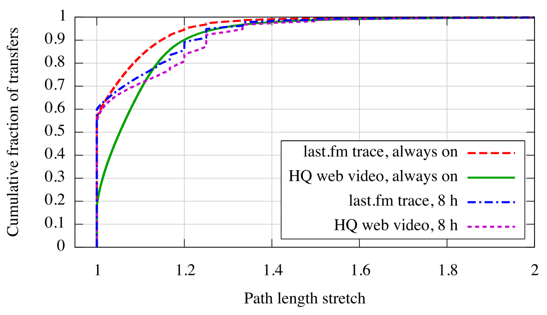


Figure 8: Path length stretch for F2F transfers for various workloads. For the last.fm workload, the majority of transfers use shortest paths. As data volume increases, capacity constraints induce stretch.

video playback reduces the fraction of overhead to 5%. Overhead with 8 hour lifetimes is higher than when peers are always on since the relative low density of the graph makes it difficult to find the 10 unique paths required to cancel the search. For peers with 8 hours lifetimes the overhead is 71% for the last.fm workload and 42% for the video workload. Although large both fractionally and by total volume, recall that search messages are forwarded only when a node has idle capacity. As a result, capacity consumed by control traffic is not capacity lost during data transfers.

An alternative search strategy trades search response latency for a reduction in overhead. Because search messages are forwarded only after a 200 millisecond delay per-hop whereas search cancel messages are forwarded without delay, OneSwarm clients can perform the equivalent of iterative deepening search in the overlay by issuing search queries followed by increasingly delayed search cancel messages. For popular objects with many replicas, this approach discovers replicas quickly. But, if lengthy paths are required, several iterations (and several seconds of delay) may be required to discover replicas.

Stretch: In addition to promoting availability by discovering potentially rare replicas, flood-based search also typically discovers short paths. When objects are large, trading control traffic for short paths is preferable; redundant forwarding of bulk data consumes traffic equivalent to an enormous volume of relatively tiny control messages. We measure how often OneSwarm discovers (and can use) the shortest available paths by computing the path length stretch for transfers during trace replay. We compute stretch as the average path lengths to all replicas used during a file download weighted by the fraction of total data attributable to a given replica. The distributions of stretch for various workload conditions are shown in Figure 8.

The last.fm workload with always on lifetimes is the best case for OneSwarm's operation on the last.fm social graph. Path diversity is high and aggregate demand is much less than aggregate capacity. In this case, OneSwarm uses shortest paths for 55% of transfers with an average path length from search source to data replica of 4.9. 95% of objects have a stretch ≤ 1.2. As expected, stretch increases when lifetimes decrease (8 hour, average path length 5.0) since path diversity in lower. In both cases, a small fraction of requests traverse paths with frequent contention, increasing stretch. Increased data rate (HQ web video) increases stretch as well, but this increase is attributable to contention for bandwidth rather than node unavailability. Even with always on lifetimes, just 21% of video transfers use shortest paths (average path length 6.2). Reducing lifetimes decreases path diversity results in an increase in transfer using shortest paths but those shortest paths are lengthier on average.

Contention: One measure for the load on the OneSwarm overlay is the contention at each node for capacity. We measure contention as the ratio of simulated download time and the download time if no concurrent transfers occur along all transfer paths. We call this the slow-down factor. If a given transfer is the only active transfer on its set of paths, this ratio is 1.

The distributions of contention ratios for our simulated trials are shown in Figure 9. For the last.fm workload with always on lifetimes, contention is rare; 71% of transfer paths are not contended throughout the transfer. As expected, contention increases when either lifetimes are reduced (capacity and path diversity decrease) or data rate increases (capacity constraints strongly influence performance). In the case of our video workloads, many paths have extremely high contention ratios due to the impact of fair queueing over lengthy overlay paths. When contention is high, stretch increases as well-connected nodes shed load, leading to lengthy paths. When data rate is high, transfers are lengthy, with high load and many concurrent transfers traversing core nodes. Fair queueing over lengthy paths with highly contended nodes inflates the slow-down factor substantially for many transfers. These are the regions of the web video distributions in Figure 9 which are not shown (slow-down factor > 20). 18% of transfers fall into this category when users have 8 hour lifetimes and 14% with always on lifetimes.

