Privacy-preserving P2P data sharing with OneSwarm

Tomas Isdal* Michael Piatek* Arvind Krishnamurthy* Thomas Anderson*
http://oneswarm.cs.washington.edu/

Abstract

Privacy—the protection of information from unauthorized disclosure—is increasingly scarce on the Internet, and yet increasingly important as every user becomes both a content consumer and a content producer. The lack of privacy is particularly true for popular peer-to-peer data sharing applications, where public rendezvous and dynamic membership mean that user behavior can be easily monitored. In this paper, we describe the design, implementation, and experience with OneSwarm, a new P2P data sharing system that provides users with explicit, configurable control over their data: data can be shared publicly or anonymously, with friends, with some friends but not others, or only among personal devices. OneSwarm is publicly available and has been downloaded by hundreds of thousands of users in the few months since its release. A key goal is to reduce the performance cost of privacy and our measurements of the live system show that anonymized data transfers are performance competitive with unanonymized use. OneSwarm's novel lookup and transfer techniques yield more than an order of magnitude improvement in transfer speeds relative to Tor, another widely-used anonymization system.

1 Introduction

Privacy—the protection of information from unauthorized disclosure—is a long-standing concern of computer system design. Privacy has become of particular concern as users become authors of content, rather than passive consumers, sharing their content and their interests with overlapping sets of people.

At a technical level, privacy is easy to accomplish with centralized solutions. If the user data is stored on a server in a data center, user directives about dissemination can be easily enforced, and data about user interests can be carefully limited or disabled on user request. However, the reality is quite different in practice. Many popular web services require users to sign away their privacy and ownership rights as a condition of service; sites often take advantage of this to collect, store, and share vast amounts of personal data about their users. Most users find this objectionable [30]. Even for vanilla Internet access, ISPs now routinely divulge identifying information about their customers to virtually any third party who asks for it [2]. Censorship is also made easier by centralization, and it is a practical concern in many countries around the globe.

Peer-to-peer (P2P) data sharing systems potentially provide an option for achieving scalability and privacy without relying on centralization. With P2P, because resources are contributed by users, there is no inherent need to sacrifice privacy. But, most widely-used P2P systems trade off privacy against usability, leaving us with little in the way of a practical alternative to cloud based solutions. On one side, systems like BitTorrent are high performance and robust, but everyone's activities are visible to anyone who cares to look. (Our research group has monitored tens of millions of BitTorrent users worldwide from a dozen machines at UW.) On the other, anonymization systems like Tor and Freenet emphasize privacy but at the cost of poor performance and robustness, in part because of misaligned incentives and inefficient protocol choices such as single path routing. In our performance evaluation, for example, OneSwarm provides more than an order of magnitude improvement in transfer rates relative to Tor.

In this paper, we describe the design, implementation, and experience with a privacy-preserving file sharing service called OneSwarm, intended to reduce the "cost" of privacy by focusing on usability concerns: ease of setup, support for a variety of different sharing and trust models, interoperability with users satisfied with public data sharing, as well as high efficiency and robustness. In OneSwarm, data objects are located and transferred through a mesh of untrusted and trusted peers populated from user social networks. We argue that combining trusted and untrusted peer relationships provides better privacy and robustness than either approach would alone. Content lookup and transfer is anonymous, congestion-aware, and multipath, providing good performance at reasonable overhead even for rare objects and diverse peer bandwidths.

OneSwarm is part of a larger effort to build an alternative to cloud computing that does not depend on centralized trust, including services for rendezvous, lookup, long-term storage, remote computation and the like. We tackle privacy first because it is very poorly handled in popular P2P systems, and yet privacy needs to be an essential feature of such systems in our view. We stress that privacy is of value for many legitimate reasons. Some say: "nothing to hide, nothing to fear" but

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

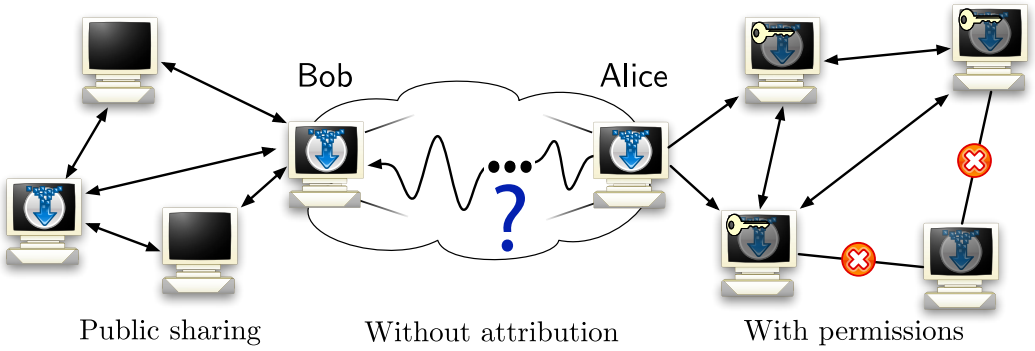

Figure 1: An example of the range of data sharing scenarios supported by OneSwarm. Bob downloads public data using OneSwarm's backwards compatibility with existing BitTorrent implementations, and makes the downloaded file available to other OneSwarm users. Alice downloads the file from Bob without attribution using OneSwarm's privacy-preserving overlay, but she is then free to advertise the data to friends. Advertisements include a cryptographic capability, which allows only permitted friends to observe the file at Alice.

we do not agree. For example, most YouTube content is freely re-distributable. Using P2P techniques would save YouTube hundreds of millions of dollars per year, but its users would likely object if as a consequence, their every search request was monitorable by third parties with minimal effort.

OneSwarm has been downloaded by hundreds of thousands of users, with active user groups in many countries, disproving the notion that "no one cares about privacy." [21] We use this deployment as the basis for our evaluation, collecting voluntarily reported usage statistics from users as well as measurements of instrumented OneSwarm clients running on PlanetLab [25]. Because our measurements of the live system are limited by the privacy needs of our users, we complement our study with simulations of OneSwarm against a trace of object sharing patterns and social connectivity of more than 1 million users of the last.fm music service [3].

The remainder of this paper is organized as follows. Section 2 outlines the OneSwarm data sharing and workload model. We describe how we manage identities and trust in Section 3 and our congestion-aware data lookup and transfer algorithms in Section 4. We conduct a brief security analysis in Section 5, evaluate the performance of our system in Section 6, and discuss our deployment experience in Section 7. We discuss related work in Section 8 and conclude in Section 9.

2 Data sharing with OneSwarm

OneSwarm is designed to allow users to share data efficiently and securely while preserving their privacy when desired. Virtually everyone on the Internet is both a content producer and a content consumer, with a diverse set of constraints on who should be allowed access to any piece of content or usage pattern. One could design separate systems for each usage model, e.g., one for anonymous publication (Freenet [10]), another for anonymous download (Tor [13]), yet another for controlled sharing with friends. A tenet of our work is to support a range of data sharing scenarios efficiently within a single framework. Our motivation is pragmatic: like BitTorrent, the performance of our system improves with increasing number of users, and it is more natural to present the user with a single interface than separate systems for each type of data.

2.1 Sharing scenarios

Figure 1 illustrates the range of privacy preserving options supported by OneSwarm. In this example, suppose users Alice and Bob both want to download a left-leaning political podcast. Suppose further that Bob does not consider his political views to be sensitive information, but Alice would prefer that her political views not be made public; instead, she might want to share the podcast with just a few like-minded friends.

OneSwarm supports all of these levels of privacy *within the context of a single swarm*. Bob downloads the podcast from a *public* set of existing BitTorrent and OneSwarm peers. During the download, Bob also acts as a replica for sharing *without attribution* using an overlay consisting of OneSwarm peers only. This overlay acts as a mix [9], using source-address rewriting and multi-hop overlay forwarding to obscure the identities of a path's source and destination. Alice is one such destination, and she downloads the podcast using only anonymizing paths to preserve her privacy from third-party monitoring. But, she is free to advertise the file explicitly to friends who may also be interested in the content.

Each case shown in Figure 1 imposes a different tradeoff between privacy and efficiency. Publicly distributed data is not private, and direct transfers between a large set of replicas yield efficient distribution. Sharing data with permissions limits access and hence distribution ca-

pacity. Finally, data shared without attribution is accessible by anyone, but the set of users sharing the data is obscured, which increases overhead. To summarize:

- **Public distribution:** All data sharing need not be private. This is the case for which existing P2P systems excel, and OneSwarm draws on this strength by serving as a fully backwards compatible BitTorrent client. This helps bootstrap content into OneSwarm's privacy preserving overlay; data originally obtained using legacy protocols can be easily shared using any other mode. Sharing recorded course lecture videos is an example of this type of distribution.
- **With permissions:** Persistent identities allow OneSwarm users to define per-file permissions. In this case, *access* to files is restricted (rather than attribution of source or destination). In OneSwarm, capabilities restrict access to protected files, allowing all permitted users to recognize one another and engage in swarming downloads for scalability.[1] For example, OneSwarm can be used to restrict the distribution of a photo archive to friends and family only.
- **Without attribution:** When sharing sensitive data, privacy depends on obscuring *attribution* of source and/or destination. Unlike data shared with permissions, which is directly advertised, data shared without attribution is located using privacy-preserving keyword search, and data transfers are relayed through an unknown number of intermediaries to obscure source and destination. This type of distribution is appropriate for sensitive material. Since it is up to the user to define what is sensitive, the same data object may be shared under all three of the models simultaneously.

