NORTHWESTERN
UNIVERSITY

Electrical Engineering and Computer Science Department

Technical Report
NWU-EECS-09-04
March 9, 2009

SwarmScreen: Privacy Through Plausible Deniability in P2P Systems

David R. Choffnes, Jordi Duch, Dean Malmgren, Roger Guiermà, Fabián E. Bustamante and Luis Amaral

Abstract

Peer-to-peer (P2P) systems enable a wide range of new and important Internet applications that can provide low-cost, high-performance and resilient services. While a strength of the P2P paradigm is the ability to take advantage of large numbers of connections among diverse hosts,each of these connections provides an opportunity for eavesdropping on sensitive data. A number of efforts attempt to conceal connection data with private, trusted networks and encryption; however, the mere existence of a connection is sufficient to reveal information about user activity. Using only the connection patterns gathered during a one-month period (comprising a stable population of10,000 BitTorrent users), we extract communities of users that share interest in the same content. Despite the fact that connections in BitTorrent require not only shared interest in content, but also concurrent sessions, we find that strong communities of users naturally form – our analysis reveals that users inside the typical community are 5 to 25 times more likely to connect to each other than with users outside. These strong communities enable a guilt-by-association attack, where an entire community of users can be classified by monitoring one of its members. Our study shows that through a single observation point, an attacker trying to identify such communities can uncover 50% of the network within a distance of two hops.

To address this issue, we propose a new privacy-preserving layer for P2P systems that disrupts community identification by obfuscating users' network behavior. We show that a user can achieve plausible deniability by simply adding a small percent (between 25 and 50%)of additional random connections that are statistically indistinguishable from natural ones. Unlike connections in anonymizing networks, these random connections have the benefit of adding available bandwidth to the related swarms. Because our solution is protocol compliant and incrementally deployable, we have made it available as an extension to a popular BitTorrent client.

Keywords: Privacy, P2P, BitTorrent, guilt-by-association attack, community detection

SwarmScreen: Privacy Through Plausible Deniability in P2P Systems

David R. Choffnes[†], Jordi Duch[*], Dean Malmgren[*], Roger Guiermà[*], Fabián E. Bustamante[†], Luis Amaral[*]

[†]EECS, Northwestern University
{drchoffnes,fabianb}@cs.northwestern.edu

[*]Chemical Engineering, Northwestern University
{jordi.duch,r-malmgren,rguimera,amaral}
@northwestern.edu

Abstract

Peer-to-peer (P2P) systems enable a wide range of new and important Internet applications that can provide low-cost, high-performance and resilient services. While a strength of the P2P paradigm is the ability to take advantage of large numbers of connections among diverse hosts, each of these connections provides an opportunity for eavesdropping on sensitive data. A number of efforts attempt to conceal connection data with private, trusted networks and encryption; however, the mere existence of a connection is sufficient to reveal information about user activity. Using only the connection patterns gathered during a one-month period (comprising a stable population of 10,000 BitTorrent users), we extract communities of users that share interest in the same content. Despite the fact that connections in BitTorrent require not only shared interest in content, but also concurrent sessions, we find that strong communities of users naturally form – our analysis reveals that users inside the typical community are 5 to 25 times more likely to connect to each other than with users outside. These strong communities enable a guilt-by-association attack, where an entire community of users can be classified by monitoring one of its members. Our study shows that through a single observation point, an attacker trying to identify such communities can uncover 50% of the network within a distance of two hops.

To address this issue, we propose a new privacy-preserving layer for P2P systems that disrupts community identification by obfuscating users' network behavior. We show that a user can achieve plausible deniability by simply adding a small percent (between 25 and 50%) of additional random connections that are statistically indistinguishable from natural ones. Unlike connections in anonymizing networks, these random connections have the benefit of adding available bandwidth to the related swarms. Because our solution is protocol compliant and incrementally deployable, we have made it available as an extension to a popular BitTorrent client.

1 Introduction

P2P computing has enabled a wide range of new and important Internet applications ranging from large-scale data distribution to video streaming and telephony. The approach provides scalability, reliability and high performance by taking advantage of a large number of cooperative, interconnected hosts.

While much of the strength of the P2P model lies in large numbers of interconnected nodes, their connections offer multiple opportunities for eavesdropping. With P2P networks increasingly under surveillance from private and government organizations [11, 25, 29] and subject to political censorship [13, 22], there is an urgent need for privacy-enhancing systems that are both effective and practical. A number of efforts attempt to conceal connection data with private, trusted networks and variable levels of encryption. Although effective at restricting access to the content exchanged over a given connection, these approaches leave the existence of the connection itself visible. In this paper, we show that these connections erode user privacy in a way that is ignored by most distributed systems and transparent to end users.

This work focuses on the BitTorrent file-sharing network where peers connect solely on the basis of common and concurrent interest in the same content, rather than on friendship [9], common language [30] or geographic proximity [2]. Using connection patterns gathered during a one-month period (comprising a stable population of 10,000 BitTorrent end-users), we investigate the existence of communities – collections of peers significantly more likely to connect to each other than to a randomly selected peer. We show that strong communities form naturally in BitTorrent, with users inside a typical community being 5 to 25 times more likely to connect to each other than with outside users.

Historically, this ability to classify users has been abused by third parties in ways that violate individual privacy. The existence of strong communities enables a guilt-by-association attack, where an entire community of users can be classified by monitoring one of its members. Our study demonstrates that, through a single observation point, an attacker trying to identify such communities can reveal 50% of the network using only knowledge about a peer's neighbors and their neighbors (i.e., peers within two hops of the attacker). Further, an attacker monitoring only 1% of the network can correctly assign users to their communities of interest more than 86% of the time.

To address this threat, we propose a new privacy-preserving layer for P2P systems that obfuscates user-generated network behavior. We show that a user can achieve plausible deniability by simply adding a small percent (between 25 and 50%) of additional random connections that are statistically indistinguishable from natural ones. Based on this result, we design a system that generates such connections by participating in randomly selected torrents. We describe how our implementation for a popular BitTorrent client protects against classification while enabling users to specify how to balance the goals of privacy and performance.

Our work makes the following four key contributions. We present the first characterization of communities among BitTorrent users. Using only connection patterns between real users in the BitTorrent network, we find surprisingly strong and distinct communities of peers that regularly connect with each other. While this behavior has been observed in social networks, we are the first to find it in the context of BitTorrent. Second, we introduce a new privacy threat model. We show that an attacker can use information about communities to efficiently classify and effectively monitor users that share interest in the same content. Third, we propose and analyze a defense strategy against this threat that provides privacy through obfuscation. Our approach disrupts attempts to classify user behavior by inducing connections to random torrents that mask those requested by the user. Fourth, we implement our defense strategy and make it publicly available. Since our solution is protocol compliant, we make an implementation available as a pluggable extension to a popular BitTorrent client.

The remainder of the paper is structured as follows. Section 2 describes how to identify communities of users based on BitTorrent connection information. We show that these communities of shared interest can be exploited in a guilt-by-association attack in Sec. 3. To mitigate this threat, Sec. 4 discusses an approach that weakens and disrupts community analysis by generating random connections. In Sec. 5 we present the design of a system to implement this strategy. We describe and evaluate our implementation of the approach in Sec. 6. Finally, we cover related work in Sec. 7, discuss open issues in Sec. 8 and conclude in Sec. 9.

2 Communities in BitTorrent

In this section, we describe our dataset, which contains connection information for 10,288 peers during a one-month period. We use this information to form a graph and analyze its properties in terms of modularity – i.e., whether there are distinct communities in which users connect to each other more often than to users outside the community. Despite the fact that the BitTorrent protocol relies on establishing connections at random, we find strong communities of shared interest in content.

2.1 Dataset

The data used in this study is collected from BitTorrent end-users during the month of March, 2008 (31 days). Our dataset contains information about P2P connections from each monitored host, and the time during which those connections were active. Based on the BitTorrent protocol, a connection between users indicates that they share interest in some content; however, we do *not* record any information that identifies the particular content.

