Privacy-Preserving P2P Data Sharing with OneSwarm

Tomas Isdal Michael Piatek Arvind Krishnamurthy Thomas Anderson

University of Washington

ABSTRACT

Privacy—the protection of information from unauthorized disclosure—is increasingly scarce on the Internet. The lack of privacy is particularly true for popular peer-to-peer data sharing applications such as BitTorrent where user behavior is easily monitored by third parties. Anonymizing overlays such as Tor and Freenet can improve user privacy, but only at a cost of substantially reduced performance. Most users are caught in the middle, unwilling to sacrifice either privacy or performance.

In this paper, we explore a new design point in this tradeoff between privacy and performance. We describe the design and implementation of a new P2P data sharing protocol, called OneSwarm, that provides users much better privacy than BitTorrent and much better performance than Tor or Freenet. A key aspect of the OneSwarm design is that users have explicit configurable control over the amount of trust they place in peers and in the sharing model for their data: the same data can be shared publicly, anonymously, or with access control, with both trusted and untrusted peers. OneSwarm's novel lookup and transfer techniques yield a median factor of 3.4 improvement in download times relative to Tor and a factor of 6.9 improvement relative to Freenet. OneSwarm is publicly available and has been downloaded by hundreds of thousands of users since its release.

Categories and Subject Descriptors C.2.4 **[Computer-Communication Networks]:** Distributed Systems

General Terms Design, Performance, Security

1. INTRODUCTION

Privacy—the protection of information from unauthorized disclosure—is increasingly scarce on the Internet due to the increasing centralization of data sharing. Most Internet users are both content consumers and content producers, with their data shared with others through centralized web sites such as Facebook, YouTube, and Flickr. However, most popular web sites collect, store, and share vast amounts of personal data about their users, despite users finding such behavior objectionable [36]. Even if we trust web sites with our usage data, centralization makes censorship much easier, a practical concern in many places around the globe.

Peer-to-peer (P2P) systems potentially provide an alternative for achieving scalable file sharing without a trusted web site as mediator. However, the P2P systems available today offer users an unattractive choice between privacy and reasonable performance. On one side, protocols like BitTorrent are high performance and robust, but participants are easily monitored by anyone who cares to look [33, 30, 29]. On the other, anonymization networks (e.g., Tor [14] and Freenet [12]) emphasize privacy, but offer comparatively poor performance.

In this paper, we explore a new design point in this tradeoff between privacy and performance. We describe the design and implementation of a file-sharing protocol called OneSwarm, intended to provide much better performance than Tor and Freenet, and much better privacy than BitTorrent. Our goal is to provide good enough performance that users turn on privacy by default for all of their non-public data sharing. To this end, data objects shared via OneSwarm are located and transferred using disposable, temporary addresses and routed indirectly through an overlay mesh, providing resistance to the systematic monitoring of user behavior. Content lookup and transfer is congestion-aware and uses multiple overlay paths, providing good performance at reasonable overhead even for rare objects and diverse peer bandwidths.

Central to our design is a notion of *flexible privacy*. OneSwarm does not adopt a universal guarantee regarding information exposure; each individual user is free to control the tradeoff between performance and privacy by managing trust in peers as well as sources of peers. Mesh connectivity can be bootstrapped by manually approving only trusted friends, automatically importing peers by piggy-backing on existing social networks, and/or via a random set of untrusted users provided by a matching service. Restricting sharing to only trusted contacts provides few avenues of attack for would-be monitoring agents, but can be an obstacle to early adopters who by definition have no one in the system they trust. Alternatively, untrusted peers improve performance and availability by increasing redundancy, but widen the attack surface.

While stronger restrictions on user behavior permit stronger statements of system-wide security properties, our deployment experience has been that support for divergent, individual notions of privacy is essential for adoption. In the year since its release, OneSwarm has been downloaded hundreds of thousands of times, translated to more than half a dozen languages, and is in active daily use by thousands of people world-wide. The feedback and behavior of this community has guided the evolution of the protocol, driving our design towards increased user control, while nevertheless retaining resistance to systematic, third-party monitoring.

In addition to its qualitative impact on design, OneSwarm's user community serves as the basis for our evaluation. We report measurements of data transfers between instrumented clients running on PlanetLab communicating through the OneSwarm mesh. Despite the overhead of providing privacy, we find that OneSwarm's performance is competitive with unanonymized BitTorrent. Furthermore, our novel lookup and transfer techniques yield a median factor of five improvement in large file download times relative to Tor and a median factor of twelve improvement relative to Freenet.

Measurements of some system properties of OneSwarm are limited by our need to protect the privacy of our users. To gain insight into the behavior of our system, we complement our Planet-

SIGCOMM'10, August 30–September 3, 2010, New Delhi, India.
Copyright 2010 ACM 978-1-4503-0201-2/10/08 ...$10.00.

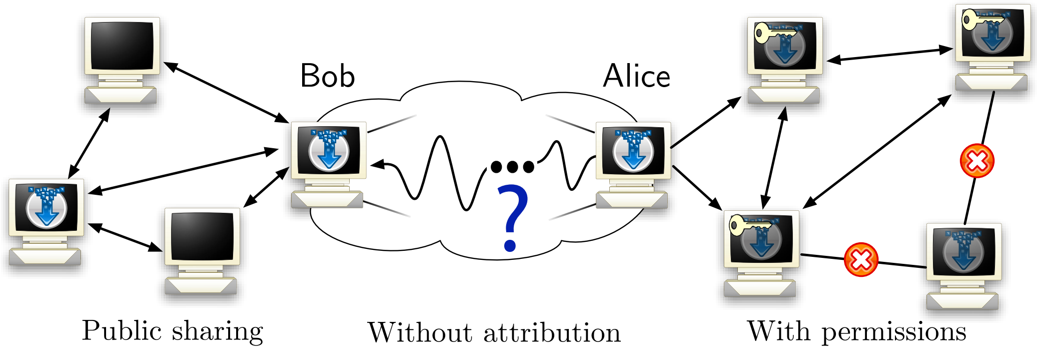

Figure 1: An example of the range of data sharing scenarios supported by OneSwarm. Bob downloads public data using OneSwarm's backwards compatibility with existing BitTorrent implementations, and makes the downloaded file available to other OneSwarm users. Alice downloads the file from Bob without attribution 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.