To the best of our knowledge, OneSwarm is the first data sharing system that unifies all of these common data sharing scenarios without relying on centralized trust. Many existing P2P systems like BitTorrent provide efficient public distribution, but lack basic mechanisms for supporting access control or privacy. Anonymous publishing systems, e.g., Freenet [10], allow data sharing without attribution, but exclude access control by design and require participants to act as a cache for the (potentially objectionable) content shared by others. A similar problem is inherent in the design of traffic anonymization systems based on onion routing, e.g., Tor [13], wherein potentially malicious traffic is attributable to the exit node of an onion route, creating a severe disincentive to host a node. As a result, such networks are woefully underprovisioned relative to demand. We consider these and other related systems in more detail in Section 8.

[1] Of course, capabilities (or data itself) can be relayed to others once obtained, but OneSwarm's default behavior is to maintain restrictions on data shared with permissions unless explicitly overridden.

2.2 Workload constraints

To guide the initial design of OneSwarm, we conducted a large scale study of the object sharing behavior of over a million users of the last.fm music web site. We initially expected that most or all of the peering links in OneSwarm to be formed between directly connected friends. (Our deployment showed that this assumption was often violated, for reasons apparent in the last.fm data, discussed below.) last.fm is unique in providing information about both the object sharing patterns and the social graph of its users. Previous characterizations of social networks measure graph structure alone, while previous studies of file sharing omit social relationships.

We summarize the results from this study to provide context for OneSwarm's design choices; a more complete description can be found in the appendix.

- *Skewed object popularity motivates popularity-aware search:* The object popularity in last.fm is heavily skewed; the top 5% of objects account for 79% of total demand. Even so, rarely requested objects comprise a significant portion of the overall demand. To support this workload, the mechanism used for finding content must be able to efficiently find popular content while still being able to locate unpopular objects.
- *Long paths motivates multipath downloads from a single source:* In last.fm, the average path length between users is 7.1. In an overlay with similar structure, the diversity of end-host bandwidth capacities means that any single path is likely to be slow, limited by its lowest-capacity and/or most congested link. To provide good performance, OneSwarm uses multiple paths per-source to transfer data.
- *A resilient core improves availability but requires adaptation to congestion:* last.fm has significant path diversity and a very resilient core. But, the popularity of a minority of well-connected users suggests that as the amount of traffic in the network increases, OneSwarm must be able to find alternate routes to avoid congested nodes.
- *Bootstrapping is crucial since many users have few trusted links:* As with many social networks, popularity is highly skewed in last.fm, and the majority of users have few social links. In an overlay, this would reduce both performance and privacy: downloads are efficient only when there are multiple path options, and privacy can likewise be more easily compromised for users with very limited fanout. For such users to benefit from OneSwarm, our design includes mechanisms for both trusted and untrusted overlay links.

These constraints shape OneSwarm's control and data transfer protocols as well as how users manage and define trust relationships, the topic we describe next.

3 Managing identities and trust

Supporting the range of data sharing scenarios described in Section 2 requires OneSwarm to expose a range of options for managing trust. Sharing data with friends only, for example, requires some notion of identity to allow users to relate real-world trust relationships to overlay connections. Robust data sharing without attribution does not depend on trust in any individual peer, but rather on the obfuscating effects of randomized data transfer via multiple peers and paths. Sharing a file with different privacy options changes the details of how data transfer occurs, which we describe in the next section. In this section, we describe 1) OneSwarm's notion of identity, 2) how users link identities to social relationships for sharing data with permissions, and 3) how groups of potentially untrusted peers are matched for sharing without attribution. We discuss each of these in turn.

3.1 Identity and connectivity

Each OneSwarm user is named using a cryptographic key that identifies that user among its peers. Each user generates a 1024 bit public/private RSA key pair when installing the client, with the public key serving as its identity. OneSwarm identities are *persistent*, allowing two users that have exchanged keys to locate and connect to one another whenever both are online. Long-term identities are linked to transient IP-addresses and port numbers via a distributed hash table (DHT) maintained among all users. 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 peer.

Multiple insertions of connectivity information enable fine-grained control over 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 information disclosure in the DHT for each peer.

DHT entries for a client P are signed by P and encrypted with the public key of a given peer. Each entry is indexed by a 20 byte randomly generated shared secret, which is agreed upon during the first successful connection between two peers. Prior to the initial connection with a newly added friend, P temporarily advertises connectivity information at a special location: the SHA-1 hash of the concatenation P's public key and the public key of the given friend. This location serves as the initial rendezvous point.

In our implementation, ID $\rightarrow$ {IP, Port} mappings are stored in a Kademlia-based DHT using twenty-fold replication for fault tolerance [20]. This level of replication has been shown to provide high availability for DHTs running on end-hosts [14]. Each client's location in the DHT is independent of its identity and is determined 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, which is certified during DHT operations by other OneSwarm peers.

3.2 Linking peers with trust relationships

The OneSwarm DHT tells a client how to connect to a given peer provided the peer's public key is known. But, this requires users to first obtain keys. In existing social-sharing P2P designs [10, 27], key exchange is typically manual. We view manual exchange as a hindrance to adoption and include multiple methods for automatically exchanging identities.

Between two OneSwarm users that share a real-world trust relationship, OneSwarm automates key exchange in three ways. First, as in UIA [15], the OneSwarm client discovers and exchanges keys with other OneSwarm users over the local area network. Second, we piggyback on existing social networks, e.g., Google Talk or Facebook, to distribute public keys. 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. Third, users can email invitations to friends. Invitations include a one-time use capability that authenticates the recipient during an initial connection, during which public key exchange occurs.

For all methods described above, users can choose whether to accept new and updated keys. This allows users to maintain separate lists of OneSwarm contacts and contacts from other social services, while still avoiding the inconvenience of manually exchanging keys with friends out-of-band.

3.3 Managing groups and untrusted peers

Exchanging keys manually, via existing social networks, or through email invitations all depend on users having preexisting relationships with their peers. While appropriate when fine-grained control is required, in many circumstances explicitly authorizing every peer relationship is cumbersome and unnecessary. For example, OneSwarm is frequently used by communities of users with dynamic membership but mutual pairwise trust, e.g., a group of friends or colleagues. In this case, users need to maintain a *subscription* to keys.

To support key management within a group, OneSwarm allows users to subscribe to one or more *community servers*. A community server maintains a list of registered users and provides authorized subscribers with a current set of public keys upon request. In effect,

subscribers to a given community server delegate trust regarding a subset of their peers to the operator, who vets prospective members. When configuring their subscriptions, users decide whether to apply updates automatically or only after manual approval. When configuring their community server, operators decide between authenticated or public access as well as the number of members to provide to each subscriber. XML-encoded peer lists are delivered to the OneSwarm client via HTTP secured using SSL, and requests to authenticated community servers use standard HTTP authentication mechanisms.

In addition to supporting automatic key exchange among trusted groups, community servers also allow OneSwarm users to easily obtain a set of *untrusted* peers that increase robustness and privacy when sharing data without attribution. Bootstrapping early adopters is a significant challenge for overlay networks based on mutual trust between directly connected peers. But, in the case of sharing without attribution, trusted peers are not required; privacy depends on the obfuscation provided by forwarding data through multiple unknown intermediaries. Untrusted peers are used for this purpose only and serve to bootstrap overlay connectivity when users have few trusted friends.

Since registration with public community servers is unrestricted, all peers obtained from one are treated as untrusted by default. Registration itself is a three step process. First, the OneSwarm client provides its public key, which the server then verifies by issuing a challenge nonce value and verifying the incremented, encrypted response. Finally, the server uses consistent hashing of the key to compute a subset of peers to return to the client.

Community server registration is designed to inhibit systematic crawling of the membership list of a public community server. Verifying keys with a challenge/response allows the server to limit the number of registrations by a single IP address, and consistent hashing limits the information obtained from repeated membership queries. Although an attacker with significant resources can evade these restrictions and obtain a complete view, doing so is of limited value. The overlay topology is an amalgam of links from community servers, manual exchanges, email invitations, and other social networks; a crawl of community servers provides only a partial view, and more privacy conscious users need not subscribe to any community server whatsoever. We consider the effectiveness of attacks enabled by public community servers in more detail in Section 5.

4 Locating and transferring data

At this point, we have described how OneSwarm peers join and maintain overlay connections and update connectivity information. We next turn to the protocol used to name, search for, and transfer data.

Our overall approach is inspired by the success of existing P2P swarming systems, e.g., BitTorrent, and we adopt existing swarming techniques wherever possible with three exceptions. First, instead of sharing all data publicly with a dynamic set of peers, OneSwarm users explicitly define the trust level of a persistent set of peers (by default peers are untrusted). Second, 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 overlay. Third, instead of sources sending data directly to receivers, data transfers occur over the reverse overlay search path, using address rewriting to obscure sender and receiver identities.