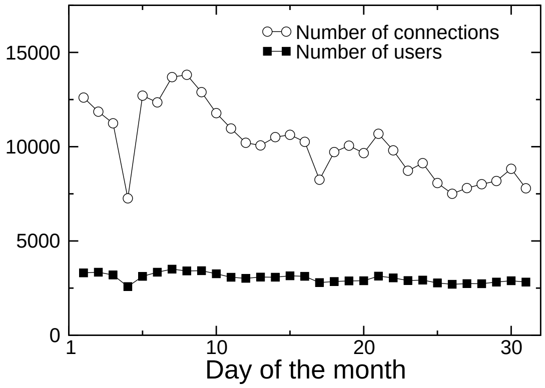


Figure 1: Number of users and connections present in each day of our dataset. With over 10,000 total users the period, we see on average 3,000 users online and 10,000 connections per day.

We restrict our analysis to a stable set of peers during the measurement period. Specifically, we filter our dataset to contain data only for hosts that have appeared in our records before March 1, 2008 and after March 31, 2008 (based on data recorded from July 15, 2007 until December 11, 2008). We are left with 10,288 users, and an average of 3,029 users online and an average of 10,162 connections between these users per day.

From this dataset, we create graphs of the connections between peers. Namely, for each day of the month we generate a graph, where each node is a peer that is online during that day and each edge indicates that there was at least one connection established between the corresponding peers during that day. To avoid issues with users that connect at regular intervals (e.g., those that connect every Saturday), we aggregate the information into four weekly graphs. These graphs consist of weighted edges where the weight w_{ij} between nodes i and j indicates how many days this pair of users have repeated a connection between them. Figure 1 plots the number of peers and connections per day during the observation period.

2.2 Extracting Communities

In social networks, individuals decide with whom they want to establish connections, so communities naturally appear. These communities are usually a reflection of past or present geographical colocation, shared interests, or co-membership in organizations, and manifest themselves in the network as groups of nodes that are more densely connected to each other than we would expect by random chance [**?**, **?**].

In contrast, nodes in many P2P networks, including BitTorrent, establish connections according to a predefined protocol that selects peers at random from a pool of eligible hosts. This observation may lead one to conclude that community structure will not be significant in P2P networks. However, much as in social networks, the existence of a connection between two users in a P2P network is a reflection of shared interest — in BitTorrent, a connection between two users indicates concurrent shared interest in at least some content. In this section, we show this shared interest is sufficient to form strong communities of users in the BitTorrent network.

The problem of community detection in graphs is NP-hard, since the space of possible partitions of nodes into communities scales faster than any power of the system size [**?**]. Part of the difficulty stems from the fact that the number of communities and their sizes are, a priori, unknown, which makes the problem of community identification qualitatively different from, and more challenging than, the well-studied graph partitioning problem.

A successful approach for solving the community identification problem is based on the maximization of a quality function $\mathcal{M}$, usually called modularity [**?**]. For a given partition $\mathcal{P}$ of a weighted graph into communities, the modularity is defined as

$$\mathcal{M}(\mathcal{P}) = \frac{1}{2L} \sum_{ij} \left[w_{ij} - \frac{s_i s_j}{2L} \right] \delta_{m_i m_j} \qquad (1)$$

where the sum is over all nodes, w_{ij} is the weight of edge (i, j), s_i is the sum of the weights of all of node i's edges, $L = \sum_i s_i$, m_i is the community to which node i belongs (in partition $\mathcal{P}$), and δ_{ij} is the Kronecker symbol ($\delta_{ab} = 1$ if $a = b$ and $\delta_{ab} = 0$ otherwise).

The modularity function is a relative measure of how much edge weight falls within communities, as opposed to between communities. If there were no communities in the network, the total connection strength s_i of each node would be evenly distributed among all the other nodes, so that the weights w_{ij} would be proportional to s_i and s_j (more precisely, $s_i s_j / L$). Positive modularities thus indicate systematic deviations from the perfectly homogeneous null model, whereas the modularity is close to zero for a random partition of the nodes into communities, when all nodes are in the same community, or when each node is in a different community.

Maximizing modularity exhaustively is intractable due to the combinatorial number of possible ways in which one can partition a graph into communities. Therefore, we use a heuristic approach to efficiently explore the space of possible partitions. Specifically, we use the extremal optimization algorithm proposed by Duch et al. [**?**], which provides a compromise between accuracy and speed [**?**]. We have validated its results by comparing them with more accurate algorithms such as simulated annealing [**?**] and found they were nearly identical.

We use the extremal optimization method to investigate the community structure of the weekly graph that spans from March 22-28. Fig. 2 shows the density of peer connections that are inside and outside of their communities. We find that the density within communities is typically 5 to 25 times higher than between communities. Although suggestive, these values do not necessarily mean that the communities we identify are significant [**?**]. To address the issue of significance, we compare the maximum modularity we obtain for this network to the same for an ensemble of randomized networks in which users connect with a uniform probability to each other (preserving the number of connections of each user). We find that the real maximum modularity of the graph is $\mathcal{M} = 0.439$, whereas the average maximum modularity of the randomized networks is $\mathcal{M} = 0.168$ with a standard deviation of 0.0012. With the real modularity more than 250 standard deviations larger than the random expectation ($z > 250$), we can safely conclude that the discovered communities are significant. For comparison, the modular structure of the world-wide air transportation network (of commercial flights between cities) has $z \approx 430$, whereas the modularity of the Internet at the autonomous system level has $z \approx 80$ [**?**, **?**].

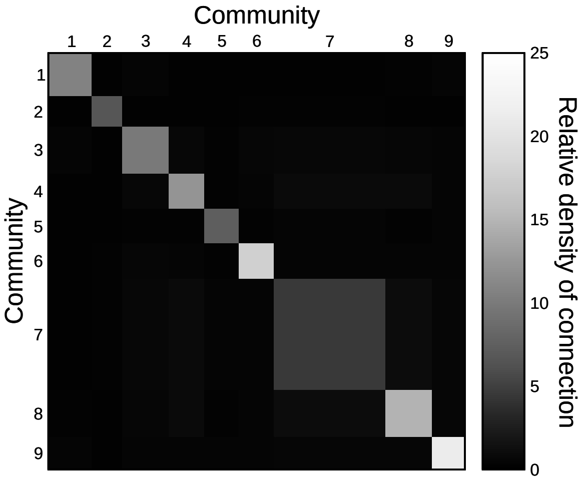


Figure 2: Density of connections within and between communities (relative to the average density of connections in the network) in the weekly network built from March 22nd to March 28th. Each row (column) corresponds to a community, and the height (width) indicates the size of the corresponding community. The density of connections within communities is typically 5 to 25 times higher than between communities.

3 Community-Based Attacks on Privacy

The BitTorrent network is already under privacy-intrusive attacks that entail using trackers and participating (rogue) clients to identify users that share particular content (e.g., to detect violation of copyrights). These attacks are limited by a number of factors, such as the need to monitor a large number of trackers, the challenge in properly identifying a large number of torrents for targeted content and the problems associated with running a large number of rogue clients. In this section, we describe an attack that eliminates many of these restrictions by exploiting the BitTorrent community structure.

As we demonstrated in previous sections, nodes in the BitTorrent network form well-defined communities of shared interest. Given this, an attacker who identifies the

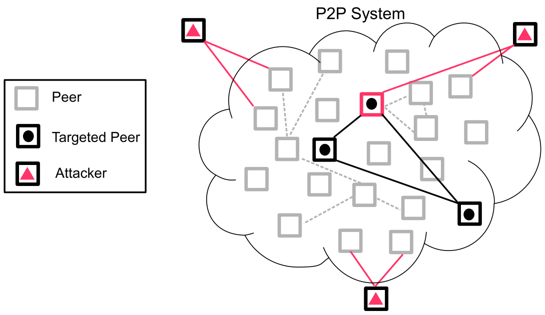


Figure 3: Diagram indicating how an attacker (or set of attackers) infiltrate a P2P system for the purpose of identifying user activity. The attacker makes connections at random, many of which are not useful, but meanwhile it collects the information about the structure of the connections of the P2P network. Once the attacker has classified a group of nodes into a community, the attacking host on the top right can classify an entire community of users by observing a single targeted peer.

content that a BitTorrent user is sharing can determine that all users in the same community are doing the same *without monitoring them directly*. We refer to this as a *guilt by association* attack – as first proposed by Cortes et al. [5] for identifying fraudulent callers in a phone network. We will show how this enables a small number of attackers to classify large numbers of peers.

