

SocialSwarm: Exploiting Distance in Social Networks for Collaborative Flash File Distribution

Matthew J. Probst, Jun Cheol Park, Ravin Abraham, and Sneha Kumar Kasera
School of Computing, University of Utah
Email: {mprobst, jcpark, abrahamr, kasera}@cs.utah.edu

Abstract—Social networks can serve as an effective mechanism for distribution of vulnerability patches and other malware immunization code. We propose a novel approach—SocialSwarm—by which peers exploit distances to their social peers to approximate levels of altruism and to collaborate on flash distribution of large files. SocialSwarm supports heterogeneous BitTorrent swarms of mixed social and non-social peers. We implement SocialSwarm as an extension to the Rasterbar libtorrent library—widely used by BitTorrent clients—and evaluate it on a testbed of 500 independent clients with social distances extracted from Facebook. We show that SocialSwarm can significantly reduce the average file distribution time, not only among socially connected peers, but also among other swarm participants.

I. INTRODUCTION

Online social networks (OSNs) are perpetually increasing in popularity and utility. Unfortunately, most if not all OSNs have been heavily exploited for malicious purposes. One of the oldest and certainly the most pervasive of all OSNs, SMTP-based email, has been widely used to self-propagate malicious code, either automatically via client vulnerabilities or manually via social engineering, in messages sent to unsuspecting or inexperienced users. When new malicious code enters a social network, it commonly infects hub nodes—nodes with higher degrees of connectivity and malware exposure—more quickly than those users with relatively few social peers.

Brumley et al. [1] have found a technique for automatic generation of malware based on contents of a patch file. Computer users can therefore no longer expect a generous time window for patching their systems before malware is released to exploit the patch. It is becoming increasingly important to study and develop counter-malware techniques such as flash [2] patch distribution, which, like malware itself, exploits hub, cluster, and relative node distance properties to enhance security within OSNs. Unfortunately, existing methods for fast distribution of large files, including typical operating system and application patch files, suffer from two problems in comparison with OSN-based malware propagation. First, systems such as BitTorrent, and its existing derivatives, implement mechanisms to minimize free-riding; such mechanisms create inefficiencies. Second, existing peer-to-peer (p2p) file distribution systems do not ensure that social hubs receive the highest priority in receiving files; such prioritization of patch distribution is necessary to effectively counter OSN-based malware infection campaigns.

The mechanisms used to minimize free-riding are typically tit-for-tat and auction-based p2p incentives [3]. Although these

incentives are valuable and necessary in fostering collaboration among purely self-interested individual peers, they come at the price of reduced efficiency. In order to barter for resources, BitTorrent reserves bandwidth in the form of unchoke slots [4]. This reserved bandwidth ensures that peers with which a node is attempting to barter are provided resources of reasonable value. Clients typically avoid increasing the number of unchoke slots because increased numbers reduce the value of each individual unchoke slot thereby increasing the difficulty of negotiating for higher levels of bandwidth. A client's offer of reserved bandwidth to one of its peers does not mean the bandwidth offered will actually be used. The recipient of the bandwidth offer may not have a sustained need for chunks held by the peer or a sustained capability of completely utilizing the offered bandwidth. This holds true even for systems that track and attempt to maintain peer reputation across different swarms [5]. For standard BitTorrent clients that do not track peer reputation across swarms, there is some probability, given optimistic unchoking, that peers will be offered and take bandwidth but will not reciprocate. Such peers are known as leeches. One of the goals of most p2p reputation-tracking systems is to handle leeches. Unfortunately, reputation-based systems are only valuable when kept fresh with a stream of evidence. For infrequent p2p users, reputation systems are commonly inaccurate and/or possibly punitive, given the lack of accumulated reputation evidence.