Utilization: Ideally, OneSwarm would distribute transfer load over all nodes in the overlay, particularly if contention is high. In practice, this is not possible because of the skewed distribution of node degrees and objects. High degree nodes in the social graph occur on a disproportionate fraction of paths, and a node serving as a replica for numerous or popular objects will serve a larger fraction of requests than users hosting few objects. We measure the impact of these workload properties by recording the average utilization of all active nodes during simulations. The distributions of utilization

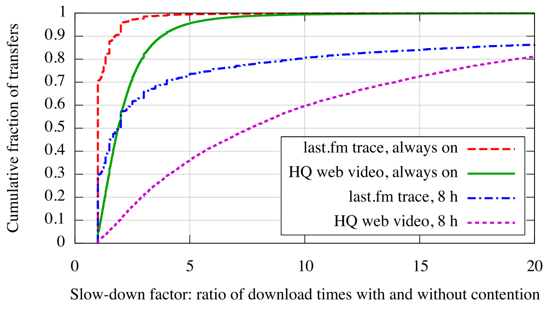


Figure 9: Contention during transfers. Each line gives the cumulative fraction of transfers (y-axis) with a given slow-down factor (x-axis) or less. The slow-down factor is the ratio of the download time using a given set of paths and the download time for the same set of paths if all capacity were available.

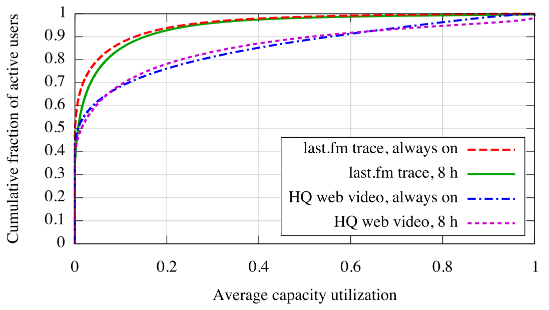


Figure 10: Cumulative fraction of active users (y-axis) with a given average capacity utilization (x-axis) or less. Nearly half of nodes have little to no utilization in all simulated workload scenarios, and few nodes are under constant load (average 1).

are shown in Figure 10.

These results show that many nodes will have fundamentally low utilization regardless of contention, a weakness of restricting sharing to the social graph. Nearly half of nodes forward little if any data during their active periods. These users are the low degree "stub" nodes at the edge of the social network; recall that the median degree of last.fm users is 3. These nodes occur on few forwarding paths, and forwarding accounts for the majority of total utilization. However, among comparatively well-connected nodes, load is distributed more evenly, particularly during simulations of highly contended video workloads. Reducing lifetimes does not significantly change this distribution; utilization is distributed similarly for pairs of always on and 8 hour trials.

4.2 Real-world deployment

While the bulk of our evaluation focuses on simulating OneSwarm's operation at large-scale with real-world sharing workloads, we have made our prototype implementation of OneSwarm available to our own friends. To date, real users have been using OneSwarm in the wild for five weeks on 21 machines spread across the United States, Sweden, and Germany. We report initial measurements of this prototype deployment.

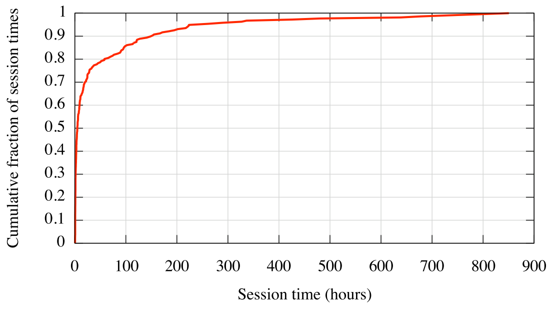


Figure 11: Distribution of session times in our small-scale OneSwarm deployment. While the majority of sessions are brief, the average session time is more than 49 hours.