To realize this attack, we assume a threat model that comprises two phases. First, the attacker attempts to discover as many connection patterns as possible, then uses this information to identify communities of users that share interests. There are several methods for discovering the structure of network connections among P2P clients. The ideal case would be a global monitoring system that is capable of tracking all the activity of every peer in the network, obtaining complete information about the system. However, in general, monitoring every peer in a popular P2P system is intractable for reasons of scale. A viable option is to use a local discovery method to uncover the structure and the patterns of connections between users. The attacker can deploy a number of participating clients to infiltrate the P2P network or sniff packets from a number of monitored hosts to observe connections. Figure 3 portrays this aspect of the threat model, where attackers participate in the P2P system via rogue clients.

In the second phase, the attacker can extract communities of shared interest for guilt by association. This can be accomplished using the same heuristic that we employed in the previous section to find modularity in the network. Because their analysis may be based on an incomplete information, we study the effectiveness of classification in this context.

3.1 Discovering user connections

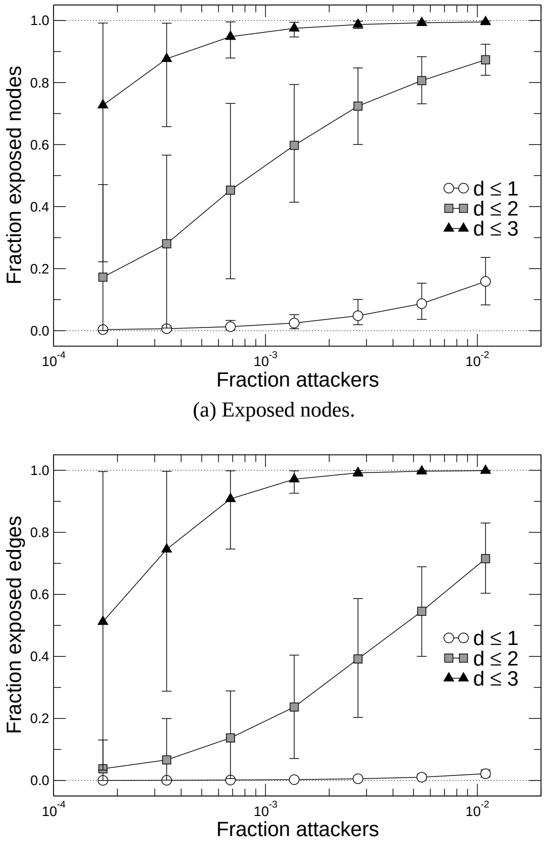
There are several methods that attackers might use to monitor the content downloading activity of BitTorrent users. For instance, they can monitor information collected by trackers, acquire sets of peers connected to their neighbors via the Peer Exchange (PEX) protocol [6] or crawl the BitTorrent DHT for lists of peers connected to a particular torrent. In the case of monitoring trackers, an attacker could essentially reveal the entire network of connections, making it trivial to determine the community structure of users. To determine the limits of the guilt-by-association attack strategy, we analyze a worst-case scenario for attackers, where an incomplete view of the connectivity patterns in BitTorrent is revealed.

To this end, we model an attacker that crawls the BitTorrent network to obtain as much connectivity information as possible. In particular, an attacker implements a breadth-first search approach to find all users within a distance d of a rogue client, as acquired through the PEX protocol. By using multiple rogue clients, the attacker should be able to increase the coverage of peer connections.

To demonstrate the effectiveness of such an attack strategy, we select $N = 1, 2, 4, 8, 16, 32, 64$ nodes uniformly at random from each of the weekly BitTorrent networks and determine the fraction of nodes that they could collectively monitor within a distance d of all attackers. We repeat this Monte Carlo sampling 100 times to obtain reliable estimates of how much information a small set of attacking nodes could gather with such an approach. Figure 4a shows the fraction of exposed nodes for a different number of attackers withing a monitoring distance d. In this case, a single attack node observes, on average, over 70% of all nodes within a distance $d \leq 3$ and a coordinated attack mounted by a small 1% of nodes observes, on average, over 80% of all nodes within a distance $d \leq 2$.

Of course, to optimize their effectiveness, attackers exploiting a breadth-first search strategy should try to connect to as many users as possible to be able to monitor, first-hand, as much of the network as possible. We demonstrate the effectiveness of highly connected attackers by selecting the $N = 1, 2, 4, 8, 16, 32, 64$ most connected nodes from each of the weekly BitTorrent networks to determine the fraction of nodes that they could collectively monitor within a distance d of all of the attackers. Figure 5a shows the fraction of exposed nodes for a different number of attackers within a monitoring distance d, in this case for highly connected attackers. Here, a single attack node can observe over 80% of the network within a distance $d \leq 2$ and almost all of the network within a distance $d \leq 3$. As the figure clearly shows, monitoring coverage becomes more extensive the larger the fraction of attack nodes.

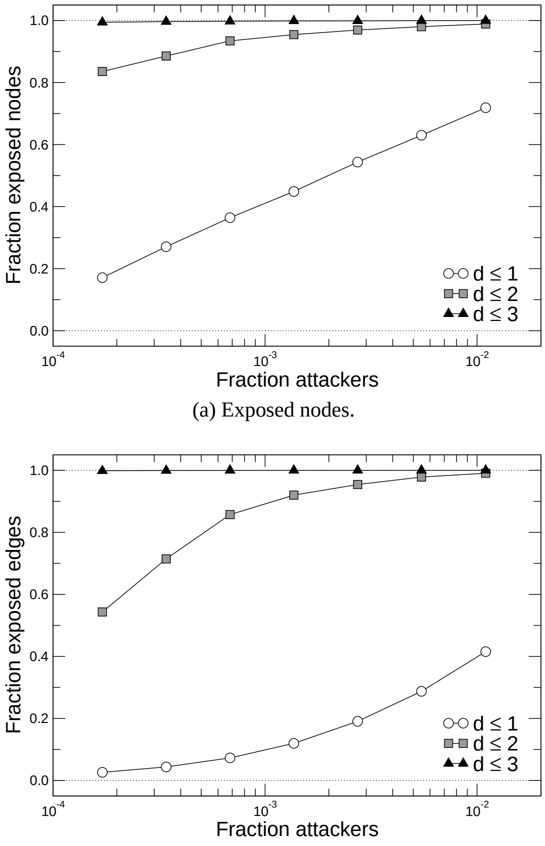
As these very simple strategies illustrate, an attacker can reveal a large portion of the BitTorrent network's connectivity patterns without centralized information. Now we examine how this incomplete information can be used to determine community structures.



(a) Exposed nodes.

(b) Exposed edges.

Figure 4: Fraction of exposed nodes and edges when a small set of $N = 1, 2, 4, 8, 16, 32, 64$ attackers can monitor all nodes and edges within a distance d over the course of a week. Attacking nodes selected uniformly at random. Symbols denote average over 100 Monte Carlo realizations and whiskers denote 95% confidence intervals.



(a) Exposed nodes.

(b) Exposed edges.

Figure 5: Fraction of exposed nodes and edges when a small set of $N = 1, 2, 4, 8, 16, 32, 64$ attackers can monitor all nodes and edges within a distance d over the course of a week. Attacking nodes are the set of N nodes with the largest degree.

3.2 Detecting community structure

In the next step of the guilt-by-association attack, the attackers attempt to identify communities of shared interests. If the attacker has access to global information about the whole network, the community detection algorithm we described in Sec. 2.2 accurately identifies communities of interest. We now address the issue of the accuracy of community detection in the presence of incomplete information.

To this end, we analyze the reliability of an attacker's inference of the community structure based on a partial reconstruction of the network. Specifically, we measure the probability p that two nodes are coclassified in the same community in the real network given that they are coclassified in the same community in the partial reconstruction.

Since the community identification method is not deterministic, however, two users are not necessarily assigned to the same community by extremal optimization. Given this variability, one must determine how to confidently assign users i and j to the same community. To address this issue, we run extremal optimization R times to obtain high-modularity partitions $\{P_1, P_2, \ldots, P_R\}$. We then assume that i and j can be confidently associated with each other if they are assigned to the same community at least τ times. We choose $R = 10$ and explore two different thresholds, $\tau = 5$ and $\tau = 8$.