In this paper, we propose a novel flash file/patch distribution method, SocialSwarm, which leverages altruism between peers in an OSN to overcome the necessity of and the inefficiency created by negotiating for resources. SocialSwarm facilitates efficient file distribution among social peers. Specifically, SocialSwarm enables groups or teams of social peers with pre-established altruism between each other to use resources more effectively by reducing the requirement of resource bartering between members of the same team. Assistance to peers is prioritized proportionally to social altruism. SocialSwarm also facilitates file distribution between social peers and non-social peers using the well established BitTorrent mechanism of resource bartering. SocialSwarm can be described as a gather-then-share technique. Nodes first work as a team to interact with anonymous non-social peers, gathering socially rare chunks of the file being propagated. As the percentage of chunks held by members of the social group gradually increases, SocialSwarm-enabled group members turn inward and share the chunks altruistically among themselves.

Relative levels of altruism toward directly connected social peers are estimated by SocialSwarm using ratios of recent reciprocal real-user social network interactions. Direct levels of altruism, however, cannot be used to determine levels of altruism between indirectly connected social peers. With the goal of facilitating the approximation of relative levels of altruism among indirect social peers, we introduce a new metric: SocialDistance. SocialDistance is a synthetic metric that combines direct levels of altruism between peers with an altruism decay for each hop to approximate indirect levels of altruism. The resulting multi-hop altruism levels are used by social peers to proportion and prioritize the sharing of resources with other social peers.

We evaluate the effectiveness of SocialSwarm by implementing it as an extension to the Rasterbar libtorrent library [6] and deploy it on a testbed of 500 clients. Each client is assigned the identity of a real-world Facebook user and given connectivity characteristics of real world networks. We find that SocialSwarm achieves an average file download time reduction of 25% to 35% in comparison with standard BitTorrent under a variety of configurations and conditions including file sizes, maximum SocialDistance, as well as leech and seed counts. The most socially connected peers yield up to a 47% decrease in download completion time in comparison with average non-social BitTorrent swarms.

The rest of this paper is organized as follows. In Section II, we present relevant related work. In Section III, we introduce SocialDistance as an approximation of levels of altruism between peers in a social network and present the details of SocialSwarm which exploits the altruism approximations. Section IV constitutes an overview of our implementation of SocialSwarm. In Section V, we evaluate the performance of SocialSwarm. We conclude with a list of several areas for further investigation in Section VI.

II. RELATED WORK

P2P networks, when compared to static single-source models, have exhibited faster transmission time and greater robustness in dissemination of large files [7]. These advantages have been repeatedly acknowledged in research on security patch propagation where the standard model for patch dissemination is p2p in nature [8], [9].

Gossip-based protocols have been successful in improving BitTorrent’s file dissemination time. For example, CREW [10], a gossip-based protocol, clearly outperforms other p2p protocols including BitTorrent for small-sized (1MB) files. However, for bigger file sizes, CREW incurs a higher overhead than non-gossip-based protocols. Lind et al. [11] show how small messages spread over social networks through gossip. Our work, however, focuses on the propagation of large files, which have very different propagation characteristics in comparison with small files and messages.

BitThief [12] highlights the free rider vulnerability of BitTorrent by demonstrating that entire files can be downloaded without reciprocation. Augmenting BitTorrent to handle the free riding problem has been the focus of numerous research

exercises [13], [14]. The key idea of these solutions is to establish a trust value that a peer accumulates over the course of time. The trust metric thus penalizes peers that do not share and rewards those who are active sharers. Although the notion of trust is a step in the right direction, the peers that participate in a typical p2p swarm change constantly and the benefit of having a trust metric is often lost.

Zhu et al. [15] propose a novel method of worm containment in cellular networks by prioritizing the patching of mobile peers based on their social connectivity. Our technique of collaborative patch distribution—based on social trust on a high level—exploits the same connectedness property proposed by Zhu, but unlike his work, we do not assume any global or centralized knowledge of social connectivity between peers.

Friedman [16] describes the motivation of utilizing the social network as an excellent medium for patch distribution. One of the more convincing reasons presented is that OSNs like real life social networks tend to follow social norms. If computer security is treated as a given norm, a good peer would expeditiously forward a patch file to its social peers: first, to enhance the overall security of the social ecosystem, and, more importantly, to protect itself by having peer nodes that are malware-immune. Our work matches the efficiency of a p2p system with the complex dynamics of a social swarm to create a unique and robust file distribution system.