Lab study with simulations of OneSwarm. For this, we use a trace of the object sharing patterns and social connectivity of more than 1 million users of last.fm, a popular music-focused website that aggregates the playback histories and social network of its users [2]. Trace replay shows that OneSwarm provides high availability, with 95% of satisfiable requests being fulfilled by the overlay during peak load.

The remainder of this paper is organized as follows. Section 2 outlines the OneSwarm data sharing and workload model. OneSwarm's protocol is described in Section 3. We conduct a security analysis in Section 4 before evaluating the performance of our system in Section 5. We discuss related work in Section 6 and conclude in Section 7.

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. While some may believe that the need for privacy is the uncommon case, even something as innocuous as using BitTorrent to download a Linux security patch has privacy implications, as it exposes that the user's machine is currently vulnerable to a known exploit.

Supporting the diversity of sharing scenarios common on the Internet today precludes a system-wide definition of privacy in our design. Even for a single data object, different users may have different notions of what constitutes private information. And different peers may be more or less trustworthy with a particular bit of information: users may trust their friends more than an anonymous peer found through a rendezvous service, or they may distrust everyone equally.

One could design separate systems for each usage model, e.g., one for anonymous publication (Freenet [12]), another for anonymous download (Tor [14]), 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: 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.

Figure 1 provides an example illustrating the range of privacy preserving options supported by OneSwarm. In this case, 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 [10], 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 protect her 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 capacity. 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,

[1] Of course, the 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.

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 [12], allow data sharing without attribution, but require participants to act as a cache for the (potentially objectionable) content shared by others. A similar problem exists in Tor [14], wherein potentially malicious traffic is attributable to the exit node of an onion route, creating a severe disincentive to host a node. We consider these and other related systems in more detail in Section 6.

3. PROTOCOL DESIGN

3.1 Overview

In this section, we describe the OneSwarm protocol. Table 1 provides a road map. We first provide an overview before describing each mechanism in detail; we defer a detailed security analysis to the next section. Broadly, the protocol supports two tasks: 1) defining and maintaining the overlay topology and 2) locating and transferring data objects. A key design insight is that good P2P data sharing performance results from being able to optimize over multiple options for each data transfer. Thus we explicit designed OneSwarm to make it easy for users to configure a rich peering topology and then to use that topology efficiently for each transfer.

Topology: OneSwarm users define overlay links by exchanging public keys, which identify nodes in the mesh and bootstrap authenticated and encrypted direct connections between peers in the underlying IP network. Thus, hassle-free key distribution is essential for usability, and OneSwarm uses social graph import and community server mechanisms to make key distribution straightforward for users. A distributed hash table (DHT) serves as a name resolution service; each client maintains encrypted entries advertising their IP address and port to authorized peers.

OneSwarm peers are either *trusted* or *untrusted*.[2] Trusted peers reflect real-world relationships, e.g., friends and family, and object permissions are defined in terms of access control lists of trusted identities. Untrusted peers are used only for data sharing without attribution, serving to bootstrap mesh connectivity for users with few trusted friends.

Supporting a mix of trusted and potentially untrusted peers provides greater performance than using only trusted peers and enhances privacy relative to using only untrusted peers. Moreover, our experience has shown it to be a practical necessity for user adoption. Our initial implementation assumed mutual pairwise trust among directly connected peers in order to simply our protocol and security analysis. But, this restriction was widely criticized (or ignored) by many early adopters, leading us to a design supporting variable trust in peers. Untrusted peers are treated differently by the protocol; the timing and delivery of messages are randomized to frustrate statistical attacks.

Transport: The mesh defined by the web of trust among users is used to locate 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 adaptations to enhance privacy. First, instead of sharing all data publicly with distinct and dynamic sets of peers, each OneSwarm client restricts direct communication to a small number of persistent contacts, which provide indirect connectivity to the rest of the mesh. 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 search path in the mesh, obscuring the identities of sender and receiver when sharing data without attribution.

Flooding lookup and indirect transfers increase the overhead of OneSwarm relative to existing protocols, potentially creating capacity constraints and/or bottlenecks. To cope with this, OneSwarm's search and data forwarding protocols are congestion-aware, automatically routing around overloaded intermediaries and allowing such nodes to shed load at will. To provide high performance in the face of overloaded or slow paths, OneSwarm transfers use multiple paths to each data source. To incentivize users to contribute capacity, each OneSwarm client maintains a history of traffic volumes provided by its peers, using this information to prioritize service during periods of congestion.

3.2 Linking peers with trust relationships

Each OneSwarm user generates a 1024 bit RSA public/private key pair when installing the client, with the public key serving as its identity among its peers. OneSwarm identities are persistent, allowing two users that have exchanged keys to locate and connect to one another whenever both are online even though their IP addresses might change. In existing social-sharing P2P designs [12, 31], key exchange is typically manual. We view manual exchange as a hindrance to adoption and include multiple methods for users to more easily distribute keys.