Based on the analysis in the previous section, Table 1 shows p calculated for different values of the fraction f of attackers that monitor nodes and edges within a distance d for the weekly network during March 22–28. For $\tau = 8$, we find that if 0.01% of the nodes in the graph are attackers, they are able to correctly coclassify users into communities more than 85% of the time for $d \leq 3$. If 1% of the nodes are attackers, they can achieve the same accuracy for community detection by only monitoring users that are within a distance $d \leq 2$. We find similar results for attackers that use a more relaxed threshold for assigning users to the same community ($\tau = 5$): $p = 0.819$ for $f = 0.0001$ and $d \leq 3$, and $p = 0.805$ for $f = 0.01$ and $d \leq 2$.

		d		
		1	2	3
f	0.0001	0.131	0.485	0.859
	0.001	0.214	0.703	0.855
	0.01	0.343	0.864	0.902

Table 1: Similarity between the community structure of the real network and a partial reconstruction of the network discovered using a fraction f of attackers that observe to distance d. Here, we define two users as being in the same community if they are coclassified in $\tau = 8$ of $R = 10$ runs of extremal optimization. We measure the similarity by the the probability p that two nodes are coclassified in the same community in the real network given that they are coclassified in the partial reconstruction of the network.

4 Hiding in the Crowd

In the previous section, we showed that a very small fraction of attackers can easily discover the network and accurately infer the community of each user. This clearly demonstrates that guilt-by-association attacks are a real threat to the BitTorrent community and, more generally, P2P systems. The success of this attack strongly depends on the assumption that attackers can reliably infer user interests based on the connections that they have with other peers. We posit that the best defense against this attack is simply to introduce noise such that this assumption no longer holds. Specifically, our approach is to add random edges to disrupt an attacker's ability to (*i*) correctly infer real connections and thus (*ii*) infer community membership.

To determine the effectiveness of our defense strategy, we simulate adding a varying number of random edges between nodes. Since we expect that exceptionally active users will have more incentive to hide their connectivity patterns than infrequent users, we add random edges proportional to the number of edges of each user. Specifically, we add a varying percentage of random edges to the weekly graph during March 22–28, and we see how effectively an attacker can correctly infer the real community structure of the resulting graph. We quantify the effectiveness of this method with the following metrics.

First, we measure the *undetectability* of users, which we define as the probability that any two users are not detected in the same community after adding random edges, given that they are identified in the same community before adding random edges. That is, if an attacker found two users i and j classified in the same community without adding random edges, undetectability quantifies the likelihood that an attacker would correctly identify i and j in the same community after adding random edges. In Figure 6 we demonstrate that random edges increases undetectability for users. We again present the results for two different thresholds for community detection, a more restrictive one with $\tau = 8$, and a less restrictive $\tau = 5$. For $\tau = 8$ and only 10% additional random edges, an attacker would incorrectly infer that two users are in the same community more than 50% of the time. For $\tau = 5$, the same result is achieved with as few as 50% random edges.

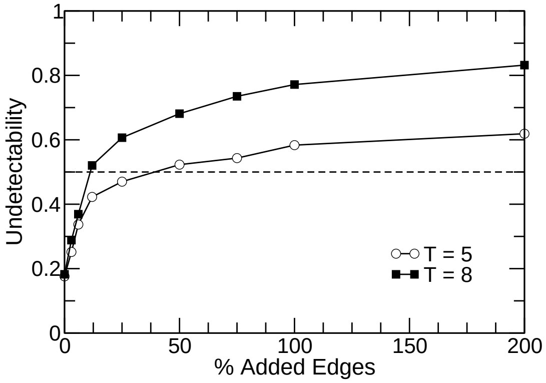


Figure 6: Undetectability of a user when varying the percentage of random edges added. This measures the probability that any two users are detected in the same community after random edges have been added, given that they were detected in the same community before doing so.

Second, we measure the *deniability* of users, i.e., the probability that any two users are not detected in the same community before adding edges, given that they are in the same community after adding random edges. That is, if an attacker found two users i and j in the same community after adding random edges, deniability quantifies the likelihood that an attacker would incorrectly determine that i and j in the same community. Figure 7 shows how deniability increases for the same two thresholds $\tau = 5$ and $\tau = 8$ as more random edges are added. For both thresholds, adding 50% additional random edges increases the deniability to 50%. Because this means that classifications made by attackers are wrong the majority of the time, this approach significantly reduces their credibility.

These results demonstrate that by adding only a few random edges (as few as $10 - 20\%$) we substantially increase the privacy of the user in two different ways. For one, we increase the difficulty of correctly associating users that share the same type of content; further, we reduce the credibility of guilt-by-association attacks.

5 System Design

In the previous sections, we showed that P2P systems, and BitTorrent specifically, are susceptible to guilt-by-association attacks and we developed a simple, yet effective strategy to defeat this threat. We now describe the design of SwarmScreen[1] – a system that realizes this strategy in BitTorrent.

At a high level, the goal of SwarmScreen is to disguise user behavior by connecting to hosts outside of the user's community of interest. To achieve this goal, our software connects to users of torrents selected automatically at ran-

[1] The software name has been changed for double-blind review.

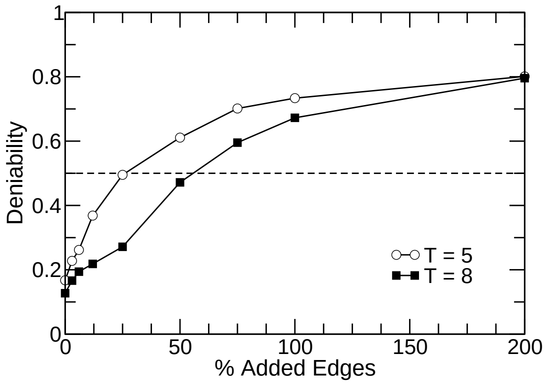


Figure 7: Deniability of a user when varying the percentage of random edges added. This is the probability that any two users are detected in the same community before adding random edges, given that they are detected in the same community afterward.

dom. While this approach can hide a user's community of interest, a simple implementation that adds a large number of random torrents will lead to poor performance for the user-generated torrents – those torrents must all share a fixed amount of bandwidth. Worse yet, such an implementation can actually *reduce* privacy by raising suspicion – a user generating an unusually large number of connections may appear to be hiding something.

SwarmScreen addresses these issues through two policies. First, our software allows the user to directly specify the desired trade-off between performance and privacy. Based on this setting, SwarmScreen provides attempts to obfuscate user-generated torrent behavior within the performance constraint. Second, instead of hiding communities of interest in a large sea of random connections, our software induces connections that mimic those from a user's community of interest. In this way, the user's cover traffic can be made statistically indistinguishable from its real traffic.

We note that SwarmScreen is designed as a new privacy layer, and is not intended as a replacement for existing approaches to privacy in P2P systems. As we discuss later in the paper, connection encryption and anonymous forwarding of HTTP requests strengthen our system against attack. In the following sections, we describe SwarmScreen's goals, architecture and design challenges.

5.1 Goals

In this section, we explain the four primary goals of the SwarmScreen design. To facilitate adoption, our system should be minimally invasive to existing clients, provide practical incentives and be easy to use and install. Further, our system should be incrementally deployable to ensure its effectiveness across a wide range of adoption rates.

Easy to use and install. To facilitate adoption, a privacy system should be easy to use and install. One way to achieve this goal is to piggyback on an existing, popular BitTorrent client. By operating at the level of managing torrents, our software can be provided as a non-invasive extension to existing clients. As we discuss in Sec. 5.3, it also provides intuitive control over privacy/performance trade-offs.

Incrementally deployable. Our software should work well regardless of how many users adopt the system. In contrast, anonymization systems require a reliable set of altruistic relay nodes to carry traffic anonymously. Likewise, closed file-sharing networks (e.g., OneSwarm [14]) are limited by the content and performance available in the private network. Our approach relies only on a diverse set of publicly available torrents and a large number of worldwide users actively using BitTorrent – both of which are available today.