A source of complexity in our design is the need to support a mix of trusted and potentially untrusted peers. Indeed, our initial implementation assumed mutual pairwise trust among directly connected peers in order to simply our protocol and security analysis. But, this requirement was largely ignored by many of our initial users. This section outlines the random perturbations of the timing and delivery of protocol messages needed to support untrusted peers, but we delay a more complete discussion of attacks and defenses until Section 5 to first provide a complete protocol description.

In the remainder of this section, we separate our discussion into three parts: how users discover peers and data sources in the overlay, how data is exchanged, and what incentives are provided to contribute resources.

4.1 Naming and locating data

OneSwarm peers connect to one another using secure sockets (SSLv3) bootstrapped by their RSA key pairs. When two peers 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 peer has permissions. For each privately shared file the meta-data includes a 512-bit capability that is used as a symmetric encryption key for use during transfers. After the initial file list is received, subsequent lists include diffs only.

Naming: Shared files (or groups of files) are named in OneSwarm using the 160 bit SHA-1 hash of their name and content. The low order 64 bits of this hash are used to identify swarms in search messages that are flooded to discover potential data sources. For public data, users obtain content hashes 1) out-of-band, e.g., from an email or website, 2) from file list messages exchanged with peers, or 3) from keyword search in the overlay. For private data the user must obtain both the hash of the data as well as capability used for decryption. We describe transfer setup via search since this subsumes the other cases.

Congestion aware search: OneSwarm search is de-

signed to manage the tradeoff between overhead and performance by being *congestion aware*. Using the shortest path minimizes overhead, but risks poor performance if the shortest path is slow or overloaded. Given that highly connected users are more likely to appear in a path, this is a practical concern.

OneSwarm addresses this by managing the propagation of searches. Because the path taken by a search message determines the path of data transfer, the key idea is to forward searches along the shortest path possible (to limit overhead) subject to each intermediary's current load (to improve performance).

To discover shortest paths, OneSwarm relies on flooding. Keyword search messages include a randomly generated search ID and list of keywords. Unlike flooding search in other P2P file sharing networks, OneSwarm search messages do not include a time-to-live value since this information would allow intermediaries nearby the source or destination to easily reason about behavior. Instead, OneSwarm forwards searches to trusted peers provided the forwarder has idle capacity and the search has not been forwarded previously. Clients maintain a history of search messages to avoid forwarding duplicates.

Among untrusted peers, forwarding is randomized to prevent collusion attacks. Instead of forwarding unmatched search messages to all peers, OneSwarm forwards searches to untrusted peers probabilistically. This inhibits colluding untrusted peers from inferring a data source by observing the lack of a forwarded search message. To prevent information leakage through repeated queries, the decision to forward a search is made randomly —but deterministically— so repeated queries for the same data will yield the same result.

To avoid the propagation of every search to every client in the overlay, each client delays each search message for at least 150 milliseconds before forwarding it to peers. 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. (Search cancels are also sent if the upstream peer disconnects.) The search cancel message is forwarded along the same paths as the corresponding search message but without any forwarding delay, allowing cancel messages to quickly reach the search frontier.

In addition to the fixed forwarding delay for search cancellation, OneSwarm also delays messages based on the load at each intermediary. Where load is high, search propagation will tend to route around it, improving performance. When excess capacity exists, search messages will follow the shortest path, reducing transfer overhead.

Path setup: If a node is sharing a file that matches a search query, it does not forward the search and instead responds with a search reply message. Among

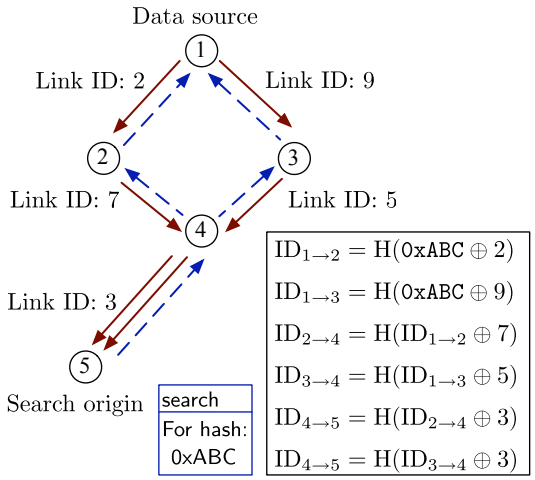

Figure 2: An example of end-to-end path ID computation. Client 5 searches for peers with file ID 0xABC and queries are forwarded along the dashed links.

trusted peers, this response is immediate. But, receiving a search reply message in less than 150 ms (our default per-hop forwarding delay) would reveal the responder as a data source to potentially untrusted peers. To prevent this, users delay search reply messages (and all protocol messages) sent to untrusted peers in order to emulate the delay of a longer path. This value is chosen randomly between 150-300 ms (i.e., 1–2 hops). As with forwarding of search messages, the delay value is persistent for a particular file and a particular peer to prevent information leakage from repeated queries.

Search reply messages include a search identifier, a list of content hashes which identify matching files, file metadata, and a *path identifier*. The path identifier allows clients to distinguish among multiple paths even if those paths partially overlap. We first describe how path IDs are computed and then how they are used to enable multipath and multi-source downloading. Each peer maintains a randomly chosen *link ID* for each peer link.[2] 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 peer (who forwarded the data request) by computing the SHA-1 hash of the initial value XOR'd with the link ID of the given peer. 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 2. The ability to recognize unique paths allows the receiver to add new paths during the course of a download. Transfers can start as soon as a one path is discovered, and new

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

searches can be launched to replace paths that fail.

4.2 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 BitTorrent file to transfer data, first obtaining a list of block hashes corresponding to the metadata stored in .torrent files [12]. But, rather than connecting directly to peers, OneSwarm tunnels BitTorrent traffic through overlay paths. Each overlay path is treated as a virtual peer, even those that terminate at the same endpoint. Of course, the receiver has no definitive way to know which paths terminate where. Rather than obtaining a list of peers from a centralized tracker, as in BitTorrent, OneSwarm discovers new paths by periodically flooding search messages for active downloads.

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 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 [1]. For example, if a path becomes congested traffic will automatically be shifted to the paths that do not traverse the congested link. If a forwarding node disconnects, the capacity of the data-source is automatically shifted to the other paths. Building OneSwarm on an existing P2P network and popular client also helps in bootstrapping the overlay. In addition to its privacy-preserving features, OneSwarm serves as a vanilla BitTorrent client; publicly shared files can also be shared privately with OneSwarm peers, bootstrapping content in the overlay.

4.3 Incentives

Persistent identities and long-term relationships provide a rich foundation on which to implement different incentive strategies. Each OneSwarm client maintains transfer statistics for each peer including total data uploaded and downloaded, maximum transfer rates, control traffic volume, and uptime.

We retain BitTorrent's default tit-for-tat policy for making servicing decisions among multiple virtual peers. This creates an incentive to contribute capacity while downloading, improving swarm performance. Persistent identities also create a strong incentive to continue sharing data after downloads complete. During periods of contention, our default policy is to allocate bandwidth among directly connected peers proportionally; each peer is assigned a weight equal to the ratio of their net contribution and net consumption. When this ratio is greater than 1, a peer is a net contributor. A client improves its standing over time by participating in the system whenever possible.

Across all peers, forwarding data is zero sum. Data consumption from the ingress peer connection is matched by contribution at the egress. At the granularity of individual paths, it is difficult to reason about whether a particular forwarding connection is helpful for a peer's long-term interests. If the egress point is a peer often on the path of a client's own transfers, forwarding contributions will improve subsequent local performance. But, if the ingress peer is a more useful data source, forwarding will reduce long-term performance. To cope with this, OneSwarm uses a default forwarding policy inspired by peering relationships between ISPs. If the incoming/outgoing traffic ratio of a peer is approximately balanced or greater than 1 over the long-term, forwarding is permitted. But, if this ratio is significantly unbalanced, forwarding is not permitted during periods of contention. This default policy can be overridden. Users are free to assign static weights per-peer or forward data without regard to traffic imbalance.

In practice, our default policy has proven sufficient to induce a surplus of forwarding capacity in the system. We verify this in our performance evaluation (Section 6).

5 Security Analysis

OneSwarm's overarching security goal is to improve privacy by allowing users to control information disclosure. When sharing data with permissions, disclosure is limited by familiar mechanisms: strong identities, capabilities, and end-to-end encryption. In this section, we focus on providing privacy in the more challenging case of data sharing without attribution. In this case, our goal is to be resistant to the disclosure of user behavior to an attacker with control over a limited number of overlay nodes. Specifically we assume attackers lack complete knowledge of the current overlay structure and that users are conservative when specifying trusted peers. We point out, however, that an explicit non-goal is to eliminate the possibility of monitoring by a highly capable monitoring agent with global wiretap capabilities or the ability to seize specific computers.

In the remainder of this section, we outline several potential attacks and quantify their effectiveness using measurements of OneSwarm users in the wild. In the

appendix, we explore a wider range of threats: inferring data sources, associating search requests to users, identifying trusted links, and so on. Because of space limitations, we restrict our attention to what we believe to be the most likely attackers conducting the most likely attacks: one or more colluding OneSwarm users bootstrapped via community servers attempting to infer the source of a data transfer. The discussion highlights the following aspects of the OneSwarm protocol that significantly enhance user privacy.