Between two OneSwarm users that share a real-world trust relationship, OneSwarm automates key exchange in three ways. First, as in UIA [16], the OneSwarm client discovers and exchanges keys with other OneSwarm users over the local area network. Second, we piggy-back on existing social networks, e.g., Google Talk, to distribute public keys automatically among friends. Third, users can email invitations. 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 overlay links. 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 allows for fine-grained control, but in many circumstances explicitly authorizing every peer relationship is cumbersome and unnecessary. Further, OneSwarm is frequently used by communities of users with dynamic membership but mutual pairwise trust, e.g., a group of colleagues at the same institution. In such cases, users can benefit from an automated service that provides 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

[2] In practice, trust can be defined on both a per-object and per-peer basis. We discuss trust at the granularity of peers for simplicity.

	Mechanism	Description	Purpose	Sec.
Topology	Social import	Automatic key exchange via email invitations, LAN discovery, or existing social network services.	Bootstrapping trusted mesh links	3.2
	Community servers	A publish/subscribe coordination server for clients. Used to bootstrap new users with a random set of peers or to manage group membership for private communities.	Bootstrapping untrusted links, group management	3.3
	Distributed hash table	Encrypted key/value storage service maintained by clients to map identities to current IP addresses and ports.	Name resolution for mesh IDs	3.4
Data transport	Congestion-aware search	Controlled flooding of search queries to locate data and construct forwarding paths without overwhelming the network or exposing endpoints.	Locating objects, discovering paths, avoiding hotspots	3.5
	Swarming data transport	Data is split into blocks, with active downloaders redistributing completed blocks. Transfers use multiple paths and multiple sources, if available.	Load balancing, efficiency	3.6
	Long-term history	Each client maintains transfer volumes for each peer, using these to prioritize service during periods of congestion.	Resource allocation, contribution incentives	3.7

Table 1: A summary of protocol mechanisms used by OneSwarm.

subscribers with a current set of public keys via a secure web connection. In effect, subscribers to a given community server delegate trust regarding a subset of their peers to the operator, who vets prospective members. These *private* community servers mediate key exchange among users with existing trust relationships.

In contrast with private community servers, *public* community servers have open membership, allowing new OneSwarm users to easily obtain a set of untrusted peers. Bootstrapping early adopters is a significant challenge for overlay networks based on pairwise trust. 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 only for this type of sharing and serve to bootstrap overlay connectivity for users with few trusted friends.

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, consistent hashing limits the information obtained from repeated membership queries, and each connection is established only when both nodes have obtained the identity and the location of the other node from the community server.[3] Although an attacker with significant resources can evade these restrictions by creating many Sybil identities from distinct IPs, 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 4.

3.4 Identity and connectivity

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.

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. Inserting connectivity information individually for each peer enables fine-grained control over network address information. A simple alternative is indexing connectivity information by the public key of P alone. But, in that case, any user that learned P's public key could monitor P's IP location as long as P maintained its identity. By encrypting updates and publishing connectivity information for each peer individually, P can control and revoke each peer's access to its IP location updates.

In our implementation, ID → {IP, Port} mappings are stored in a Kademlia-based DHT using twenty-fold replication for fault tolerance [23]. This level of replication has been shown to provide high availability for DHTs running on end-hosts [15]. 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.

Taken together, OneSwarm's various key exchange mechanisms and DHT are the basis for creating and maintaining the overlay mesh. We next turn to the protocol details of naming, searching for, and transferring data.

3.5 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, and other meta-data for files for which a particular peer has permissions. (The node sends an empty list to each untrusted peer, or if it has nothing to share with a specific peer.) For each privately shared file the meta-data includes a 256-bit key that is used as a symmetric encryption key for use during transfers.

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 as an ID in search messages; these messages 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 the key used for decryption. We describe transfer negotiation via search since this subsumes the other cases.

Congestion aware search: OneSwarm search is designed to manage the tradeoff between overhead and performance by being *congestion aware*. Using the shortest path minimizes overhead, but

[3] An alternate approach would be to obtain a random set of peers from a DHT, but a significant limitation is that current DHTs are not robust to Sybil creation from a single IP.

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. By maintaining a set of rotating Bloom filters, an hour of search history can be remembered space-efficiently, ruling out the possibility of searches living forever in the overlay.

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. We explore the privacy implications of this in Section 4.4.

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 the search, once enough data sources have 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. Previous studies have shown that most searches in P2P network are for files with many replicas [11] and as a result these popular searches will be canceled quickly, reducing overhead.[4] Our design trades global reachability for lower overhead. Specifically our design does not guarantee that each object can be found by all nodes at all times. During times of congestion searches for objects that lack nearby replicas are likely to fail. An evaluation of the efficiency of the search algorithm is provided in Section 5.2.

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 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.

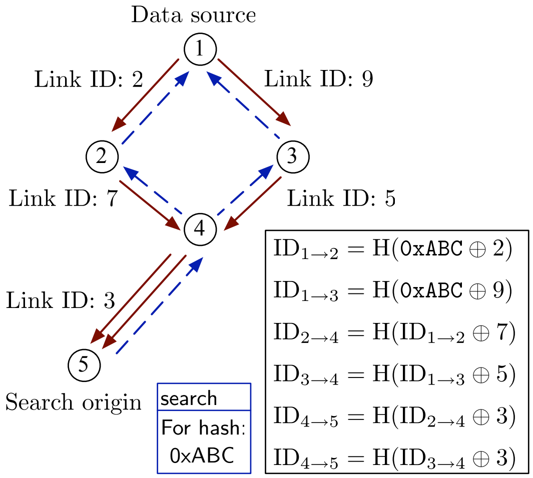
Data source
1
Link ID: 2
Link ID: 9
2
3
Link ID: 7
Link ID: 5
4
Link ID: 3
5
Search origin
search
For hash: 0xABC
$\mathrm{ID}_{1\to2} = \mathrm{H}(\texttt{0xABC} \oplus 2)$
$\mathrm{ID}_{1\to3} = \mathrm{H}(\texttt{0xABC} \oplus 9)$
$\mathrm{ID}_{2\to4} = \mathrm{H}(\mathrm{ID}_{1\to2} \oplus 7)$
$\mathrm{ID}_{3\to4} = \mathrm{H}(\mathrm{ID}_{1\to3} \oplus 5)$
$\mathrm{ID}_{4\to5} = \mathrm{H}(\mathrm{ID}_{2\to4} \oplus 3)$
$\mathrm{ID}_{4\to5} = \mathrm{H}(\mathrm{ID}_{3\to4} \oplus 3)$

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. In this case 2 unique paths are found.