Practical incentives. Any extension to an existing, popular service should provide subscribers with proper incentives without undermining the performance of nonsubscribers. In our approach, subscribers bear the burden of reduced end-to-end performance in proportion to the level of privacy that they gain. Nonsubscribers also benefit from this service by receiving a certain level of additional privacy from subscribers participating in their torrents. Moreover, the software actually *improves* performance for nonsubscribers (at a global level), because SwarmScreen users contribute their bandwidth to the random torrents they join. This is in contrast with systems that enable privacy through "cover traffic" that is dropped along paths between senders and receivers, and thus results in wasted bandwidth.

Protocol compliance. Our approach does not require modifying the BitTorrent protocol to preserve privacy because it relies on connecting to a set of real, live torrents and exchanging data with other peers. The only requirement is that data for torrents not requested by the user can be placed in transient storage.

5.2 Architecture

Figure 8 depicts the architecture for our privacy-enhancing BitTorrent extension. The user specifies the torrents to download and privacy settings that guide the privacy/performance trade-off. The privacy manager then adds to the user-specified torrents a set of randomly selected *transient torrents* that the client will join but not store persistently. As the downloads progress, the privacy manager rotates the active set of torrents and connections in a way that obfuscates any patterns in connection behavior. The BitTorrent protocol itself remains unmodified.

5.3 Challenges

While the high-level architecture for SwarmScreen is ostensibly simple, we discovered and addressed several im-

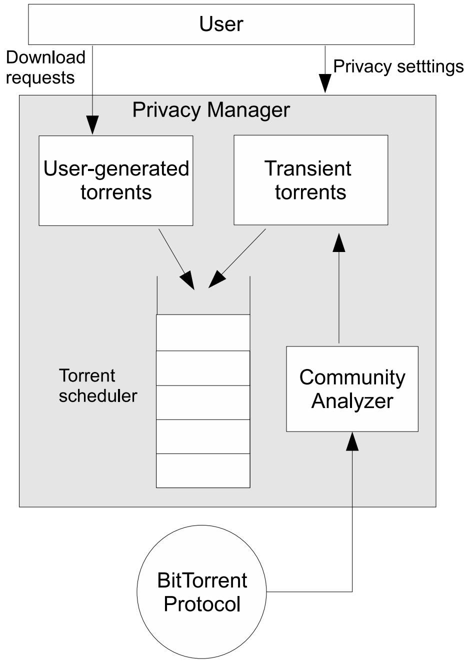


Figure 8: Architecture for our privacy-enhancing BitTorrent extension. Key system components are located in the gray box; note that because our approach operates above the protocol layer, it avoids invasive client modification.

portant challenges during our design and implementation of the system.

Privacy/performance trade-off. Privacy systems fail their users when they perform poorly, are difficult to use and/or are misconfigured. SwarmScreen addresses these issues through a simple interface that allows users to select a privacy/performance trade-off from an integer space that we call the SwarmScreen Protection Factor (SPF). Analogous to the Sun Protection Factor, a value of 0 indicates no protection and small values (less than 35 or 50) indicate that sustained exposure with these settings may lead to privacy violations. Further extending the analogy, one can set SPF values of 100 or higher, but the returns in protection diminish. In the next section, we discuss how we use SPF to explicitly control privacy and performance.

In addition to providing an intuitive mechanism for setting privacy levels, our system must give user feedback to ensure that those levels provide the expected level of protection (i.e., so the user does not get burnt by privacy violations). To address this issue, SwarmScreen displays statistical information about real-time privacy levels to the user. A user uncomfortable with current privacy levels can simply increase the system's SPF until achieving the desired protection.

Being inconspicuous. SwarmScreen improves privacy by inducing connections to peers in random swarms. To ensure that an attacker cannot detect the torrent to which each data packet corresponds, a user should enable connection encryption (a feature supported by most BitTorrent clients).

Simply adding large numbers of random connections, however, is not sufficient to evade attack; after all, an eavesdropper may simply look for an excessively large volume of connections to identify targets for further surveillance. To provide even more protection, the connections generated by SwarmScreen should be statistically indistinguishable from those generated by user-selected torrents.

To ensure that SwarmScreen-generated connection patterns are similar to those for user-specified torrents, we must define a notion of similarity. A simple approach is to ensure that the time spent in connections to random torrents is similar to those for user-specified torrents. Alternatively, the software can ensure that the distributions of connection durations for each category of torrents is identical, or emulate the distribution of bytes transferred over each connection.

Downloading random content. An important issue for any system that automatically transfers random content is determining the source of that content. In BitTorrent, locating content in a fully automated way is difficult because content metadata is separated from the protocol that transfers its data. Further complicating the issue, it is well known that a significant portion of content available through the protocol may be subject to legal restrictions for distribution. Thus, even if content is located automatically, transferring the content may subject the user to prosecution.

The case law on this issue varies from one jurisdiction to another, and the legal environment surrounding online content distribution is dynamic. As explained by Bauer et al. [1], the legal disposition of a system that forwards arbitrary content is far from settled — this issue affects *any* open relay system.[2] One solution, adopted by users of PlanetLab, is to download and discard torrent data, thus never uploading it. Unfortunately, this violates the goal of inconspicuous behavior – it is trivial to identify users that never upload content.

Because there is no universal solution for this problem, we allow users to determine where to find content and where to store it on their machines. The former option allows users to specify sites that contain only torrents that can be legally retransmitted while the former allows the user to choose where to store partially downloaded content (e.g., on a persistent or volatile store). Further, because the transient torrents are explicitly not content that users wish to keep, SwarmScreen transfers them for finite durations and immediately deletes content upon completion and when the user's session ends. Note that this approach is not particularly conspicuous; after all, the selfish behavior of BitTorrent users is well documented [19]. We also emphasize that deleting a file upon completion does not prevent the user from uploading content to other users –

[2] The authors note, however, that US law allows caching and retransmission of unmodified files.

any completed file pieces are made available to the swarm during the downloading phase.

6 Realization in BitTorrent

In this section, we discuss the implementation of our approach to improving privacy in P2P systems through an extension to a popular BitTorrent client. We then evaluate this implementation in terms of privacy and performance overhead.

6.1 Implementation Details

SwarmScreen is currently implemented as a plugin (i.e., extension) for the popular Vuze BitTorrent client, which facilitates user adoption by providing an interface for our software to be installed without any modifications to the mainline client. At the time of submission, our plugin is publicly available on our project webpage and it has been accepted for inclusion on the Vuze official plugin list.[3]

Our implementation is written in Java, which enables it to run on nearly every operating system. The core functionality contains approximately 2,100 LOC and has been released under an open-source (GPL) license. Throughout this section we mention configurable parameters as part of our implementation; their current values are listed in Table 2. In the remainder of this section, we describe several key implementation details.

Fetching torrents. As discussed in the previous section, SwarmScreen requires the user to select one or more Web pages containing links to torrent files. At uniformly random intervals (currently with a 1-hour mean), SwarmScreen connects to these pages and downloads a list of links for torrents. To enhance privacy and to evade detection, communication between a user and the Web server should be protected. Our software allows users to specify `https` sites so that the connection is encrypted. If needed, the user can tunnel these HTTP connections through Tor or other anonymization networks to fetch the Web pages – the fetching of random content is separated from the data-transfer protocol.

Once the links to torrent files have been fetched from the specified Web pages, SwarmScreen selects torrents from this set at random. In practice, we have found that the storage space required for any given torrent varies widely from tens of megabytes to tens of gigabytes. Our current implementation addresses the associated storage capacity issue by allowing users to specify a maximum file size for torrents to use. Any torrents with files larger than the maximum size are not selected.

There are many alternative ways to address the issue of limited storage. For instance, some BitTorrent clients (other than Vuze) assemble files incrementally, requiring storage space only for the pieces that have been downloaded. This approach would significantly reduce storage requirements, and could allow transient torrent data to be sufficiently compact to fit entirely in volatile storage (e.g., DRAM). At the time of submission, we are working with the Vuze team to add support for this feature in their client.

Connection profiling. A key goal of our approach is that it obscures a user's communities of shared interest without appearing to do so. In particular, the connection behavior generated by SwarmScreen should be statistically indistinguishable from user-generated behavior. To determine whether this is the case, SwarmScreen monitors its client's connections. At a regular interval, our software polls peers connected to user-generated and transient torrents to determine the total time spent in connections for each category.