Different methodologies have been tried to incentivize sharing in social networks. 2Fast encourages sharing in a traditional p2p network by introducing the concept of “helper” peers, which assist “collector” peers—nodes interested in downloading a particular file [17]. The helpers use their idle bandwidth to collect chunks under the direction of the collectors with the ulterior motive that the collectors will help when the helpers need to download a file.

KARMA [18] proposes an incentive system with a more fluid “currency-like” mechanism, where a node can transfer some of its positive currency balance to bootstrap a lower placed node. Our work uses a multi-hop-based incentive metric that automatically confers the advantage of being associated with a higher placed node or even being part of a higher trust path.

The Tribler [19] system extends 2Fast to social networks by applying the concept of helper and collector peers to social cliques extracted from p2p networks by grouping peers of similar characteristics. Although the concept of a drone helper to retrieve content is useful for assisting a peer with the retrieval of some content, it does not assist a large group of social peers in identifying and retrieving socially rare chunks.

We share with Tribler the common goal of harnessing a social network for file distribution, but our work differs from Tribler’s in several meaningful ways. The collaborative download in Tribler needs “helper” nodes that do not participate in the actual file being distributed. This is in contrast to our work, where all the nodes in the swarm actively collaborate in the file being shared. Another important operational aspect of Tribler is that the helper nodes need explicit approval from the collector node regarding the uniqueness and rarity of a file

chunk before downloading it on behalf of the collector node. We develop the notion of social rarity of a chunk that gives a node sufficient confidence to download a chunk that is socially relevant to the clique to which it belongs. Tribler’s incentive mechanism does not allow for a transitive relationship between a prospective collector node and a helper node. All incentives that are reclaimed correspond to the direct interactions in the past between the nodes in purview.

The standard BitTorrent protocol uses a fixed and small number (typically 5 to 8) of unchoke slots for playing tit-for-tat with swarm members. When a peer joins a swarm, it initially chooses a random chunk to download and then begins to offer this chunk to barter with other swarm members. During these initial stages of bootstrapping swarm entry and tit-for-tat negotiation, some portion of the peer’s upload bandwidth is underutilized given the small and fixed number of unchoke slots. As has been shown in the SeCond [20] protocol, a more efficient use of p2p bandwidth is to freely share it with swarm peers regardless of reciprocity. Although free bandwidth sharing is more efficient, it is vulnerable to exploitation by any purely self-interested swarm member. Tit-for-tat thus serves as a required and effective enforcement mechanism for minimizing the level of free-riding possible in a swarm whose members are purely self interested.

Karame et al. [21] defend the analytical rationale behind decomposing p2p peers engaged in a collaborative download into “small coalitions” to give a near-optimal file distribution time. The key idea of this work is that the aggregation of locally optimal solutions obtained in the smaller teams form a globally optimal solution, which is often very expensive to compute if the problem is not decomposed. Our work builds upon these conclusions by engaging social peers to work together as teams.

Dynamic configuration of p2p peers of similar characteristics into teams has been shown to limit free riding in collaborative downloads [22]. A common approach tried in team-based collaborative downloads is to presume altruism with social peers while employing tit-for-tat policy with non-social peers. This model has been studied by Galuba et al. [23] where preference is always given to peers. In contrast, we employ a “gather-then-share” approach with social peers where the altruism exhibited with social peers is inversely proportional to the overall rarity of the file chunks. Our approach gives faster file dissemination because the social peers actively try to get file chunks that are rare in the group, which ultimately benefits the group in later stages when altruism comes into play.