- *Persistent peering relationships limit monitoring power:* In BitTorrent, peers are dynamically assigned, allowing attackers to become a peer of virtually everyone, given enough time. By contrast, OneSwarm peers are persistent, improving contribution incentives but also limiting the ability of attackers to inject nodes at arbitrary locations in the overlay.
- *Heterogeneity of trust relationships foils timing attacks:* OneSwarm users define links as either trusted or untrusted and keep this information private. As the protocol behavior varies with link type, the combined use of trusted and untrusted links greatly diminishes an attacker's ability to infer the length of an overlay path based on timing information.
- *Lack of source routing limits correlation attacks:* OneSwarm does not provide peers with the ability to construct arbitrary overlay paths. Attackers could use this to correlate performance with ongoing transfers. Such an attack is known to degrade privacy in Tor, for example [32]. Individual clients have a limited view of the overlay and cannot control path setup beyond directly connected neighbors.
- *Constrained randomness frustrates statistical attacks:* The uncertainty arising from random perturbations in the protocol could be reduced through statistical analysis if repeated probes yielded different draws. OneSwarm prevents such analysis by making all random decisions deterministically with respect to a given query and link.
- *Network dynamics limit value of historical data:* While relationships in OneSwarm are long lived, the end-to-end paths between senders and receivers change rapidly due to churn and transient congestion. This reduces the window of opportunity for adversaries to combine data from multiple observations in order to reverse-engineer user behavior.

5.1 Inferring data sources

Timing attack: By measuring the round trip time (RTT) of search / response pairs, an attacker can estimate the proximity of a data source. Usually, paths are lengthy, making the chances of being next to any particular data source quite low. For a small number of requests, however, an attacker might be directly connected to a data source and also be able to identify it as such based on the low RTT of response messages.

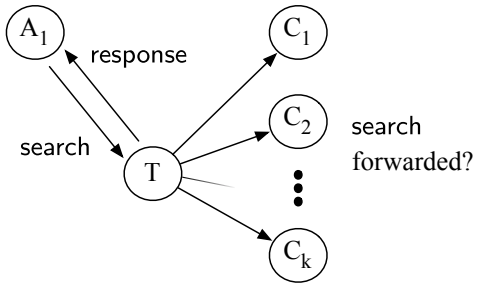

Figure 3: An attacker, A, with $C_1, ..., C_k$ colluders tests if a target T is sharing a file.

To frustrate this attack, OneSwarm artificially inflates delays for queries received from untrusted peers; all responses to untrusted peers are delayed by a random but deterministic amount (computed based on the content hash) in order to emulate the delay profile of forwarded traffic from one or more hops away.

Even when data sources choose the minimum artificial delay, the RTT observed by an attacker is indistinguishable from that of a data source that is two overlay hops away and connected via low latency, trusted forwarding links. In other words, the combined use of trusted and untrusted links provides many more possible explanations for a given delay profile than a system that uses only untrusted links.

Collusion attack: Next, we analyze the case of multiple colluding peers as illustrated by Figure 3. In this example, A sends a targeted search to T, receives a search response, and observes whether the search was forwarded to colluding peers $C_1, ..., C_k$. Recall that forwarding search messages is probabilistic to provide deniability. Each search message has a configurable probability, p_f, of not being forwarded to a particular peer. As a result, a lack of forwarding does not definitively identify a data source; missing search messages may arise from random chance. But, a lack of forwarding observed by many colluding peers is highly suggestive of T sourcing the object. Assuming a fixed forwarding probability of p_f and k colluders, $\Pr[\text{Not source}|\text{response received}] = (1-p_f)^k$. With just a few colluders, an attacker can gain very high confidence.

Although effective, this attack requires both attacker and colluders to be directly connected to the target. The most likely avenue for this is a public community server to which the target subscribes. Community servers give a random set of users to each client. As a result, the likelihood of an individual attacker being matched with a specific target for a community server with N members is $\frac{n}{N}$, where n is the number of peers returned for a single request, 26 by default. To prevent an attacker from sys-

tematically crawling the entire set of community server peers, key registrations are limited per-IP and per-prefix, and the set of randomly returned peers is determined by performing a consistent hash on the requesting client's initially provided public key.

As a specific example, consider achieving greater than 95% confidence in the identification of a data source given $p_f = 0.5$ for peers received from a community server.[3] Achieving 95% confidence in identification requires at least six directly connected peers (an attacker and five colluders). For a community server with N users, the likelihood of achieving a particular number of direct connections is given by the complement of a binomial CDF with success probability $\frac{n}{N}$. In the case of a community server returning $n = 26$ peers with 1,000 users, the probability of 30 attackers achieving six direct connections with a target is much less than 1%. Attempting to achieve six or more connections with *any* peer (rather than a specific target) increases the likelihood of success to 10%. More broadly, the effectiveness of either variant of this attack in practice depends on the resources of an attacker relative to the population of a public community server. Privacy depends on this ratio being small, and privacy-conscious users are free to decrease their forwarding probability (p_f) or avoid public community servers completely. By contrast we note that we were able to monitor the interest patterns of tens of millions of BitTorrent users with only a dozen machines at UW.

5.2 Deconstructing overlay paths

Our discussion so far has considered attacks aimed at confirming whether a specific user is sharing a particular object. We next consider the more generic attack of attempting to locate *any* data source for a particular object, but without having a specific target a priori. This requires first deconstructing the overlay path to a potential data source before testing if it is sharing the object. To do this, a group of attackers can use coordinated measurements of search response message propagation to infer the likely next hop along an overlay path, monitor or attempt to peer with that client, and then repeat.

The feasibility of this attack depends on the length, stability, and diversity of paths to the object. Lengthy paths require more iterations to deconstruct, during which time the path may vanish due to mesh dynamics. Similarly, the existence of a large, dynamic replica set and/or many paths creates an ever-changing "direction" towards sources, confounding inference based on search response RTTs. We find that this is frequently the case for the OneSwarm workload; search response messages do not have a consistent next hop, even for back-to-back searches monitored by many vantage points.

[3]Low values of p_f for community server peers are offset by the high amount of path diversity among them.

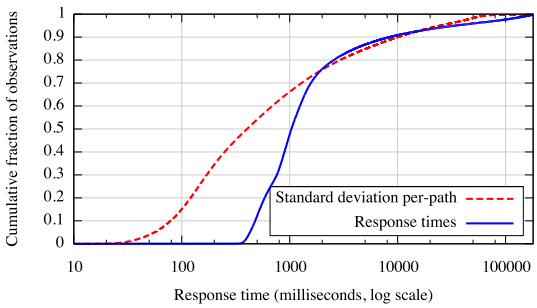

Figure 4: The distribution of search / response RTTs and the distribution of variance for RTTs on identical overlay paths with more than 10 search responses.

To evaluate this, we analyze search response RTT measurements collected by a set of PlanetLab nodes running instrumented OneSwarm clients. As with would-be attackers, these nodes are bootstrapped via public community servers. Each node monitors all search requests it forwards, recording the RTTs of search response messages. For a given search, the peer responding with the least RTT across all measurement nodes is the likely next hop to the data source. We measure the stability of first responders for back-to-back search requests; i.e., is the first responder for a given search the same as the first responder for the next search? With ten vantage points, 65% of back-to-back searches have the same first responder. Surprisingly, increasing the number of vantage points to 100 *reduces* back-to-back consistency to 63%. On the whole, it is difficult to reason about the likely direction of search response messages since the ordering of responses is highly variable.

The unpredictable ordering of search response messages is attributable to the naturally large variations in message delays. Figure 4 summarizes the distribution of response RTTs for more than 42 million searches. Large RTTs suggest lengthy paths; the majority of search response messages are observed more than one second after forwarding their corresponding search. Even so, a variety of confounding factors make reasoning about path length on the basis of delay difficult. OneSwarm is willing to tolerate lengthy queueing delays at congested nodes (up to 7 seconds in our current implementation). Since search response messages are interleaved with data traffic, response times may be controlled by either 1) network propagation delay, 2) lengthy overlay queueing delay at congested intermediaries, or 3) the protocol-imposed propagation delay of search messages. These effects manifest in significant variations in RTTs for even identical paths (i.e., responses carrying the same path ID). We point out that this data was collected before the reduction of the minimum search delay to 150 ms in the publicly available client release and also before the inclu-

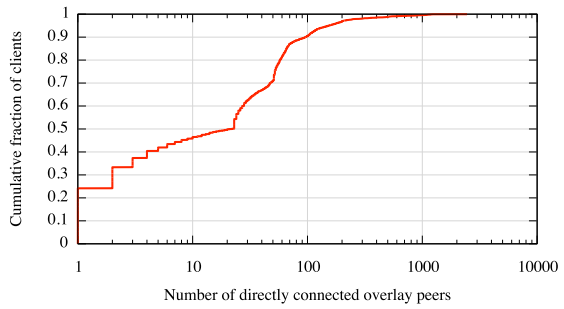

Figure 5: Cumulative distribution of peers per-client.