This data allows SwarmScreen to determine the ratio of time spent connected to transient torrents versus the same for user-generated ones. Based on our goal of inconspicuous connections, this ratio should be kept near 1.0. The ratio is displayed to the user for monitoring real-time protection levels.

Torrent scheduling and shaping. When a user is downloading content, SwarmScreen induces connections by activating torrents. While the transfers progress, our software adjusts the number of active torrents and their maximum transfer rates to meet the privacy/performance trade-off specified by the user.

As we showed in Sec. 4, increasing the number of random connections increases undetectability and deniability for community analysis. To add random connections in BitTorrent, SwarmScreen joins additional swarms. Because the BitTorrent protocol is nondeterministic in terms of the number of connections established, it is difficult to precisely control the percent of random connections in SwarmScreen. We observe, however, that the Vuze BitTorrent client by default places an upper bound on the number of connections established per torrent. When this bound is reached, the fraction of random torrents is the same as the fraction of random connections. Based on this observation, we approximate control of random connections through the number of active torrents. Specifically, SwarmScreen by default adds SPF percent random torrents to the client. Based on our analysis from Sec. 4, we set the default SPF value to 50, which provides at least 50% deniability for $\tau = 5$.

If torrents do not use all of their allotted connections, this approach may not provide the desired privacy level. Thus, SwarmScreen currently monitors the ratio of connection times, r, and uses it to the drive a feedback loop for ensuring the appropriate amount of cover traffic. Specifically, we define an acceptable range $r_{min} > r > r_{max}$ and the software attempts to keep the ratio within the range. If $r < r_{min}$, SwarmScreen adds more transient torrents to

[3] Official plugins are signed to ensure their integrity.

Parameter	Value
SPF	50
Connection ratio thresholds $[r_{min}, r_{max}]$	[0.8, 1.2]
Maximum torrent size	1 GB
Torrent scheduling interval	15 sec
Torrent fetch interval (mean)	1 hour

Table 2: Key settings in our current SwarmScreen implementation. Parameters in bold are user-configurable parameters.

increase time spent in connections for random torrents. It also increases the time spent participating in each transient torrent – in essence, buying time for more connections to be established and used. Both the number of transient torrents and the time spent connected to them are directly proportional to $r_{min} - r$. Thus, as the ratio falls farther from the acceptable range, SwarmScreen works harder to bring the ratio back. Because the random connections can outnumber the user-specified ones, SwarmScreen similarly reduces the number of active transient torrents and the duration that they are connected when $r > r_{max}$.

It is important to note that if user-generated torrents are always connected, their communities of interest can easily be extracted from the random noise that SwarmScreen generates. Our software addresses this issue by randomly pausing and resuming user-generated torrents, similar to the way that it adds and removes transient torrents. It also increases the time that user-generated torrents are active when $r > r_{max}$ and decreases it when $r < r_{min}$.

Finally, SwarmScreen uses the SPF value to control the overhead imposed by the system. To limit the impact on transfer rates, the SPF value is interpreted as the maximum percent overhead that transient torrents can impose on user-generated ones. At any interval, SwarmScreen sets a cap on the bandwidth for each transient torrent, B_t, using the formula:

$$B_t = \frac{\text{SPF}}{100} \frac{\sum B_u}{N_t} \qquad (2)$$

where $\sum B_u$ is the sum of the bandwidth (B_u) for each user-generated torrent and N_t is the number of transient torrents currently active.

6.2 Evaluation

To demonstrate the effectiveness of SwarmScreen, we evaluate its performance in terms of privacy, performance and responsiveness to changing conditions. We measure privacy in terms of the connection ratio as described in the previous section. By tracking the ratio over time, we can observe the responsiveness of SwarmScreen to changing conditions. To measure performance, we use the ratio of throughput for user torrents to the same for transient torrents.

All of the tests were performed using Vuze 4.1.0.0 running behind a residential cable Internet connection. Unless otherwise indicated, we used an SPF value of 100 and used two popular Linux distributions as the user-generated downloads.

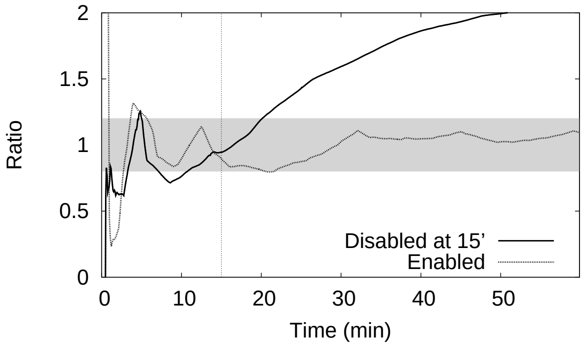


Figure 9: Timeline showing how the ratio changes without using SwarmScreen to adapt for r. At $t = 15$ minutes, we disable SwarmScreen for the experiment with the solid curve. Without the feedback loop, the ratio of durations for random connections to user connections increases dramatically, leading to identification through conspicuous behavior.

Fixed downloads and SPF. We begin our analysis by evaluating how well SwarmScreen provides privacy with a fixed number of downloads (2) and SPF value (100). The dashed curve in Fig. 9 shows how the connection ratio changes during a 60-minute session. Initially SwarmScreen traffic charges ahead of user traffic, then quickly recovers to provide relatively even connection rates within a few minutes. Note that it is normal for ratios to fluctuate during early portions of a session because the total time spent in connections is small relative to the amount of time added for each sample.

To demonstrate that it is not sufficient to simply add random torrents without controlling for the statistical properties of their connections, we repeat the same experiment, but this time we disable SwarmScreen's control mechanism at $t = 15$ min (solid curve in Fig. 9). Soon thereafter, it appears that the random torrents have achieved a balance with the user-specified ones. However, within a couple of minutes, the portion of time spent in connections to random torrents rises compared to the user-specified ones, and over time this discrepancy grows quite large. Because this represents vastly different behavior than the user-specified torrents, it is likely that an attacker would be able to classify the user as conspicuous and thus target that user for more advanced statistical attacks.

Fixed downloads, vary SPF. In this experiment, we use the same two fixed user-selected torrents but vary the SPF value across sessions. In Fig. 10, we plot a timeline of the r values for SPF 10, 50 and 100. The figure shows that larger SPF values allow SwarmScreen to adapt to changing conditions more rapidly, while lower SPF levels cause slower reactions to changes in r. One way to determine the impact of these different settings on privacy is to calculate

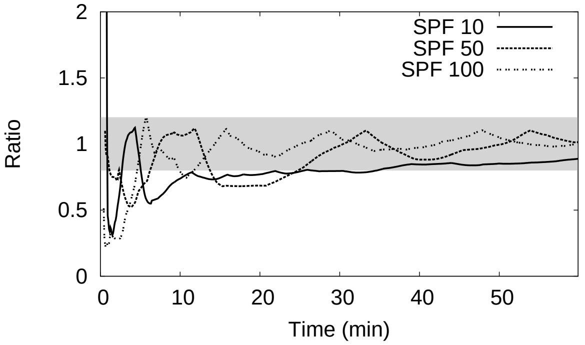


Figure 10: Timeline showing how different SPF levels affect the ratio of connections. Lower SPF values cause SwarmScreen to react more slowly to changes in r. For SPF 100, the amount of time outside the acceptable range for r is 10%; for SPF 50 the value is 27% and more than 50% for SPF 10.

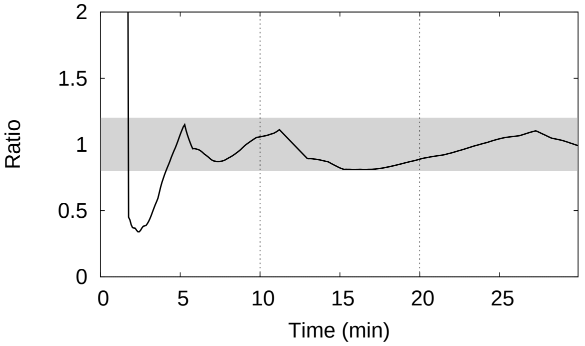


Figure 11: Timeline of similarity of total connection durations and distributions of connection durations while varying then number of user-selected downloads. Each vertical line indicates the time at which another user download is added.

the amount of time the system is outside the acceptable range for r for each SPF value. We find that with SPF 100, 10% of the time is spent outside the range. This time increases by nearly a factor of three for SPF 50, and for SPF 10 the system spends the majority (51%) of the time outside the acceptable range.