The notion of trust applied to social networks is a promising method to encourage sharing. SPROUT [24] models a social network-based routing scheme where the path selected has peers contributing to the highest computed gross trust value. We extend the multi-hop notion of trust to generate altruism values for spatially distant non-peers by combining the intermediate altruism values and a linear decay factor proportional to spatial distance. Different approaches to trust computation over multiple hop distances have been tried. Walter et al. [25]

calculate the trust of a node at a given distance to be the product of the trust of all nodes along the path. Decay-based multi-hop trust metrics have been studied extensively by Marti et al. [24]. Decay-based models are advantageous because they adhere to the social network trust model of having more confidence in nodes closer to the source. We chose to implement a linear decay-based model, which is shown in the aforementioned work, to perform equally as well as the exponential decay-based model.

III. SOCIALSWARM DESIGN

SocialSwarm combines tit-for-tat/auction between non-social peers with an altruistic sharing of resources among social peers. A SocialSwarm client freely offers bandwidth to its social peers based on each of their SocialDistances. For non-social peers as well as those who are distant socially, a peer uses the standard BitTorrent method of engaging in tit-for-tat to negotiate bandwidth. In section III-A we provide an overview of the design of SocialSwarm, which is followed, in section III-B, by details on the notations we use. Finally, section III-C provides the core details of how these notations are combined and used within SocialSwarm.

A. Overview

With the goal of maximizing collaboration between social peers by reducing the inefficiencies of BitTorrent while still maintaining game-based techniques to encourage the cooperation of non-social peers, we have developed the following design characteristics for SocialSwarm.

- Full compatibility with the BitTorrent protocol: SocialSwarm is designed to leverage the existing benefits of the BitTorrent protocol, while enhancing the capabilities of BitTorrent clients to collaborate with social peers. SocialSwarm thus adapts to mixed swarms of socially connected and non-socially connected peers.
- Social network independence: With its modular social network analyzer, SocialSwarm can exploit the history of connectivity and interaction among social peers within any capable social networking system, obviating reliance on any particular social protocol or messaging system.
- Coordination of chunk collection among social peers: Like standard BitTorrent, SocialSwarm begins by relying on peers seeking chunks of data that are rare system-wide. Over time, however, SocialSwarm increasingly seeks after chunks that are rare among social peers only—socially rare chunks. Thus, as a file download progresses, a peer transitions its focus from globally rare chunks towards socially rare chunks. This transition of chunk collection focus is the “gather” portion of the “gather-then-share” approach and is detailed in section III-C2.
- Adaptive unchoke slot count and upload bandwidth allocation: Like standard BitTorrent, SocialSwarm begins by allocating a fixed number of unchoke slots for playing tit-for-tat/auction games with non-socially connected peers. Over time as more chunks are acquired by social peers, SocialSwarm gradually decreases the number of unchoke

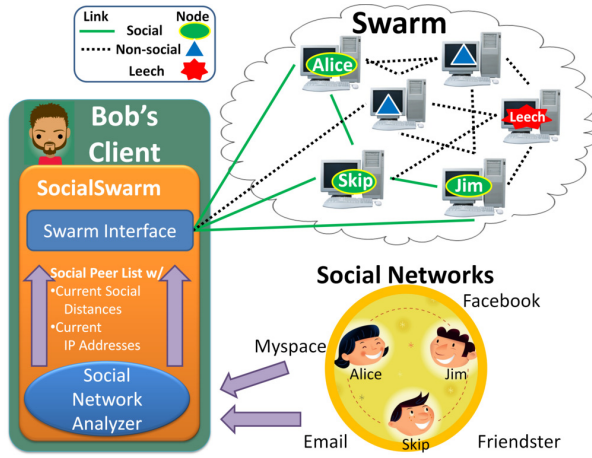


Fig. 1: SocialSwarm Interaction Overview

slots allocated to tit-for-tat/auction and re-purposes the bandwidth associated with those slots, offering unchoke slots freely to social peers who are granted bandwidth with a priority based on SocialDistance. This adaptive bandwidth allocation is the “share” portion of the “gather-then-share” approach and is detailed in section III-C1.