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 multi-path and multi-source downloading. Each peer maintains a randomly chosen *link ID* for each peer link and keeps this information private.[5] The data source sets the initial value of the path ID to the lower 32 bits of the first matching file's hash. Next, the search reply is sent (to each peer who forwarded the data request) with 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 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 one path is discovered, and new searches can be launched to replace paths that fail.

3.6 Swarming 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 to transfer data [13]. But, rather than connecting directly to peers, OneSwarm tunnels BitTorrent traffic through overlay paths. Each overlay path is treated as a virtual BitTorrent 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

[4] Search overhead could also be reduced by routing queries through super-nodes or performing random walks [11], but such optimizations either impact privacy or result in transfers over long paths.

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

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.

Like the unpredictable and heterogeneous end-hosts BitTorrent is designed for, 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. We take advantage of parts of the BitTorrent protocol that allow multiple requests to be queued at the data source, effectively giving the host control over the end-to-end transfer window size. For example, if a path becomes congested, traffic will automatically be shifted to paths that do not traverse the congested link. If a forwarding node disconnects, the capacity of the data source is automatically shifted to other paths. As in BitTorrent, content integrity is protected by SHA-1 hashes of file blocks, allowing recipients to detect data corruption.

3.7 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 BitTorrent peers. This creates an incentive to contribute capacity while downloading, improving swarm performance. Persistent identities among directly connected peers provide an 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. 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 peer is 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 evaluation (Section 5).

4. 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 analyzing privacy properties in the more challenging case of data sharing without attribution.

4.1 Threat model

Our goal is to be resistant to the disclosure of user behavior to an attacker with control over a limited number of overlay nodes. Native BitTorrent is susceptible to just this attack, enabling a small number of monitoring agents to infer the behavior of tens of millions of users [33, 29]. Specifically, we assume that an attacker that can join the network with a limited number of nodes, monitor network traffic to/from its nodes, and generate, modify, and delete OneSwarm overlay messages flowing through its nodes. The attacker can record timing information about the messages it sends/receives to infer information about the behavior of the rest of the OneSwarm network, and may spawn any number of OneSwarm instances on its nodes. We do not attempt to guarantee privacy against attackers that can sniff, modify, or inject traffic on arbitrary network links, or attackers that can seize the physical hardware of OneSwarm users, e.g., law enforcement.

OneSwarm assumes that users are conservative when specifying trust in peers, as trusted peers can view files for which they have permissions. If trust is misplaced or a peer compromised, OneSwarm limits the resulting disclosure to only the trusted peers of the compromised nodes. This is in sharp contrast to private BitTorrent communities [38], wherein a single compromised member can monitor all users of the service.

4.2 Attacks and defenses

In this section, we outline several potential attacks and quantify their effectiveness using measurements of OneSwarm users in the wild. In a technical report [20], we explore a wider range of threats: associating search requests to users, identifying trusted links, impact of additional attacker capabilities, 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 public 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 snoop from 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 path properties 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 [39]. 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.

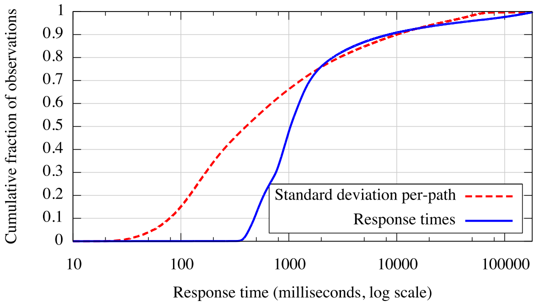

Figure 3: The distribution of search / response RTTs and the distribution of variance for RTTs on identical overlay paths with more than 10 search responses. Even for identical paths, variation in delay is significant.

- *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.

4.3 Timing attacks

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 a particular data source quite low. If the attacker has sufficient resources to connect nodes at many different points in the mesh, however, some of them might be able to infer that they are near to or directly connected to a data source based on the low RTT of response messages.

To frustrate this attack, OneSwarm artificially inflates delays for queries received from untrusted peers. Recall that attackers bootstrapped via community servers are marked as untrusted by default. All responses to untrusted peers are delayed by a random but deterministic amount (computed based on the content hash and a persistent local salt value) in order to emulate the delay profile of forwarded traffic from one or more hops away. The RTT observed by an attacker over an untrusted link is similar to that of a data source that is one or two overlay hops away and connected via low latency, trusted forwarding links. In other words, the combined use of trusted and untrusted links provides more possible explanations for a given delay profile than a design using untrusted links only.

We now consider two experiments that illustrate the uncertainty associated with inferring data proximity based on timing information. First, we measure the variability of latency and path properties in practice using our PlanetLab deployment. Next, we consider the effectiveness of this attack for the last.fm topology and workload.

PlanetLab: The feasibility of inferring behavior based on message timings depends on the length, stability, and diversity of paths to the object. Lengthy paths have greater variability due to mesh dynamics and network level effects. Similarly, the existence of a large, dynamic replica set and/or many paths confounds inference based on search response RTTs.