Vary downloads, fixed SPF. The goal of our next experiment is to determine how SwarmScreen adapts to changing workloads while providing acceptable privacy levels. To test this, we vary the number of downloads (from one to three) while keeping the SPF fixed at 100. Fig. 11 shows that our system maintains acceptable levels of privacy even when the user increases the number of torrents that must be hidden.

Controlling performance overhead. We have shown that SwarmScreen can provide acceptable privacy in a variety of scenarios; now we determine how well it bounds the overhead imposed by cover traffic. Specifically, for a given SPF value, we determine how well the system satisfies Equation 2. Our current implementation takes advantage of Vuze's built-in support for limiting the transfer rates on a per-torrent basis. Thus, the effectiveness of our implementation is primarily limited by the responsiveness of our system to changes in bandwidth and of Vuze to changes in bandwidth caps. Figure 12 shows a CDF of the ratio of bandwidth consumed by transient torrents to the same for user-selected torrents. We sample transfer rates every 5 seconds for three 30-minute sessions: one for each SPF value in the set of $\{10, 50, 100\}$. According to Eq. 2, the ratio of bandwidth allocated to transient and user-specified torrents, R_b, should be less than $\mathrm{SPF}/100$:

$$R_b = \frac{\sum B_t}{\sum B_u} \leq \frac{\mathrm{SPF}}{100}$$

We add vertical lines at $x = \{0.1, 0.5, 1.0\}$ to highlight the target values for each curve. The figure shows that the vast majority of samples fall within the target range for all of the download rates (87.5% to 95% of the values). For uploads, there are slightly more violations; this occurs because the upload capacity is about an order or magnitude smaller than the download capacity, meaning small absolute differences in upload bandwidth lead to larger relative differences. Even when the bandwidth ratios exceed the target value, the violations are not generally large.

Note that the variance in ratio values increases as SPF increases. This happens because large bandwidth allocations for random torrents may be underutilized if there is insufficient bandwidth available in the associated swarms. In this case, our user-specified torrents were Linux distributions, which are hosted by many peers located in well-provisioned university networks with high available bandwidth. As a result, the bandwidth received from these torrents was often greater than the same for random (non-Linux) torrents, leading to many ratio values less than $\mathrm{SPF}/100$. There are several ways to address this issue. For one, SwarmScreen can attempt to restrict transient torrents to those in its random set with similar swarm bandwidth availability. If this is not feasible, SwarmScreen could shape the bandwidth allocated to user-specified torrents to

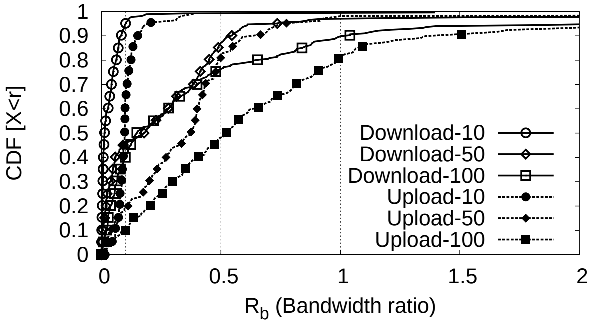


Figure 12: CDF of bandwidth allocation ratio samples for SPF values of 10, 50 and 100, corresponding to maximum allowed ratios of 0.1, 0.5 and 1.0. The caps are met most of the time, though there's varying degrees to which the allocated bandwidth is fully utilized.

match the same for transient torrents. As this policy would change the interpretation of SPF, we plan to make this option available to users but disabled by default.

7 Related Work

P2P systems have been the subject of numerous studies. Our work falls into the category of studies that use live data to drive their results [3, 12, 19]. Unlike previous work, this paper focuses on communities formed by hosts in P2P networks. In particular, we analyze how high-level communication patterns reveal information that can be exploited by eavesdroppers and develop techniques to thwart such attacks.

A common approach to enhance privacy in networked systems is to use encryption (e.g., with symmetric keys) to secure the contents of a data stream between two endpoints. While this makes it prohibitively expensive for an eavesdropper to obtain the plaintext data stream, it does not necessarily prevent an attacker from determining the contents of that stream. For example, Saponas et al. [26] demonstrated several classes of devices and attacks that allow an attacker to obtain information about users. In particular, they show that an eavesdropper can identify the video being watched simply by observing patterns in the encrypted Slingbox data stream. In a similar vein, we showed that an attacker needs access only to connection patterns (encrypted or not) to classify users. Thus, encryption alone is not sufficient to address this class of attacks.

Another privacy layer is anonymization, which entails disassociating user-identifiable information from network flows. There have been many approaches to providing this service, for communication channels [10, 27], content storage [4,7,28] and both [16]. Onion routing [23] attempts to provide anonymity of senders and receivers by sending data over multiple overlay hops, with each hop providing another layer of anonymity. A popular implementation of this approach is Tor [8], which focuses on providing a resilient, usable service for low-latency, interactive Internet tasks such as Web browsing and SSH sessions. However, attempts to integrate Tor into P2P systems has significantly reduced global Tor performance to the detriment of non-P2P users [18]. Recently, BitBlender [1] has been proposed for anonymizing BitTorrent, but their approach relies on open routers to forward BitTorrent traffic without incentives for participation. Unlike these solutions, which focus on hiding senders and receivers of data, our approach hides a user's communities of interest, which does not itself require hiding senders and receivers.

An alternative to anonymity in open networks is trusted identities in private (i.e., closed) networks [14, 20]. The advantage to private networks is that they provide end-to-end communication over trusted and encrypted channels, preventing attackers from identifying the senders, receivers or content being transferred. A disadvantage is that, unlike open networks, availability of content and bandwidth is limited by the set of trusted hosts. Further, closed networks often suffer significant performance overhead from the multiple overlay hops required to anonymize senders and receivers. Finally, these "networks of friends" are susceptible to community-based classification as described in this paper (i.e., the network is the community) and the guilt-by-association attack.

Statistical disclosure attacks are closely related to our work [17]. Their goal is to determine the set of message recipients for a particular target node in an anonymity network. Unlike this work, our approach allows an attacker to identify arbitrary recipients; however, we provide privacy by obfuscating which recipients are in the target's community of interest. Obfuscation can provide privacy by obscuring high-level structure in data sources. This technique is commonly used to hide trade secrets in executable code, e.g., by modifying an instruction stream to make it more difficult to reverse engineer automatically [21]. Unlike obfuscation in instruction streams, our system hides user behavior using online techniques, and eliminates wasted resources at a global level by participating in real torrents.

Our work is inspired by several projects that share its spirit but are applied in different contexts. Cortes et al. [5] describe an approach for identifying communities of interest, but this was intended to identify fraudulent callers in a phone network. Li et al. [15] describe a crowd-based approach for protecting privacy in data streaming applications. Finally, Crowds [24] hides a user's Web request in a crowd of other requests for the same content; similarly, our work provides a degree of privacy by hiding user-generated BitTorrent traffic in a crowd of traffic for random torrents.

8 Discussion and Future Work

Any distributed system implicitly reveals information about the participating user; in this paper, we focus on one such case in the BitTorrent P2P system. Our analysis formed communities based only on the fact that two P2P users established a network connection. While this simple approach still reveals strong community structure in BitTorrent, rich connection information can further enhance the statistical significance of detected communities. For example, one can construct graphs that weight edges according to the number of times two users connect, the duration of the connections and the amount of data transferred. Similarly, one can account for patterns in connection behavior over time when forming communities. For example, users that connect to each other at regular intervals are more susceptible to classification. We are currently investigating these aspects of community analysis; defenses against them are part of our future work.

SwarmScreen enhances privacy by defending against the guilt-by-association attack enabled by community structure in BitTorrent networks. It is important to note that an explicit nongoal of our approach is anonymity. Rather, we focus on the goals of disrupting community detection and providing users plausible deniability, as defined in Sec. 4, when assigned to a community. However, we make no claim about how our approach satisfies the legal definitions of these goals.

Privacy in the Internet has become a critical issue, with notable catalysts such as identity theft and government eavesdropping on network communications. In this paper, we focused on the BitTorrent P2P protocol because community structure in this network is not expected a priori. We believe that the related guilt-by-association attack is relevant in many other P2P systems, especially where community structure is imposed by the semantics of the network (e.g., in a friend-to-friend network).