- Targeted non-social peer selection for optimistic unchoke: Rather than a commonly used metric of time-since-last-unchoke for optimistic unchoke peer selection, SocialSwarm is designed to select peers probabilistically based on the social rarity of file pieces that they hold. Details of this peer selection method are given in section III-C3.

Fig. 1 presents an example of how SocialSwarm running on Bob’s client retrieves social peer interaction history from social networks on behalf of Bob. When a file distribution swarm starts, Bob’s SocialSwarm client identifies Bob’s social peers in the swarm, coordinates chunk collection with them, and altruistically shares bandwidth with them. The SocialSwarm clients interact with each other as well as with standard (non-social) BitTorrent clients. In this example Jim’s SocialSwarm client also has a connection to a leech. Although Jim initially uses resource bartering and preallocates resources to negotiate with the leech, over time Jim reserves less bandwidth for resource bartering and openly shares its bandwidth with its social peers. By reserving fewer resources for specific peers, Jim’s risk is diversified given that his reliance on any particular node is reduced. Thus, even though Jim is not explicitly aware of the leech, SocialSwarm’s diversified collaboration allows it to be less affected by the leech in comparison with standard BitTorrent clients.

B. Notations

Table I provides an overview of the given, measured, and derived variables used by SocialSwarm. All variables are from the perspective of each node in the system. Each node independently receives, measures, and derives its own set of variables. Each measured variable is over a time period of t_i . Each derived variable is recalculated each t_i . The given

TABLE I: Notations

Given Variables	Descriptions
$A(a, b)$	Normalized level (0,1] of altruism a has towards a social peer b .
$A(s)$	The short form of $A(myself, s)$
$D_s(a, b)$	SocialDistance between a and b
$D_s Max$	Maximum SocialDistance whereby a client considers peers to be part of its social network.
C	Set of chunks in the file being downloaded. Knowledge of this set is provided by the .torrent file used to start the swarm.
t_i	The time interval i . The interval used here is the optimistic unchoke interval (commonly 30 seconds).
S	The set of social peers ($D_s \ll \infty$) that are using a SocialSwarm-enabled BitTorrent client. To form S , a list of all peers is retrieved from the torrent tracker; non-social peers are excluded.
N	The set of all other (non-social) peers participating in the swarm (where $SocialDistance = \infty$). To form S , a list of all peers is retrieved from the torrent tracker, and social peers are excluded.
Measured Variables	Descriptions
$V(t_i, c, n)$	0 or 1 indicating the availability of a particular chunk, c , at a particular peer, n , at time interval t_i . This information is shared between peers and the tracker as part of the BitTorrent protocol.
$B_s(t_i)$	The percentage of a client’s upload bandwidth used at t_i for altruistic sharing with its social peers. (each client measures its own bandwidth)
$B_n(t_i)$	The percentage of a client’s upload bandwidth used at t_i for bartering with its non-social peers. (each client measures its own bandwidth)
$R_o(t_i, c)$	The overall rarity of a chunk c across all peers, social and non-social, at t_i .
$R_s(t_i, c)$	The social rarity of a chunk c across its set S of social peers at t_i .
$R_n(t_i, c)$	The non-social rarity of a chunk c across its set N of non-social peers at t_i .
Derived Variables	Descriptions
$R_w(t_i, c)$	The combined weighted rarity at t_i .
$R_A(t_i)$	The average social rarity across all the chunks C at t_i .
$H(t_i, p)$	The weighted rarity of chunks held by a peer p at t_i .

and measured variables that could not be fully described in the Table are described in detail in this section. The next section (III-C) describes the derived variables along with the SocialSwarm algorithm.

1) *Altruism Between Direct Social Peers:* Altruism between two social peers should not be considered a dichotomy but rather a scale ranging from minimal to very high. Prompting system users to quantify levels of altruism for each of their social peers would be cumbersome and impractical. Instead

SocialSwarm calculates a proportional and directed level of altruism between each given peer a and one of its peers b via equation (1).

$$A(a, b) = \frac{I(a, b)}{I(a, all)} \quad (1)$$

where $I(a, b)$ is the number of reciprocal interactions a has had within a given time window with b , and $I(a, all)$ is the number of reciprocal interactions a has had with all of its peers during the same window of time. Effectively $A(a, b)$ represents the proportional willingness that a peer a has to share resources with each of its direct peers. It is important to note that $A(a, b)$ is from the perspective of a and is asymmetric.