In spite of its small size and limited amount of shared content, users have transferred a significant amount of data using OneSwarm. Our prototype client regularly polls a server at the University of Washington to check for updates, and, during these polls, clients report limited usage statistics including the total amount of data sent to friends and session duration. Over the course of five weeks, 21 users transferred an aggregate 17.9 GB of data (including forwards), an average of 800 MB per user.

One hypothesis we hope to test by measuring real-world use of OneSwarm is whether or not restricting sharing to friends encourages contribution to the overlay. Since OneSwarm continues to share files after download, one approximate measure of contribution is session times. We summarize session times reported by clients in Figure 11. Although many sessions are short (18% are less than 1 hour), the median session time is 5 hours. A minority of lengthy sessions dominates the average of 49 hours. While these lengthy sessions are a small fraction of total session times observations, these periods account for a large fraction of total OneSwarm activity. In contrast with popular currently P2P networks where average sessions times are measured in hours [14, 15], OneSwarm peers have an average session time of days. Although preliminary, we find these results encouraging.

5 Related work

Providing privacy and anonymity for Internet data transfers is a longstanding goal of the research community, and we draw on many existing ideas in the design of OneSwarm's F2F sharing. Our work differs from existing proposals in three main ways:

- *Goal:* OneSwarm provides privacy against the systematic monitoring of user behavior common in today's P2P networks. An explicit non-goal is to provide strong guarantees of anonymity; this is often the goal of existing systems [7, 10]. Relaxing this constraint allows us to use techniques, e.g., address rewriting, that improve performance at the risk of exposing behavior to an attacker with significant resources.

- *Target environment:* OneSwarm is designed to support the file sharing workloads that currently dominate P2P usage on the Internet. Many existing systems are focused on low-bandwidth environments (e.g., anonymous e-mail) or environments where data reordering and high delays degrade the user experience (e.g., anonymous web browsing). In contrast, bulk data sharing does not share these constraints, simplifying our design.
- *Implementation:* OneSwarm is a complete system that users can download and use today, and it is architected to address longstanding challenges of making source-rewriting F2F networks work in practice. Piggy-backing on existing social networks bootstraps the OneSwarm overlay, and a DHT stores ID $\rightarrow$ IP to simplify locating friends when they change location (or refresh DHCP).

Relaying electronic messages through intermediaries to obscure the source and destination from third parties was first proposed anonymous email in a seminal paper by Chaum [6]. Anonymizer provides anonymization services commercially, providing a centralized service that relays web traffic [1]. Crowds also provides anonymous web browsing by grouping users in a way that obscures which user in a group initiates a request [28].

Tor [10] uses onion routing techniques to anonymize requests via a set of relay nodes. More recent work has shown that the same functionality can be achieved without a public key infrastructure [17]. Tarzan uses similar address rewriting techniques in a P2P context [13]. Although we use similar data forwarding techniques, OneSwarm restricts sharing to friends only and does not use exit nodes. Often, the malicious activity emanating from exit nodes is attributed to their hosting organizations, discouraging contribution to the network. Also, OneSwarm is not architected as a service; to use the network, users must run the client, promoting balanced capacity and demand.

Herbivore [29] enables anonymous file-sharing by providing a more scalable implementation of DC-nets [5]. Herbivore provides strong anonymity but increases overhead relative to address rewriting, which OneSwarm uses.

Turtle is a similar F2F file sharing network that limits communication to a social graph [27]. Like Freenet, Turtle uses only a single path for transfers, limiting performance. Also, Turtle requires manual configuration of keys and manual updates to IP and port information for each friend.

Freenet [7] version 0.7 provides a so-called *darknet* mode of operation that provides anonymous generic data transport through overlay forwarding via trusted friends. Though data transport is similar in OneSwarm, Freenet aims to provide an anonymous storage service where data is public and stored by other nodes in the network. In contrast, OneSwarm users control the sharing of their own data via permissions and store only the data that they produce or have explicitly downloaded. Additionally, objects in Freenet supports hash-based naming of objects only and uses a single path for transfers. In contrast, OneSwarm supports keyword search and multi-path downloads.