9 Conclusion

As P2P systems grow in size and popularity, privacy becomes an increasingly important – and challenging – goal to achieve. In this paper, we analyzed connection information from real users in the BitTorrent network and revealed strong communities of shared interest. We showed that this information can be exploited by an attacker to classify large numbers of users with relatively little monitoring. To address this threat, we designed and implemented a strategy to disrupt attempts to classify users. As part of our future work, we are exploring other privacy threats based on connection behavior in P2P systems and defenses against them.

References

[1] Arenas, A., Danon, L., Díaz-Guilera, A., Gleiser, P. M., and Guimerà, R. Community analysis in social networks. *Eur. Phys. J. B 38* (2004), 373–380.

[2] Bauer, K., McCoy, D., Grunwald, D., and Sicker, D. BitBlender: Light-weight anonymity for bittorrent. In *Proc. AIPACa* (Istanbul, Turkey, September 2008).

[3] Biip. Biip.no. Internet-based community in Norway.

[4] Brandes, U., Delling, D., Höfer, M., Gaertler, M., Görke, R., Nikoloski, Z., and Wagner, D. On finding graph clusterings with maximum modularity. In *Proceedings of the 33rd International Workshop on Graph-Theoretic Concepts in C* (2007), Lecture Notes in Computer Science.

[5] Choffnes, D. R., and Bustamante, F. E. Taming the torrent: A practical approach to reducing cross-ISP traffic in peer-to-peer systems. In *Proc. of ACM SIGCOMM* (Aug. 2008).

[6] Clarke, I., Sandberg, O., Wiley, B., and Hong, T. W. Freenet: A distributed anonymous information storage and retrieval system. *Lecture Notes in Computer Science 2009* (2001), 46–66.

[7] Cortes, C., Pregibon, D., and Volinsky, C. Communities of interest. In *IDA* (2001), pp. 105–114.

[8] Crooks, A. BitTorrent peer exchange conventions. `http://wiki.theory.org/BitTorrentPeerExchangeConventions`.

[9] Danon, L., Díaz-Guilera, A., Duch, J., and Arenas, A. Comparing community structure identification. *J. Stat. Mech.: Theor. Exp.* (2005), art. no. P09008.

[10] Dingledine, R., Freedman, M. J., and Molnar, D. The free haven project: Distributed anonymous storage service. In *Proc. of PETS* (July 2000).

[11] Dingledine, R., Mathewson, N., and Syverson, P. Tor: The second-generation onion router. In *Proc. of USENIX Security Symposium* (2004), pp. 303–320.

[12] Duch, J., and Arenas, A. Community detection in complex networks using extremal optimization. *Phys. Rev. E 72* (2005), art. no. 027104.

[13] Facebook, Inc. facebook.com.

[14] Freedman, M. J., and Morris, R. Tarzan: A peer-to-peer anonymizing network layer. In *Proc. of ACM CCS* (Washington, DC, November 2002).

[15] Graham-Rowe, D. Sniffing out illicit bittorrent files, February 2009. `http://www.technologyreview.com/computing/22107/page1/`.

[16] Guimerà, R., and Amaral, L. A. N. Functional cartography of complex metabolic networks. *Nature 433* (2005), 895–900.

[17] Guimerà, R., Mossa, S., Turtschi, A., and Amaral, L. A. N. The worldwide air transportation network: Anomalous centrality, community structure, and cities' global roles. *Proc. Natl. Acad. Sci. USA 102*, 22 (May 2005), 7794–7799.

[18] Guimerà, R., Sales-Pardo, M., and Amaral, L. A. N. Modularity from fluctuations in random graphs and complex networks. *Phys. Rev. E 70* (2004), art. no. 025101.

[19] Guimerà, R., Sales-Pardo, M., and Amaral, L. A. N. Classes of complex networks defined by role-to-role connectivity profiles. *Nature Phys. 3* (2007), 63–69.

[20] Gummadi, K. P., Dunn, R. J., Saroiu, S., Gribble, S. D., Levy, H. M., and Zahorjan, J. Measurement, modeling, and analysis of a peer-to-peer file-sharing workload. In *Proc. of the ACM SOSP* (2003), pp. 314–329.

[21] IFJ. IFJ condemns censorship pact as Dubai takes Pakistani media off the air, November 2007. http://www.ifj.org/en/articles/ifj-condemns-censorship-pact-as-dubai-takes-pakistani-media-off-the-air.

[22] Isdal, T., Piatek, M., Krishnamurthy, A., and Anderson, T. Friend-to-friend data sharing with OneSwarm. Tech. report, University of Washington, February 2009.

[23] Li, F., Sun, J., Papadimitriou, S., Mihaila, G., and Stanoi, I. Hiding in the crowd: Privacy preservation on evolving streams through correlation tracking. *Proc. ICDE* (April 2007), 686–695.

[24] Marc Waldman, A. D. R., and Cranor, L. F. Publius: A robust, tamper-evident, censorship-resistant, web publishing system. In *Proc. of USENIX Security Symposium* (August 2000), pp. 59–72.

[25] Mathewson, N., and Dingledine, R. Practical traffic analysis: Extending and resisting statistical disclosure. In *Privacy Enhancing Technologies Workshop* (2004), pp. 17–34.

[26] McCoy, D., Bauer, K., Grunwald, D., Kohno, T., and Sicker, D. Shining light in dark places: Understanding the Tor network. In *Proc. of PETS* (Leuven, Belgium, July 2008), pp. 63–76.

[27] Newman, M. E. J., and Girvan, M. Finding and evaluating community structure in networks. *Phys. Rev. E 69*, 2 (2004), 026113.

[28] Piatek, M., Isdal, T., Anderson, T., Krishnamurthy, A., and Venkataramani, A. Do incentives build robustness in bittorrent. In *Proc. of USENIX NSDI* (April 2007).

[29] Popescu, B. C., Crispo, B., and Tanenbaum, A. S. Safe and private data sharing with Turtle: Friends team-up and beat the system. In *Proc. Cambridge Intl. Workshop on Security Protocols* (2004).

[30] Popov, I. V., Debray, S. K., and Andrews, G. R. Binary obfuscation using signals. In *Proc. of USENIX Security Symposium* (2007), pp. 275–290.

[31] QIANG, X. The development and the state control of the chinese internet. *Before the U.S.-China Economic and Security Review Commission* (April 2005). http://www.uscc.gov/hearings/2005hearings/written_testimonies/05_04_14wrts/qiang_xiao_wrts.htm.

[32] REED, M. G., SYVERSON, P. F., AND GOLDSCHLAG, D. M. Anonymous connections and onion routing. *IEEE Journal on Selected Areas in Communications 16* (1998), 482–494.

[33] REITER, M. K., AND RUBIN, A. D. Crowds: Anonymity for web transactions. *ACM Transactions on Information and System Security 1* (1998), 66–92.

[34] RISEN, J., AND LICHTBLAU, E. Spying program snared U.S. calls. *The New York Times* (December 2005).

[35] SAPONAS, T. S., LESTER, J., HARTUNG, C., AGARWAL, S., AND KOHNO, T. Devices that tell on you: privacy trends in consumer ubiquitous computing. In *Proc. of USENIX Security Symposium* (2007), pp. 55–70.

[36] SHERWOOD, R., BHATTACHARJEE, B., AND SRINIVASAN, A. P5: A protocol for scalable anonymous communication. In *Proc. IEEE Symposium on Security and Privacy* (May 2002).

[37] SIRER, E. G., GOEL, S., ROBSON, M., AND ENGIN, D. Eluding carnivores: file sharing with strong anonymity. In *Proc. ACM SIGOPS European Workshop* (2004), p. 19.

[38] SMITH, C. Anti-piracy law a reasonable way to protect artists' rights. *The New Zealand Herald* (March 2009). http://www.nzherald.co.nz/politics/news/article.cfm?c_id=280&objectid=10560605.

[39] VKONTAKTE. Vkontakte.ru. Primary Russian social network.

[40] WASSERMAN, S., AND FAUST, K. *Social Network Analysis*. Cambridge University Press, Cambridge, UK, 1994.