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 smallest RTT across all measurement nodes is the likely closest 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. Increasing the number of vantage points to 100 *reduces* back-to-back consistency to 63% as multiple attacker nodes at similar distance to the source get responses within a small enough interval that traffic conditions on the paths determines which attacker to first see the response. On the whole, it is difficult to reason about the likely direction of search response messages since the ordering of responses is highly variable.

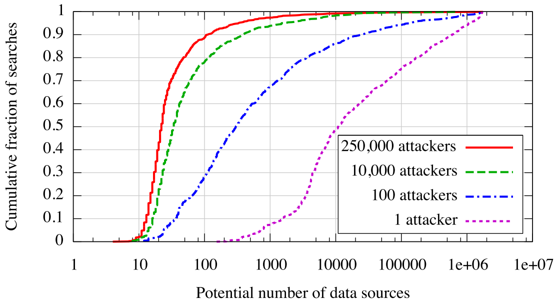

Figure 4: Using a latency and topology oracle, the number of potential data sources (x-axis) for a cumulative fraction of searches by attackers (y-axis). Even with thousands of attackers and complete topology/latency information, search response delays do not localize data sources.

The unpredictable ordering of search response messages is attributable to the naturally large variations in message delays. Figure 3 summarizes the distribution of response RTTs for more than 42 million searches, collected prior to the public release of a OneSwarm client incorporating artificial delays. 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 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).

Trace replay of last.fm: To complement our PlanetLab study, we use trace data from the last.fm music website to drive a large-scale simulation. 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, interpreting last.fm friend links as trusted links in the overlay topology. For users with less than 26 friends, we add additional untrusted links. Object download histories determine object placement and popularity, and latencies for overlay links are drawn randomly from measured latencies provided by the iPlane project [22].

We use this trace to revisit timing attacks in the idealized setting of an unloaded network and attackers with complete information regarding the overlay topology and the network delay of every link. For varying numbers of attackers bootstrapped via a community server, we simulate 1,000 searches in the last.fm topology, sampled to match the measured popularity of objects. For each

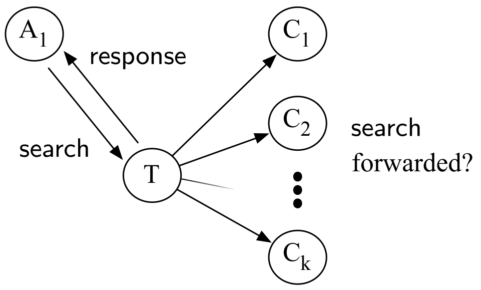

Figure 5: An attacker, A, with $C_1, ..., C_k$ colluders tests if a target T is sharing a file by sending a targeted search and observing a lack of forwarding.

search, we record the delay of the first response, and then inspect the topology and link delays to compute the number of possible data sources associated with a given delay and vantage point. Figure 4 summarizes the results. Even with complete topology and latency information as well as 250,000 vantage points, search response latencies do not localize a single data source.

4.4 Collusion attack

Next, we analyze the case of multiple peers colluding to infer whether a directly connected user is sharing a particular file. In this case, an attacker A sends a targeted search to target T, receives a search response, and observes whether the search was forwarded to colluders $C_1, ..., C_k$ who are also peers of T. (This attack is illustrated in Figure 5.) Recall that forwarding search messages is probabilistic. Each search message has a configurable probability, p_f, of 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 colluding attackers, $\Pr[\text{Not source}|\text{response received}] = (1 - p_f)^k$. With just a few colluders, an attacker can gain high confidence.

This attack requires both the attacker and colluders to be directly connected to the target. When matched randomly by a public community server, the likelihood of an individual attacker being assigned a specific target for a community server with N members is $\frac{n_c}{N}$, where n_c is the number of peers returned for a single request. 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.[6] 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 $\frac{n_c}{N}$.

In practice, the effectiveness of systematic monitoring 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), avoid public community servers completely, or request fewer peers than n_c. Figure 6 provides several concrete examples of the relationship between exposure, forwarding probability, topology, and the number of untrusted peers. In these examples, $p_f = 0.5$, and we vary n_c. Decreasing the maximum number of peers provided by a community server makes compromising its users more difficult. But, we find in our evaluation that increasing peers improves performance (Section 5).

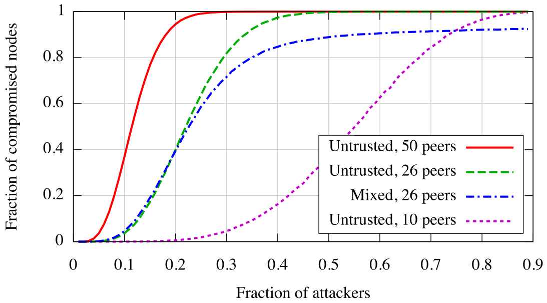

Figure 6: The cumulative fraction of nodes whose behavior can be inferred with 95% confidence (x-axis) by a given fraction of colluding attackers (y-axis). Even assuming widespread use of public community servers, a significant fraction of colluding attackers is required to infer user behavior.

Figure 6 also shows the privacy benefits associated with a mix of trusted and untrusted peers. For this case (Untrusted, 26 peers), we considered the vulnerability of clients in our last.fm trace when adopting a policy of peering with untrusted clients only when they did not have n_c or more contacts from their social network. Users with a large number of trusted friends are completely isolated from colluding attackers, shifting risk to others that are forced to more heavily rely on untrusted peers.

5. 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.

5.1 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 OneSwarm's structure, performance, and utilization in the wild. The first of these is voluntarily reported summary statistics from more than 100,000 distinct users collected over a ten month period since the public release of our software. These include the total number of peers, the method used for key exchange, and aggregate data transfer volumes.

Our second source of data is instrumented OneSwarm clients running on hundreds of PlanetLab [27] machines. Subscribing to several public community servers bootstraps connectivity for these clients, providing each with dozens of OneSwarm peers drawn randomly from the user population. Our PlanetLab nodes act as passive vantage points, measuring the the background traffic generated by users. (This includes both data forwarding and control traffic.) On average, these nodes relay more than one terabyte of data per day.