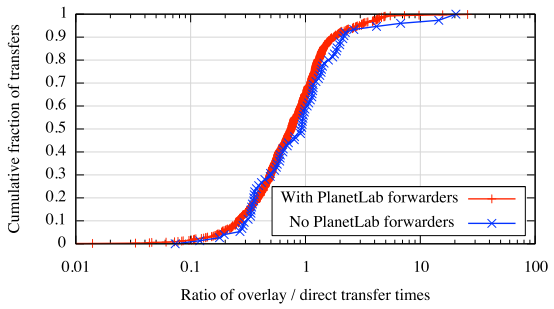

Figure 6: Comparing transfer times mediated by the OneSwarm overlay to direct transfer.

sion of randomized search response delays, and so the current implementation is likely to exhibit even greater variability, but a smaller minimum delay.

Very capable monitoring agents can use these types of attacks to deduce some activities of OneSwarm users in limited cases. But, systematically monitoring user behavior requires significant effort and resources, e.g., to quickly compromise the machines of multiple overlay hops to deconstruct paths. Compared to the ease with which third parties monitor P2P networks today, OneSwarm provides users with substantial privacy gains.

6 Evaluation

To evaluate OneSwarm, we measure its performance and robustness both in the wild and synthetically using trace replay. OneSwarm has been downloaded hundreds of thousands of times to date, and we use a combination of both voluntarily reported user data as well as instrumented clients to quantify OneSwarm's real-world effectiveness at the scale of thousands of users. To examine OneSwarm's operation at even larger scale, we replay traces of the social graph and usage behavior of more than one million last.fm users. In both cases, our main result is that OneSwarm provides high throughput and availability in spite of the overhead arising from preserving privacy. In support of this conclusion, we also measure the effectiveness of OneSwarm's protocol mechanisms and report usage and workload statistics.

6.1 Real-world deployment

Methodology: Although many aspects of user behavior are (deliberately) obscured by designing for privacy, we draw on two sources of data to profile overall system overhead, utilization, and performance. The first of these is voluntarily reported summary statistics from more than 100,000 distinct OneSwarm users collected over a seven month period. These include the total number of peers, how frequently various peer import methods are used, and aggregate data transfer volumes.

Our second source of data is instrumented OneSwarm clients running on 150 PlanetLab [25] machines. Subscribing to several public community servers bootstraps connectivity for these clients, providing each with dozens of random OneSwarm peers. Our PlanetLab nodes act as passive vantage points to measure the the background forwarding traffic in the overlay. To date, this has resulted in an average of 766 GB of traffic forwarded per day.

We have also measured other properties of OneSwarm's workload such as session times, geographic distribution, network-level locality, diurnal usage patterns, upload and download capacity distributions, NAT status, and object popularity. These results are generally consistent with existing studies of widely-used P2P networks.

Overlay structure: Although many overlay links in OneSwarm are based on social relationships, the graph structure overall is strongly influenced by the random matching of public community servers, as well as the tendency for many users to import a large number of keys en masse from websites maintaining active user lists.

Both of these effects are reflected in the distribution of overlay peers per user shown in Figure 5. This distribution shows significant variations in connectivity. While some users maintain hundreds or even thousands of peer connections, the median value is just 22. The sudden increase in mass near this value is attributable to community servers, which return 26 peers by default. Subsequent increases arise from users subscribing to multiple community servers. For clients reporting data, 53% of peers are imported from community servers, 46% manually, with the remaining 1% of peers coming from LAN, email invitations, or social network import.

Overhead: OneSwarm uses multihop overlay forwarding to share data without attribution, introducing significant overhead relative to direct point-to-point transfers. Given the lengthy paths suggested by our measurements of search response message timings, a concern is that forwarding demands might overwhelm overlay capacity and degrade end-to-end performance.

To quantify the impact of overhead on transfer performance, we compare the time taken by transfers 1) when mediated by overlay forwarding and 2) when using a

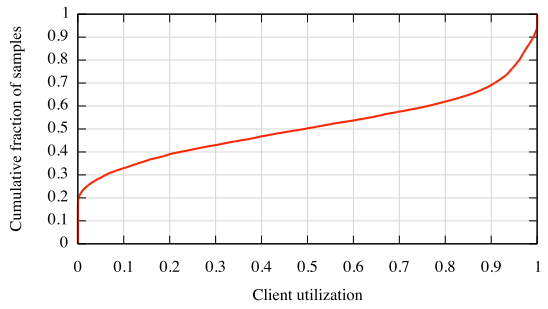

Figure 7: The distribution of client upload capacity utilizations over the course of one day. Although most clients have excess capacity, transient congestion occurs at many nodes.

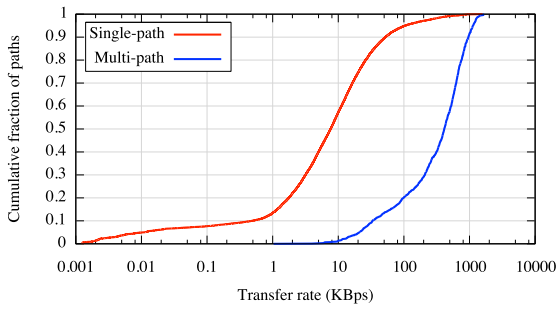

Figure 8: A comparison of single and multi-path transfer performance.

direct point-to-point connection between sender and receiver. If the overlay is not capacity constrained, we would expect both transfers to have a similar duration, on average, and indeed, we find this to be the case for transfers conducted between our PlanetLab nodes.

Figure 6 summarizes the ratio of the overlay and direct transfer times between our PlanetLab nodes. There are two cases. We first measured transfer times when sharing random data between pairs of 20 PlanetLab nodes and while disabling all other PlanetLab clients; i.e., the overlay did not benefit from any additional forwarding capacity. We measured transfers between 75 pairs chosen randomly without replacement. A ratio of 1.0 means that overlay and direct transfers took identical time, with ratio > 1 indicating a faster direct transfer and ratio < 1 indicating a faster overlay transfer. This is a worst case for OneSwarm as PlanetLab nodes are generally of higher capacity than the typical OneSwarm peers doing the forwarding. In addition the download had only one data source ruling out any performance gains from multi-source downloads. Even without the addition of PlanetLab forwarding capacity, overlay transfers does not impose a performance bottleneck in most cases, some transfers are faster and some slower with the median ratio of overlay and direct transfer times being 0.94.

We next investigated whether adding PlanetLab forwarding capacity to the overlay would improve transfer times. We repeated the experiment over several weeks and between all our PlanetLab hosts, comparing performance for 683 pairs of transfers. In this case the median performance ratio is 0.76; i.e., more often than not, transfers mediated by the overlay complete *faster* than direct point-to-point transfers. We attribute these performance gains to OneSwarm's use of multiple overlay paths causing favorable TCP effects due to concurrent TCP connections and potentially lower per-hop RTT.

Utilization: Although the overlay benefits from a surplus of capacity in aggregate, individual paths and individual nodes are often congested, motivating our use of congestion-aware search and multi-path transfers. To confirm this, we examine each user's reported utilization over time. For the set of users reporting transfer volume statistics, we compute the maximum transfer rate over all reported 15-minute intervals and treat this as the capacity for a given IP address, computing utilization for all other 15 minute periods relative to this maximum. These samples are summarized in Figure 7. Although average utilization is 49%, many nodes are frequently bandwidth limited; node utilization is 95% or greater during 23% of measured intervals. In short, temporarily overloaded clients are not uncommon despite the overlay being over-provisioned on average.

Multi-path transfer performance: Unlike systems that anonymize traffic at the packet level, OneSwarm data transfers can tolerate out-of-order data delivery, allowing us to use multi-path and multi-source transfers to improve performance and robustness. This is crucial in wide-area P2P environments defined by heterogeneity. Each individual path exhibits the bandwidth capacity of its slowest link. Given the highly skewed bandwidth capacity distribution of P2P nodes, the capacity of individual multi-hop paths is typically low.

To confirm this, we compare the multipath transfer rates achieved between PlanetLab nodes during overlay transfers to the performance of separately measured individual forwarding paths. Both distributions are summarized in Figure 8. Multi-path transfers average 457 KBps, while single path transfer rates average just 29 KBps. As an additional comparison, we measured transfer rates achieved when routing traffic over Tor between the same set of PlanetLab nodes, which yielded an average transfer rate of 20 KBps. The combination of transient congestion, bandwidth heterogeneity, and potentially lengthy paths all contribute to the benefits of multi-path transfer, which is essential for providing good performance and robustness.

6.2 Trace replay in the last.fm social graph

Our evaluation of OneSwarm in the wild is constrained by our limited view of the network and its topology. To

complement this, we use trace data from the last.fm music website to drive a large-scale simulation of file sharing mediated by a social overlay network. The site allows users to publish their music playback histories to others and define social relationships. We crawl these histories to build a trace of the user behavior and social relationships of 1.7 million users. last.fm's workload is a challenging case for OneSwarm as the overlay structure is sparse and limited to social links only. In practice, many OneSwarm users complement their trusted friend links with untrusted links from public community servers. In this section, we apply this trace to OneSwarm. Additional details regarding our crawl and analysis are available in the appendix.