Our measurements and analysis of the last.fm workload are largely consistent with existing work that characterizes sharing in P2P networks [2, 14, 23] and usage of popular content sharing sites [4]. Independent measurement efforts have shed light on the properties of popular online social networks [3, 20, 21] as well as the potential for using social networks as a basis for designing new network protocols [22]. Our measurements build on understanding developed in this prior work, combining measurements of a social graph with a trace of sharing activity on that graph, and we make this combined data set available to the community.

6 Conclusion

Although widely used, currently popular P2P networks expose the sharing behavior of their users to monitoring by third parties. To curb the indiscriminate sharing that enables this, we have built OneSwarm, a friend-to-friend file sharing client that restricts direct data sharing to trusted friends with verifiable persistent identities. Associating persistent names with peers gives users explicit control over their privacy by defining sharing permissions at the granularity of data objects and friends. To guide our design, we report measurements of a crawl of the last.fm website, which provides us with a unique trace of both a social network and a social sharing workload. We make this data publicly available and use it as a basis for trace replay, evaluating OneSwarm in a variety of workload scenarios at scale. The OneSwarm client is available for download on Linux, Windows, and Mac OS X, and our experiences with a small-scale OneSwarm deployment point to the feasibility of friend-to-friend sharing in the wild.

References

[1] The anonymizer. *http://anonymizer.com*, 1997.

[2] E. Adar and B. Huberman. Free riding on Gnutella. *First Monday*, 2000.

[3] Y.-Y. Ahn, S. Han, H. Kwak, S. Moon, and H. Jeong. Analysis of topological characteristics of huge online social networking services. In *Proceedings of the 16th inter-*

national conference on World Wide Web, pages 835–844, Banff, Alberta, Canada, 2007. ACM.

[4] M. Cha, H. Kwak, P. Rodriguez, Y.-Y. Ahn, and S. Moon. I Tube, You Tube, Everybody Tubes: Analyzing the world's largest user generated content video system. In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, pages 1–14, San Diego, California, USA, 2007. ACM.

[5] D. Chaum. The dining cryptographers problem: unconditional sender and recipient untraceability. *J. Cryptol.*, 1(1):65–75, 1988.

[6] D. L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2):84–90, 1981.

[7] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: a distributed anonymous information storage and retrieval system. In *International workshop on Designing privacy enhancing technologies: design issues in anonymity and unobservability*, pages 46–66, Berkeley, California, United States, 2001. Springer-Verlag New York, Inc.

[8] A. Clauset, C. Shalizi, and M. Newman. Power-law distributions in empirical data. http://arxiv.org/abs/0706.1062, 2007.

[9] B. Cohen. Incentives build robustness in BitTorrent. *In Proceedings of the 1st Workshop on Economics of Peer-to-Peer Systems*, 2003.

[10] R. Dingledine, N. Mathewson, and P. Syverson. Tor: the second-generation onion router. In *Proceedings of the 13th conference on USENIX Security Symposium - Volume 13*, pages 21–21, San Diego, CA, 2004. USENIX Association.

[11] J. R. Douceur. The Sybil Attack. In *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, pages 251–260. Springer-Verlag, 2002.

[12] J. Falkner, M. Piatek, J. P. John, A. Krishnamurthy, and T. Anderson. Profiling a million user DHT. In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, pages 129–134, San Diego, California, USA, 2007. ACM.

[13] M. J. Freedman and R. Morris. Tarzan: a peer-to-peer anonymizing network layer. In *Proceedings of the 9th ACM conference on Computer and communications security*, pages 193–206, Washington, DC, USA, 2002. ACM.

[14] K. P. Gummadi, R. J. Dunn, S. Saroiu, S. D. Gribble, H. M. Levy, and J. Zahorjan. Measurement, modeling, and analysis of a peer-to-peer file-sharing workload. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 314–329, Bolton Landing, NY, USA, 2003. ACM.