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

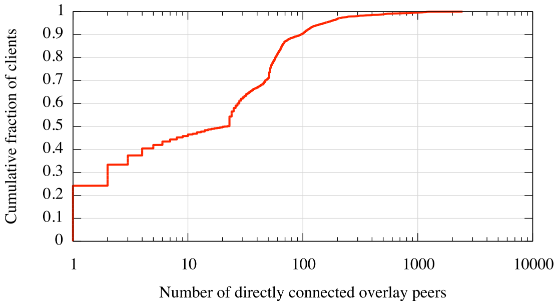

Figure 7: Cumulative distribution of peers per-client. The upper half of the bimodal shape is due to community server subscriptions. The wide range of the distribution reflects the diversity of usage behavior.

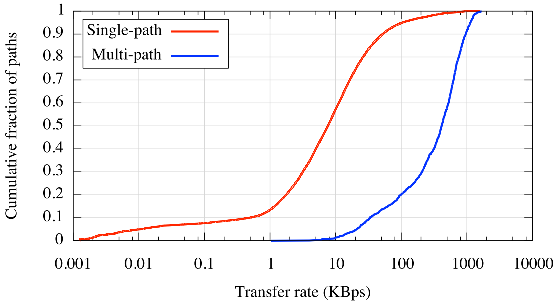

Figure 8: A comparison of single and multi-path transfer performance. Because most individual paths are slow, multi-path transfers are essential for achieving good performance.

Overlay structure: Although many overlay links in OneSwarm are based on social relationships, the graph structure is also influenced by the random matching of public community servers, as well as the tendency for some 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 7. This distribution shows significant variations in connectivity. While some users maintain hundreds or even thousands of peer connections, the median value is 22. For clients reporting data, 53% of peers are imported from community servers, 46% are entered manually, with the remaining 1% of peers coming from LAN, email invitations, or social network import.

Multi-path transfers: Unlike systems that anonymize traffic at the granularity of TCP connections, OneSwarm tolerates 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, since an individual path is limited by the bandwidth capacity of its slowest link. Given the highly skewed bandwidth capacities typical of P2P participants [28], the capacity of any single multihop path is likely to be 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. Among PlanetLab nodes, routing single path transfers over Tor yields similar results; transfer rates average 20 KBps.

Comparison with existing systems: Tor's comparable single path transfer performance suggests that simply tunneling multiple BitTorrent connections over Tor might suffice to achieve the benefits of multi-path transfers. But, in practice, we find that OneSwarm significantly outperforms Tor. To evaluate this, we compared the transfer performance of BitTorrent, BitTorrent over Tor, and OneSwarm. Tor's reliance on address translation at exit nodes precludes bidirectional connectivity and, when used by a BitTorrent client as a tunneling agent, limits the benefits of swarming data transfer by creating bottlenecks at nodes with bidirectional connectivity. To limit overhead, Tor defaults to creating new paths only once every ten minutes. We modified Tor in our experiments, instead creating new paths every ten seconds to increase the opportunity for multi-path transfers[7]. To measure the scalability of BitTorrent over Tor, we compare transfer performance when 50% of downloaders use Tor to performance when 90% of downloaders use Tor. In back-to-back trials, we used each of these methods to download a 20 MB file hosted at UW from a set of 120 PlanetLab machines. In each case, all participants joined the swarm simultaneously and remained available as sources after completion. Figure 9 summarizes the results.

OneSwarm improves both the performance and scalability of data transfer relative to Tor, which increases median download times relative to OneSwarm by a factor of 1.9 and 3.4 when used by 50% and 90% of participants, respectively. BitTorrent clients masked by Tor cannot communicate directly with one another, creating a scalability bottleneck as the fraction of Tor users increases. Downloads are effectively serialized by the limited capacity of a small number of detour nodes.

In addition to Tor, we also compared OneSwarm's transfer performance to that provided by Freenet, an anonymous P2P publishing system [12], and found that it provides performance far short of either OneSwarm or BitTorrent/Tor. In Freenet, data distribution is a two step process. First, data is published, which involves proactive caching at several points in the mesh. Afterwards, client requests are serviced from this set of replicas, with more popular files becoming more widely replicated.

As in our previous experiments, we attempted to distribute a 20 MB file from a set of Freenet nodes running on PlanetLab.[8] But, a large fraction of these transfers failed to complete, and publishing of 20 MB files often failed. Reducing the file size to 5 MB improved robustness, allowing us to compare Freenet and OneSwarm on the basis of transfer rate rather than completion time. For our PlanetLab nodes, Freenet's median transfer rate was just 17 KBps, compared to a median 118 KBps achieved by OneSwarm. This rate does not include publishing time, which would further reduce Freenet's effective distribution rate.

Overhead: Despite performance improvements compared to Tor and Freenet, the results of Figure 9 suggest that OneSwarm incurs a performance penalty relative to BitTorrent. We attribute this difference largely to the resource constraints typical of PlanetLab nodes rather than a fundamental performance property of OneSwarm. OneSwarm transfers are encrypted while BitTorrent transfers are not, and the OneSwarm transfer rate for (oversubscribed) PlanetLab nodes is often limited by the available CPU rather than their network capacity.

To verify this, and to directly measure the influence of multi-hop forwarding on transfer performance, we compare the performance of OneSwarm transfers 1) when mediated by the overlay and

[7] This configuration provides nearly a factor of 2 performance improvement relative to the default, but aggressive path creation is discouraged as it increases CPU load on intermediate routers.

[8] We allowed these nodes to operate for several hours before our experiments in order to quiesce in Freenet's mesh.