Methodology: Our last.fm trace data drives a discrete event simulator with ten second timesteps. Each last.fm user is interpreted as a OneSwarm user, friend links in the last.fm social graph correspond to OneSwarm peers, and each unique song request made by a user is interpreted as an object request in the overlay network. Searches are cancelled when 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 [18]. Each user starts as a replica for songs that user listened to during the first week of our trace, and we begin the trace playback at the outset of the second week. Object sizes are derived from the measured lengths of songs, 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 evaluate the impact of user lifetimes on 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 eight hours after playback of the final song of their session.

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, 11% 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 24% as a large fraction of the network becomes disconnected because of the sparse nature of the last.fm overlay.

Searches can fail for any of three reasons: 1) the song being requested occurred only during the second week of 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 5% of cases during peak load and in 18% of cases during minimum load.

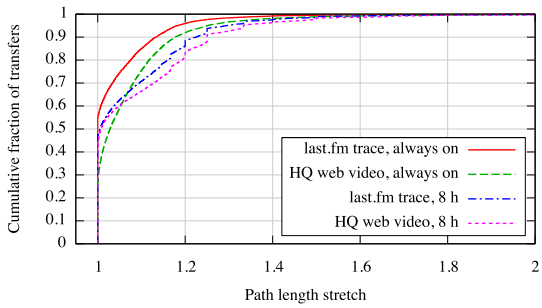

Figure 9: Path length stretch. For the last.fm workload, the majority of transfers use shortest paths. As data volume increases, capacity constraints induce stretch.

Overhead: OneSwarm discovers paths to replicas by flooding search messages among friends. Although the majority of data transfered is due to popular objects, the majority of control traffic stems from requests for unpopular object 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.

We compute search overhead as the fraction of control 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 video playback reduces the fraction of overhead to 6%. 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 hour lifetimes, the overhead is 77% for the last.fm workload and 43% 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, assuming unconstrained download capacity.

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; reducing the number of forwarding hops for bulk data can save the equavilent of 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 9.

The last.fm workload with always on lifetimes is the best case. 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 source to replica of 4.8. 95% of objects have a stretch ≤ 1.2. Path diversity is reduced when lifetimes decrease (8 hour, average path length 5.1); this increases stretch. 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. With always on lifetimes, just 28% of video transfers use shortest paths (average path length 5.8).

7 Deployment experience

Since its release, OneSwarm's evolution has been guided by feedback from the user community. Broadly, our experience has been extremely positive, with enthusiastic users providing debugging insights, language translations, and suggestions for future improvements. We summarize two aspects of user behavior and feedback that have had a fundamental impact on the evolution of OneSwarm's design.

Bootstrapping requires nurturing communities: Our initial software release included three methods of exchanging keys to bootstrap overlay connectivity: 1) importing contacts Google Talk (GTalk), 2) local network discovery, and 3) manual exchange. Our expectation was that the majority of users would prefer the automatic management provided by GTalk key import and rarely use other options. This was wrong.

In practice, the most common method of bootstrapping connectivity among early adopters was manual key exchange. Thousands of users exchanged keys freely on the public message board at OneSwarm's website. Surprisingly, several technically savvy power users set up dedicated websites for so-called *regional key sharing*, wherein users from a particular country could exchange keys to foster data sharing among a community with a single language and/or shared interests. Users of these sites provide their public key and are provided an up-to-date list of keys from other members in turn. (Unsurprisingly, software support for rapidly importing multiple keys was the most frequently requested feature during this time.) This model for key exchange motivated the design and implementation of community servers, which have largely supplanted manual key sharing sites.

Users ignore inconvenient trust assumptions: Because we expected that peer connections would be based primarily on social relationships, our initial design assumed mutual trust among directly connected peers. Unfortunately, even technically savvy users typically ignored this requirement, adding peers from public bulletin boards. For most users, the sophistication required for launching attacks, even when directly connected, provided sufficient privacy to make performance and availability their primary concern.

Assuming trust among directly connected peers greatly simplified the security analysis of our initial design by removing the challenging case of a directly connected attacker. Since this assumption was ignored, we provided protocol support for untrusted peers and to consider explicitly the possible attacks of this case.

8 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 our design.

Privacy: Relaying electronic messages through intermediaries to obscure the source and destination from third parties was first proposed for anonymous email by Chaum [9]. Anonymizer provides anonymization services commercially, providing a centralized service that relays web traffic [4]. Crowds [28] provides anonymous web browsing by randomly tunneling requests via other system participants. Herbivore [29] enables anonymous file-sharing by providing a more scalable implementation of DC-nets [8]. Herbivore provides strong anonymity at the cost of significantly increased overhead relative to address rewriting. Our focus on bulk data distribution leads us to adopt a design that adapts these classic techniques to modern workloads.

Tor [13] 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 [19]. Tarzan uses similar address rewriting techniques in a P2P context [16]. Although we use data forwarding for privacy, OneSwarm does not have exit-nodes. Often, the malicious activity emanating from exit nodes is attributed to their hosting organizations, discouraging users from hosting exit nodes. Also, OneSwarm is not architected as a service; to use the network, users must run the client, promoting balanced capacity and demand.

OneSwarm differs from all these systems in its support for a spectrum of data-sharing models and peer trust relationships. Our deployment showed that this diversity was needed in practice.

Trust: Incorporating real-world trust relationships has been a crucial design element in several recently proposed systems. SybilGuard [31] uses properties of social networks to ferret out synthetic identities in social systems. In Ostra [24], the scarcity of social connections is used to combat spam. UIA [15] provides data routing

and name resolution over a socially constructed overlay of personal devices. Turtle [27] is a file-sharing application that limits direct communication to only the social graph in an attempt to circumvent third-party monitoring. Freenet [10] version 0.7 includes a so-called *darknet* mode of operation that is similar, restricting transfer to a social connections only.

Our experience suggests that using social connectivity alone is insufficient for many users. Instead, OneSwarm augments a social topology with a variety of additional untrusted links to ease bootstrapping, improve robustness, and by allowing for a mixture of peer sources further enhance privacy. Anonymous publishing systems such as Freenet provide anonymous storage for public data 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.

Workload: Our measurements and analysis of the last.fm workload are largely consistent with existing work that characterizes sharing in P2P networks [5, 17, 26] and usage of popular content sharing sites [7]. Independent measurement efforts have shed light on the properties of popular online social networks [6, 22, 23]. 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.

9 Conclusion

Although widely used, currently popular P2P file sharing networks expose user behavior to silent, third party monitoring. This occurs even when the material being shared is completely legitimate. To address this, we have built OneSwarm, a file sharing system designed to reduce the cost of privacy to the average user. We develop novel techniques for efficient, robust, and privacy-preserving lookup and data transfer. We provide users flexible control over their privacy by defining sharing permissions and trust at the granularity of individual data objects and peers. The OneSwarm client is publicly available for download on Linux, Mac OS X, and Windows, and it is in widespread use around the globe. Our measurements with the live OneSwarm deployment show that it delivers on its promise: privacy-preserving downloads on OneSwarm are roughly as fast as a direct Internet transfer between the two nodes, and an order of magnitude faster than using Tor for the same operation.

References

[1] Azureus. *http://azureus.sourceforge.net*.

[2] DMCA: The Digital Millennium Copyright Act of 1998.

[3] last.fm. *http://last.fm*.

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

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

[6] Y.-Y. Ahn, S. Han, H. Kwak, S. Moon, and H. Jeong. Analysis of topological characteristics of huge online social networking services. In *Proc. of WWW*, 2007.

[7] 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 *Proc. of IMC*, 2007.

[8] D. Chaum. The dining cryptographers problem: unconditional sender and recipient untraceability. *J. Cryptol.*, 1, 1988.

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

[10] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: a distributed anonymous information storage and retrieval system. In *Proc. of Privacy Enhancing Technologies*, 2001.

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

[12] B. Cohen. Incentives build robustness in BitTorrent. *Proc. of P2PEcon*, 2003.

[13] R. Dingledine, N. Mathewson, and P. Syverson. Tor: the second-generation onion router. In *Proc. of USENIX Sec.*, 2004.

[14] J. Falkner, M. Piatek, J. P. John, A. Krishnamurthy, and T. Anderson. Profiling a million user DHT. In *Proc. of IMC*, 2007.

[15] B. Ford, J. Strauss, C. Lesniewski-Laas, S. Rhea, F. Kaashoek, and R. Morris. Persistent personal names for globally connected mobile devices. In *Proc. of OSDI*, 2006.

[16] M. J. Freedman and R. Morris. Tarzan: a peer-to-peer anonymizing network layer. In *Proc. of ACM CCS*, 2002.

[17] 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 *Proc. of SOSP*, 2003.

[18] T. Isdal, M. Piatek, A. Krishnamurthy, and T. Anderson. Leveraging BitTorrent for end host measurements. In *Proc. of PAM*, 2007.

[19] S. Katti, D. Katabi, and K. Puchala. Slicing the onion: Anonymous routing without PKI. *Proc. of HotNets*, 2005.