[15] L. Guo, S. Chen, Z. Xiao, E. Tan, X. Ding, and X. Zhang. Measurements, analysis, and modeling of BitTorrent-like systems. *In Proceedings of the ACM/SIGCOMM Internet Measurement Conference (IMC-05*, 2005.

[16] T. Isdal, M. Piatek, A. Krishnamurthy, and T. Anderson. Leveraging BitTorrent for end host measurements. In *In the 8th Passive and Active Measurement Conference (PAM '07)*, 2007.

[17] S. Katti, D. Katabi, and K. Puchala. Slicing the onion: Anonymous routing without PKI. *in Proceedings of the 4th ACM Workshop on Hot Topics in Networks (HotNets*, 2005.

[18] P. Mahadevan, D. Krioukov, M. Fomenkov, X. Dimitropoulos, k c claffy, and A. Vahdat. The internet as-level topology: three data sources and one definitive metric. *SIGCOMM Comput. Commun. Rev.*, 36(1):17–26, 2006.

[19] D. McCoy, K. Bauer, D. Grunwald, T. Kohno, and D. C. Sicker. Shining light in dark places: Understanding the tor network. In N. Borisov and I. Goldberg, editors, *Privacy Enhancing Technologies*, volume 5134 of *Lecture Notes in Computer Science*, pages 63–76. Springer, 2008.

[20] A. Mislove, H. S. Koppula, K. P. Gummadi, P. Druschel, and B. Bhattacharjee. Growth of the flickr social network. In *Proceedings of the first workshop on Online social networks*, pages 25–30, Seattle, WA, USA, 2008. ACM.

[21] 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 SIGCOMM conference on Internet measurement*, pages 29–42, San Diego, California, USA, 2007. ACM.

[22] A. Mislove, A. Post, P. Druschel, and K. P. Gummadi. Ostra: leveraging trust to thwart unwanted communication. In *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, pages 15–30, San Francisco, California, 2008. USENIX Association.

[23] M. Piatek, T. Isdal, T. Anderson, A. Krishnamurthy, and A. Venkataramani. Do incentives build robustness in BitTorrent. *Proceedings of the 4th USENIX Symposium on Networked Systems Design and Implementation*, 2007.

[24] M. Piatek, T. Isdal, A. Krishnamurthy, and T. Anderson. One hop reputations for peer to peer file sharing workloads. In *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, pages 1–14, San Francisco, California, 2008. USENIX Association.

[25] M. Piatek, T. Kohno, and A. Krishnamurthy. Challenges and directions for monitoring P2P file sharing networks –or– Why my printer received a DMCA takedown notice. In *3rd USENIX Workshop on Hot Topics in Security (HotSec '08)*, 2008.

[26] M. Piatek, H. V. Madhyastha, J. P. John, A. Krishnamurthy, and T. Anderson. Limits on ISP-friendly P2P design. In *Under submission*.

[27] B. C. Popescu, B. Crispo, and A. S. Tanenbaum. Safe and private data sharing with turtle: Friends team-up and beat the system. *In Proc. of the 12th Cambridge Intl. Workshop on Security Protocols*, 2004.

[28] M. K. Reiter and A. D. Rubin. Crowds: anonymity for web transactions. *ACM Trans. Inf. Syst. Secur.*, 1(1):66–92, 1998.

[29] E. G. Sirer, S. Goel, M. Robson, and D. Engin. Eluding carnivores: file sharing with strong anonymity. In *Proceedings of the 11th workshop on ACM SIGOPS European workshop*, page 19, Leuven, Belgium, 2004. ACM.

[30] M. Sirivianos, J. Han, P. Rex, and C. X. Yang. Free-riding in BitTorrent networks with the large view exploit. *In Proc. of IPTPS07*, 2007.

[31] K. Zetter. Tor researcher who exposed embassy e-mail passwords gets raided by swedish fbi and cia. *http://blog.wired.com/27bstroke6/2007/11/swedish-researc.html*, 2007.