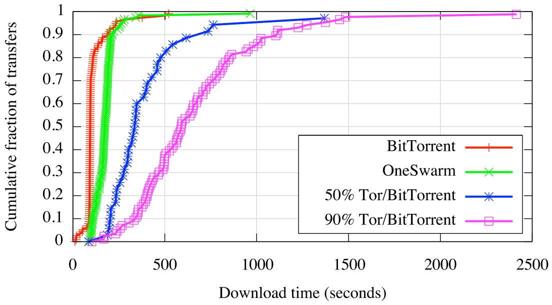

Figure 9: Transfer performance of OneSwarm, BitTorrent, and BitTorrent/Tor on PlanetLab. OneSwarm significantly outperforms existing anonymization systems and is performance competitive with BitTorrent.

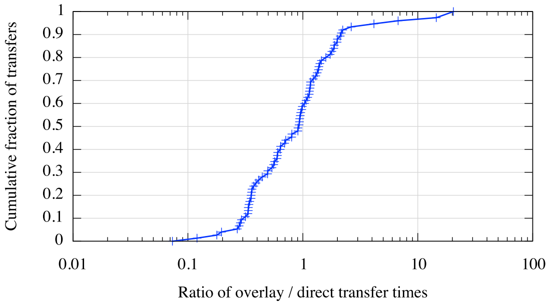

Figure 10: Comparing transfer times mediated by the OneSwarm overlay to direct transfer. Averaged over many trials, overlay transfers are performance competitive with direct point-to-point transfers.

2) when using a direct point-to-point connection between sender and receiver. (In both cases, transfers are encrypted, providing an apples-to-apples comparison.) If the overlay is not capacity constrained, we would expect both direct and overlay transfers to have a similar duration, on average, and we do find this to be the case for transfers conducted between PlanetLab nodes.

Figure 10 summarizes the ratio of the overlay and direct OneSwarm transfer times between PlanetLab nodes. We measured transfer times between pairs of 20 PlanetLab nodes while all other PlanetLab clients were disabled; i.e., the overlay did not benefit from any forwarding capacity beyond that of its existing user base. 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 challenging case for OneSwarm as PlanetLab nodes are generally of higher capacity than typical OneSwarm peers, which are often hosted from ordinary home broadband connections. Even without the addition of PlanetLab forwarding capacity, overlay transfer does not impose a performance bottleneck in most cases. Some transfers are faster, and some transfers slower, but the median ratio of overlay and direct transfer times, 0.94, suggests that overlay forwarding is not a fundamental performance bottleneck.

Next, we repeated the transfer measurements, restricting the number of peers connected to each PlanetLab node to a randomly chosen value between 1 and 35. Performance increases with the number of connected peers. For example, increasing the number of connected peers from 17 to 29 doubles median transfer performance. But, returns are diminishing; a further increase to 35 peers improves median performance by just 1%. Our default value for the maximum number of peers provided by a community server—26—reflects the tradeoff between client performance and resistance to systematic monitoring (Section 4). Of course, this value is a configurable parameter.

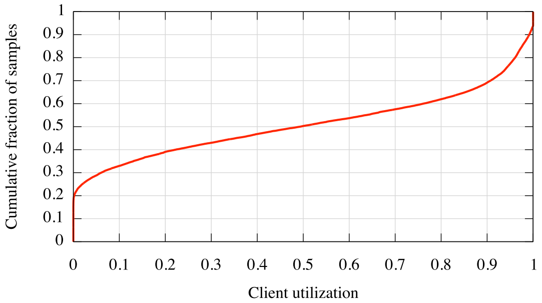

Figure 11: The distribution of client upload capacity utilizations over the course of one day. Although most clients have excess capacity, transient congestion occurs at many nodes.

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 11. 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.

5.2 Trace replay in the last.fm social graph

As in Section 4.3, we complement our evaluation of OneSwarm on PlanetLab with trace-driven simulation of its operation when applied to our measured last.fm workload. While last.fm is focused on music, making it slightly different than OneSwarm where any data can be shared, we chose last.fm as it the only service we know that publishes both the social network and the media consumption of its users. This data allows us to parameterize the simulator with real values for both user interest and social network.

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 downloading a file. Unlike Section 4.3, we do not assign clients peers from community servers, and so our availability results should be taken as a lower bound (additional paths would increase redundancy), and our overhead results an upper bound (random shortcuts would lead to shorter paths and reduced search propagation).

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 [19]. Each unique file request made by a user is interpreted as an object request in the overlay network. Each user starts as a replica for files that the user had downloaded during the first week of our trace, and we begin the trace playback at the outset of the second week.

Object availability: A simple metric that distills the feasibility of overlay forwarding is the fraction of object requests satisfied; i.e., those that discover at least one replica in the overlay. During trace

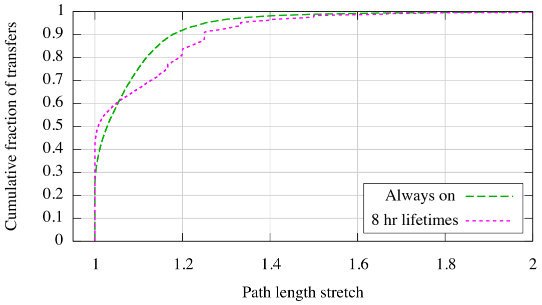

Figure 12: Path length stretch. For the last.fm workload, the majority of transfers use shortest paths. As data volume increases, capacity constraints induce stretch.

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 with a smaller number of users, 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 file being requested was downloaded 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 files downloaded 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.

Overhead: OneSwarm discovers paths to replicas by flooding search messages among friends. Although the majority of data transferred is due to popular objects, the majority of control traffic stems from requests for unpopular object for which search messages are forwarded to a large number of nodes in the overlay (during periods of low contention). This is an explicit design choice to improve availability without compromising privacy (as discussed in Section 3.5).