[20] P. Maymounkov and D. Mazières. Kademlia: A peer-to-peer information system based on the XOR metric. In *Proc. of IPTPS*, 2002.

[21] S. McNealy. On the Record: Scott McNealy. San Francisco Chronicle, 2003-09-14, page I-1, 2003.

[22] A. Mislove, H. S. Koppula, K. P. Gummadi, P. Druschel, and B. Bhattacharjee. Growth of the flickr social network. In *Proc. of the first workshop on Online social networks*, 2008.

[23] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee. Measurement and analysis of online social networks. In *Proc. of IMC*, 2007.

[24] A. Mislove, A. Post, P. Druschel, and K. P. Gummadi. Ostra: leveraging trust to thwart unwanted communication. In *Proc. of NSDI*, 2008.

[25] L. Peterson, T. Anderson, D. Culler, and T. Roscoe. A blueprint for introducing disruptive technology into the Internet. *SIGCOMM Comput. Commun. Rev.*, 2003.

[26] M. Piatek, T. Isdal, T. Anderson, A. Krishnamurthy, and A. Venkataramani. Do incentives build robustness in BitTorrent. *Proc. of NSDI*, 2007.

[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 Intl. Workshop on Security Protocols*, 2004.

[28] M. K. Reiter and A. D. Rubin. Crowds: anonymity for Web transactions. *ACM Trans. Inf. Syst. Secur.*, 1998.

[29] E. G. Sirer, S. Goel, M. Robson, and D. Engin. Eluding carnivores: file sharing with strong anonymity. In *Proc. of ACM SIGOPS European workshop*, 2004.

[30] J. Turow, J. King, C. J. Hoofnagle, A. Bleakley, and M. Hennessy. Americans Reject Tailored Advertising and Three Activities That Enable It. *SSRN eLibrary*, Sept. 2009.

[31] H. Yu, M. Kaminsky, P. B. Gibbons, and A. Flaxman. SybilGuard: defending against sybil attacks via social networks. In *Proc. of SIGCOMM*, 2006.

[32] Y. Zhu, X. Fu, R. Bettati, and W. Zhao. Anonymity analysis of mix networks against flow-correlation attacks. In *Proc. of GLOBECOM*, 2005.

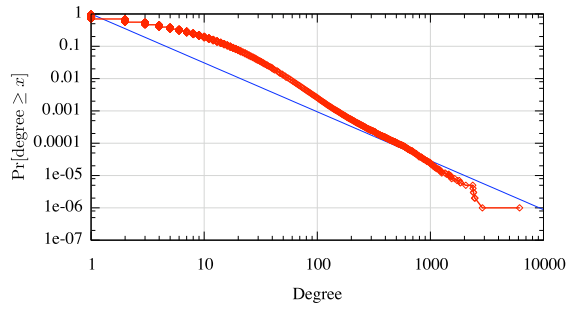

Figure 10: Complementary cumulative distribution (CCDF) of degrees for all users in the last.fm trace. A best-fit power law distribution is shown ($\alpha = 1.51$) for comparison.

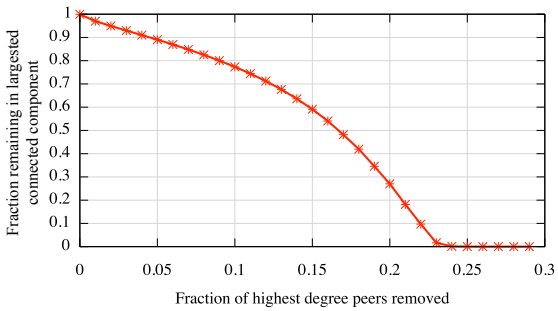

Figure 11: The fraction of nodes in the largest connected component of the last.fm social graph (y-axis) as an increasing fraction of high degree nodes are removed (x-axis).

Appendix

A last.fm workload

In this section, we report additional details regarding our measurements of the last.fm workload.

A.1 Social network

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.[4] 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.

Degree distribution: Figure 10 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 have just one social link, the median degree is 3, and 81% of users have 10 or fewer friends in last.fm. This is in many ways the worst case for our work: reaching the majority of fringe users requires longer average path lengths. Also shown is a best-fit power law distribution ($\alpha = 1.51$) obtained using the maximum likelihood method [11]. The Kolmogorov-Smirnov goodness-of-fit metric for the fit is 0.137. Unlike other social networks, the last.fm degree distribution does not strongly follow a power-law.

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

A.2 Resilient core

Social networks tend to have a highly connected core of nodes. 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, and some nodes become completely disconnected.

For our purpose, understanding the structure of the core is crucial for system design. If most paths *necessarily* transit the core, these nodes will need to manage carefully the sharing of scarce resources. But, if significant path redundancy exists, core nodes can (and should) be avoided during periods of congestion.

To understand which of these effects dominates, we perform the following analysis. After removing a fraction of the highest degree nodes from the graph, we compute the resulting connectivity and repeat this removal for an increasing fraction of nodes. The results are summarized in Figure 11. Connectivity degrades slowly, suggesting the existence of redundant paths around any highly connected nodes. This data differs somewhat from previous studies of online social networks [23]. For example, Mislove et al. showed that removing only 0.01% of nodes split off over twenty percent of users into their own disconnected islands, while leaving most of the rest connected. We speculate that this difference is due to last.fm lacking publish/subscribe support for extremely popular nodes; lacking these nodes, the last.fm graph is already split into a connected component and many isolated users. President Obama may (as of this printing) have millions of "friends", but he is unlikely to mediate file sharing requests for each of them. The Flickr connected component fractured completely after the removal of 10% of the highest degree nodes; in contrast, the last.fm social graph fractured after removing 24% of the highest degree nodes. At the very least, our data indicates more diversity in resilience among social graphs than previously thought, and we caution therefore that

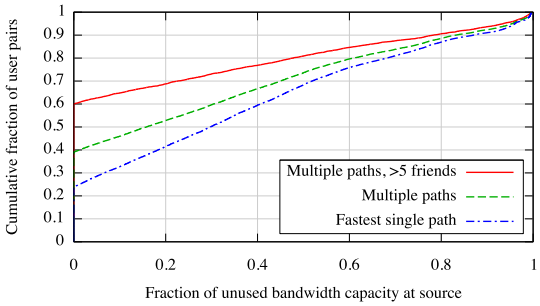

Figure 12: Unused client bandwidth for transfers involving either the fastest single path, multiple paths, or multiple paths for the subset of clients with more than five friends

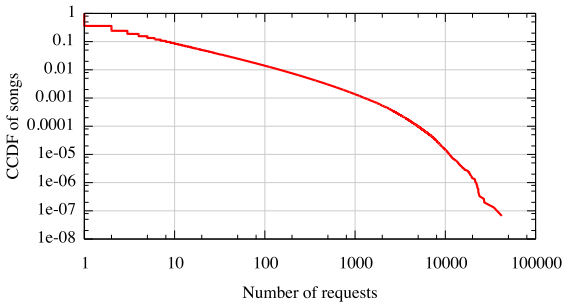

Figure 13: CCDF of the number of unique users listening to a given song for all observed songs.

our results may not generalize beyond our data set.

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, these results call for an adaptive design. High load on core nodes should be detected and alternate paths used. But, in circumstances where such paths are the only option, resource sharing must be effective.

A.3 Path properties

The average path length in the last.fm social graph is 7.1, and the diameter is 14.[5] 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 [23]. We attribute this difference to the absence of very high degree nodes in the last.fm data set, and to the relative prevalence of low degree nodes; both factors increase path length.

Longer average path lengths present a challenge for multi-hop overlay forwarding; any single path is likely to have some node with limited capacity, and each path is only as fast as its slowest link. However, we lack the ability to measure the bandwidth of each last.fm user. Instead, we synthesize this data by assigning each user in the last.fm social graph a bandwidth capacity, drawn randomly from a previously measured bandwidth distribution of BitTorrent users [18].

Figure 12 compares 50,000 randomly selected {source, receiver} pairs in terms of utilization of sender's capacity, for various transfer disciplines. This data shows the potential for improvement from using multiple paths. Even assuming we could find the fastest single path, just 24% of user pairs saturated the sender's capacity. This increases to 39% when using multiple paths. With multiple paths, performance is limited by the large fraction (nearly 30%) of last.fm users with only a single friend. The most significant increase in performance comes from combining multiple paths *and* multiple friends. In this case, 60% of senders are fully utilized. Figure 3 conservatively assumes only a single source for a specific piece of data; we relax that assumption next.

[5]Because computing all shortest paths in such a large graph is not computationally feasible, these results are based on a sample of 50,000 randomly selected user pairs.

A.4 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 replication. We first consider object popularity in terms of requests per object. This is shown in Figure 13. 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, as expected, popular objects account for the majority of total demand. Figure 14 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. 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

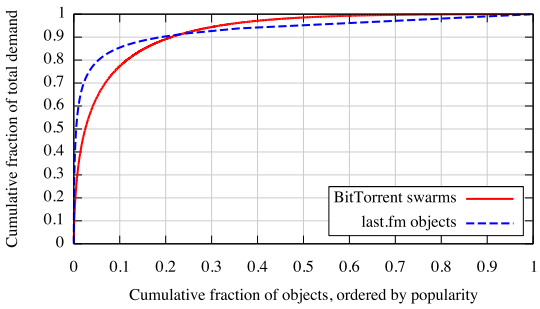

Figure 14: Cumulative distribution of object demand in the last.fm and BitTorrent workloads.