We compute search overhead as the fraction of control messages making up overall traffic. For the last.fm workload with always on lifetimes and average object size of 20 MB, the overhead is 6% of total data traffic. 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 43%. Although large both fractionally and by total volume, this measurement corresponds to a conservative upper-bound as: 1) BitTorrent downloads are typically larger (about 400 MB [1]), and 2) the connectivity for the last.fm social network is sparser than that of our deployment (median degree of 3 vs. 22). Further, 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.

We emphasize that control overhead is large only in the presence of excess capacity; congested clients drop searches. Since OneSwarm is over-provisioned in practice as well as in our simulations, overhead is high because search coverage is high. This promotes availability. Yet, because the mesh is not capacity constrained, overhead does not degrade performance.

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 equivalent 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 overlay 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 12.

For the last.fm workload with always on lifetimes, path diversity is high. In this case, OneSwarm uses shortest paths for 48% of transfers with an average path length from source to replica of 4.8 overlay hops. 92% of objects have a stretch ≤ 1.2. Path diversity is reduced when lifetimes decrease to 8 hours; this increases stretch. In both cases, a small fraction of requests traverse paths with frequent contention, increasing stretch.

6. 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 [10]. Anonymizer provides anonymization services commercially, providing a centralized service that relays web traffic [3]. Crowds [32] provides anonymous web browsing by randomly tunneling requests via other system participants. Herbivore [34] enables anonymous file-sharing by providing a more scalable implementation of DC-nets [9]. 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 [14] 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 [21]. Tarzan uses similar address rewriting techniques in a P2P context [17]. 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. A similar challenge is inherent in BitBlender [7], which attempts to mask deliberate participants in BitTorrent swarms by including a number of randomly selected “blender" nodes in the distribution as well. More recently, Baden et al. have applied cryptographic techniques to enable data sharing with permissions in current social web services without exposing content to service providers [6]. OneSwarm supports permissions in addition to allowing users to share data publicly without attribution.

Broadly, OneSwarm differs from all these systems in its support for a spectrum of data-sharing models and peer trust relationships, as well as an evaluation grounded in a large-scale deployment and user population.

Trust: Incorporating real-world trust relationships has been a crucial design element in several recently proposed systems. SybilGuard [37] uses properties of social networks to ferret out synthetic identities in social systems. Friendstore [35] is a P2P backup system where users store backup data only on other trusted nodes owned by friends or colleagues. In Ostra [26], the scarcity of social connections is used to combat spam. UIA [16] provides data routing and name resolution over a socially constructed overlay of personal devices. Turtle [31] is a file-sharing application that limits direct communication to only the social graph in an attempt to circumvent eavesdropping.

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.

Workload: Our measurements and analysis of the last.fm workload are largely consistent with existing work that characterizes sharing in P2P networks [4, 18, 28] and usage of popular content sharing sites [8]. Independent measurement efforts have shed light on the properties of popular online social networks [5, 24, 25]. 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.

7. CONCLUSION

Although widely used, currently popular P2P file sharing networks expose users to silent, third party monitoring of their behavior. 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 Windows, Mac OS X, and Linux, 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 with OneSwarm are performance competitive with BitTorrent and substantially outperform existing anonymization networks.

8. REFERENCES

[1] BitTorrent Statistics. *http://www.slyck.com/story370_BitTorrent_Statistics*.

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

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

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

[5] 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.

[6] R. Baden, A. Bender, N. Spring, B. Bhattacharjee, and D. Starin. Persona: An online social network with user-defined privacy. In *Proc. of SIGCOMM*, 2009.

[7] K. Bauer, D. McCoy, D. Grunwald, and D. Sicker. BitBlender: Light-weight anonymity for BitTorrent. *SecureComm*, 2008.

[8] 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 *IMC*, 2007.

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

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

[11] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker. Making Gnutella-like P2P Systems Scalable. In *Proc. of SIGCOMM*, 2003.

[12] 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.

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

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

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

[16] 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.

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

[18] 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 *SOSP*, 2003.

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

[20] T. Isdal, M. Piatek, A. Krishnamurthy, and T. Anderson. Privacy-preserving data sharing with OneSwarm. Technical report, Dept. of CSE, University of Washington, 2010.

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

[22] H. V. Madhyastha, T. Isdal, M. Piatek, C. Dixon, T. Anderson, A. Krishnamurthy, and A. Venkataramani. iPlane: An Information Plane for Distributed Services. In *OSDI*, 2006.

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

[24] 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.

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

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

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

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

[29] M. Piatek, T. Isdal, A. Krishnamurthy, and T. Anderson. One hop reputations for peer to peer file sharing workloads. In *Proc. of NSDI*, 2008.

[30] M. Piatek, T. Kohno, and A. Krishnamurthy. Challenges and directions for monitoring P2P file sharing networks –or– Why my printer received a DMCA takedown notice. In *Proc. of HotSec*, 2008.

[31] 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 Sec. Prot.*, 2004.

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

[33] G. Siganos, J. M. Pujol, and P. Rodriguez. Monitoring the Bittorrent Monitors: A Birds Eye View. In *PAM*, 2009.

[34] 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.

[35] D. N. Tran, F. Chiang, and J. Li. Friendstore: cooperative online backup using trusted nodes. In *Proc. of WSNS*, 2008.

[36] 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.

[37] H. Yu, M. Kaminsky, P. B. Gibbons, and A. Flaxman. Making gnutella-like p2p systems scalable. In *Proc. of SIGCOMM*, 2006.

[38] C. Zhang, P. Dhungel, D. Wu, Z. Liu, and K. W. Ross. BitTorrent Darknets. In *Proc. of INFOCOM*, 2010.

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