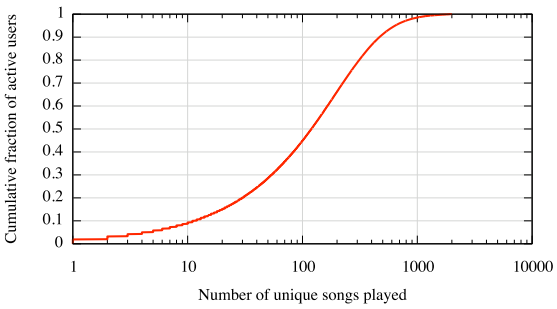

Figure 15: The cumulative fraction of users (y-axis) playing a given number of unique songs (x-axis) or fewer in our two week trace.

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.

The implications of this data for the design of OneSwarm are twofold. 1) The skew in object popularity implies that many requests will be for popular objects with plentiful replicas; locating these will not require a thorough search of the entire overlay, presenting an opportunity to reduce overhead. 2) But, to provide high availability for less unpopular objects, OneSwarm should be able to conduct a thorough search if needed.

Demand per user: Figures 13 and 14 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 15. Demand varies by orders of magnitude; some user histories include 10s of songs while others include 1000s. This type of skew in demand is typical of object request workloads. While one might expect heavy users of last.fm to also have many friends, the length of play history and the number of friends are only 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 in Figure 15 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. Multiplying this value by the average song length in bits (weighted by popularity) gives an estimate for the total demand. Assuming an audio bitrate of 128 Kbits/s, total demand for measured last.fm users over two weeks 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. The number of unique songs added by the second week of our trace was roughly half the unique songs discovered during the first week.

B Supplemental security analysis

While different attackers might seek a wide range of information we are focusing our analysis towards protecting the information that our users consider most important to protect.

- *Sharing behavior:* Who is sharing a certain file F? / Is a person X sharing file F? This is the primary information we are protecting and our aim is to make attacks aimed at revealing sharing behavior difficult even for powerful attackers.

- *Content of privately shared files:* Not all files in OneSwarm is shared with the everyone in the network. OneSwarm supports private sharing and keeping access to this information to the users allowed is critical.

- *User interest:* Who is searching for file F? / Did person X initiate a certain search?

- *Overlay structure:* Some attacks are easier to mount if the global overlay structure is known by the attacker. Because of this we do not want to reveal information about the overlay structure unnecessarily, on the other hand, limited information about the overlay structure is of lesser value so leaking some information is acceptable.

B.0.1 Adversary capabilities:

OneSwarm aims to protect against a wide range of attackers, the attacker could be anything from an private corporation monitoring user behavior or a large number of user, to a nosy person trying to figure out if a friend is sharing a certain file. Below is a list of capabilities that we expect attackers to be able to possess. We enumerate (but do not consider further) attacks and attackers described in Section 5.

- *Arbitrary OneSwarm users:* An attacker that is connected to the OneSwarm network and thus *indirectly* connected to every other OneSwarm user in that component of the network.

- *Untrusted peers:* A user connected directly connected to a OneSwarm user, Alice, as an untrusted peer. This could either be a peer that is marked as untrusted by Alice or a trusted friend that wishes to get information about files that Alice has chosen to not show to that particular friend thus making him untrusted with regards to those files.

- *Colluding untrusted peers*

- *Trusted friends*

- *Local wiretap:* An attacker that can monitor all Alice's network traffic. This encompasses everything from law enforcement with wiretap permission to a hacker running a rouge access point or monitoring traffic at an open wireless network.

We have systematically investigated which attacks against the OneSwarm network that can be launched by users with different capabilities. An overview of these can be found in Table 1.

B.1 Attacker with local wiretap

In this section we consider attacks from an attacker that can monitor all network traffic to and from Alice. This could for example someone at a coffeeshop monitoring an open wireless network, a employer monitoring a corporate network or a small ISP monitoring its customers. It should be noted that if Alice was using any of the currently popular P2P networks an attacker with this capability would have complete knowledge of what she is downloading/uploading and searching for.

Inferring overlay links: An attacker with local packet sniffing capability will be able to see which IPs Alice is connected to enabling them to discover the IPs of the connected friends.

Inferring the source of data: Since the attacker can see only encrypted network traffic, it will not be possible to know which files are shared by a person. However, an attack can inspect the differences in the amount of data uploaded and downloaded. If the amount uploaded is larger than the amount downloaded, it indicates that the person is sharing *some* unknown data at that time. Since OneSwarm will discard any data queued for forwarding when overlay channels are closed, the natural churn in the system will cause some uncertainty regarding the accuracy of uploaded and downloaded volumes.

Inferring the source of searches: An attacker can only see the encrypted network traffic, so it will have to rely on the difference in uploaded and downloaded traffic to be able to detect if the person is performing a search. Because of the small size of search messages and the fact the a OneSwarm user constantly is forwarding searches for other users, this attack requires a very low volume of background traffic to be successful. To further limit the usefulness of this attack, even a successful attack would only be able to detect that the user performed a search at that time, while the content of the search would not be visible to the attacker.

B.1.1 Attacks by friends in the social network

Inferring the source of data: Alice has complete control over which files that are visible to which friends, even if the friends are trusted. Alice is free to change OneSwarms behavior towards them on a per-file-per-friend basis. If Alice allows a peer to see a certain file that file is included in a file list, that file can be requested by the peer. If Alice does not allow a certain friend to see a certain file, Alice will treat any requests from that peer as though received from an untrusted peer.

Inferring the source of searches: The vulnerability in this case is the same as for colluding untrusted friends. We point out that before Alice starts the download, she can specify which peers that can see the file. If she allows a peer to observe a file, that peer will be able to see the file in Alice's file list once the download starts.

B.1.2 Attacks by people distant in the network

Inferring social links: Reasoning 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 a user's Google Talk contact list could

Attacker	Infer source of data	Infer source of search	Infer overlay links
Internet user	Absolute privacy	Absolute privacy	DHT inital friend connect attack
OneSwarm user	Can get user/IP of "likely next hop" and rough estimate of hop count	Can get user/IP of "likely previous hop"	Search timing attack discovering if 2 untrusted friends are friends with each other
Untrusted peer	Rough estimate of hop count, lowest possible estimate will be at least 2 hops giving Alice plausible deniability	Know if Alice is "likely previous hop"	Search timing attack discovering which directly connected peer are friends with Alice
c *colluding untrusted peers*	Same as *single untrusted* + search forward attack with $P(falsepositive) = ((1 - p_f)^{c-1})$	same as *single untrusted*	same as *single untrusted*
Trusted peer	Exposed by design	Exposed by design	Same as *single untrusted*
Local wiretap	Know existance of transfer but not content / final destination	Know existance of search but not content / final destination*	Get IP of Alices currently connected peers
Untrusted peer + local wiretap	Know if Alice is sharing file with hash h	TCP reset spoof attack → Know if Alice is source of search	Same as *Local wiretap*

Table 1: Information discovered by attackers with different capabilities, *In the absense of background traffic

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 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 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). This is frustrated by randomized delays and the high level of background traffic in the network.

B.2 Combination attacks

In this section we consider attacks where the attacker has several of the capabilities listed above. Rather than enumerating all possible combinations we will instead discuss attacks that can be launched with limited resources but still provide the attacker with information.

B.2.1 Local packet sniffing, untrusted peer link:

Here we consider an attacker that can monitor all of Alice's network traffic and is connected to Alice via an untrusted peer link. The attacker seeks to deduce whether Alice is sharing content with hash h. He will monitor the difference in uploaded and downloaded bytes on Alice's network interface. Unless Alice's upload link is saturated with other uploads, the difference in uploaded vs downloaded bytes will change as the attacker is starting and stopping the download of h, while the download is running Alice will upload more data than she downloads. By repeatedly starting and stopping the download the attacker can look for a matching pattern in Alice's network activity. The existence of such a pattern implies that Alice is sharing the file.

B.2.2 Local packet sniffing, indirectly connected to Alice through the OneSwarm network

Here we consider an attacker that can monitor all of Alice's network traffic and in addition is connected to the OneSwarm network and indirectly connected to Alice through any number of intermediaries. The attacker will lanuch the start/stop download attack described above. If a pattern emerges in Alice's network traffic Alice is sharing the specific file. For this attack to be successful the following must be true:

- Alice has spare upload capacity
- The attacker is close enough to Alice in the overlay that searches for the hash will reach her before getting dropped.
- The file is rare enough or attacker close enough that

the search will not be canceled before reaching Alice.

- The overlay path(s) between the attacker and Alice has enough spare capacity to cause a significant change in Alices network traffic.

Very capable monitoring agents can use these types of attacks to deduce some activities of OneSwarm users in limited cases. Compared to the ease with which third parties monitor P2P networks today, OneSwarm provides users with substantial privacy gains.