2) *Approximating SocialDistance Between Indirect (Multi-Hop) Peers*: The SocialDistance between two direct or indirect peers can be considered the inverse of the altruism between those peers: $D_s(a, b) = \frac{1}{A(a, b)}$. Although SocialDistance itself does not have any absolute meaning, it is a useful synthetic metric to assess relative depreciation of altruism across intermediate peers and to determine which indirect paths between peers yield the highest levels of altruism (the shortest SocialDistance paths). It is important to note that altruism between two peers, $A(a, b)$, is asymmetric and thus SocialDistance, $D_s(a, b)$, is also asymmetric.

Given the known levels of SocialDistance across two pairs of social peers $D_s(a, b)$ and $D_s(b, c)$, SocialSwarm calculates a candidate multi-hop-directed SocialDistance from a to c with equation (2).

$$D_s(a, c)_{candidate} = \left(\frac{1}{\frac{1}{D_s(a, b)} \times \frac{1}{D_s(b, c)} \times HopDecay} \right) \quad (2)$$

Where $HopDecay$ is between 0 and 1 and is set by the evaluating peer a , the SocialDistance between any pair of peers (x, z) indirectly connected via a set of intermediary peers I is defined as the highest altruism level among all known paths between x and z . SocialDistance is calculated via equation (3).

$$D_s(x, z) = \forall i \in I : \min \left(\frac{1}{\frac{1}{D_s(x, i)} \times \frac{1}{D_s(i, z)} \times HopDecay} \right) \quad (3)$$

where $\left(\frac{1}{\frac{1}{D_s(x, i)} \times \frac{1}{D_s(i, z)} \times HopDecay} \right) \leq D_sMax$

$HopDecay$ is thus applied for each additional hop in a given social network path. Note that SocialSwarm narrows its search to a computationally reasonable search space by using a specified maximum useful SocialDistance (D_sMax) beyond which peers are not considered as social peers. These calculations are very similar to those of Dijkstra to find shortest paths; but instead of adding path lengths, SocialSwarm multiplies approximated levels of altruism.

For computing cumulative altruism values for a non-social peer who is multiple hops away, we take the product of the

TABLE II: SocialSwarm in a Nutshell

	SocialSwarm Action	Input Heuristic
1	Vary % bandwidth offered to social vs non-social peers	% of game completed (from social group perspective)
2	Vary the set of targeted chunks based on the group (social or non-social) being collaborated with currently	Bandwidth % used by social & non-social peers
3	Probabilistically unchoke the non-social peers that hold the most desired chunks	Rank peers based on the rarity of the chunks that they hold

individual altruism values along the path. This conforms to the social network convention of having the decrease in altruism value proportional to the relative distance. The same rationale holds for having a linear $HopDecay$ applied at each hop on the multi-node path between two peers.

3) *Approximating Altruism Between Indirect (Multi-Hop) Peers*: For all peers a and c for which there exists no direct social relationship, the Altruism between a and c is defined as the inverse of the SocialDistance between those peers:

$$A(a, c) = \frac{1}{D_s(a, c)}$$

4) *Overall Rarity for Each Given Chunk*: For each given chunk c , a peer calculates the overall rarity of the chunk across all peers in the swarm (peers in S and N) via equation (4).

$$R_o(t_i, c) = \begin{cases} 1 & \text{if } |S \cup N| = 0 \\ 1 - \frac{\sum_{n \in S \cup N} V(t_i, c, n)}{|S \cup N|} & \text{otherwise} \end{cases} \quad (4)$$

5) *Social Rarity for Each Given Chunk*: For each given chunk c , a peer calculates the rarity of the chunk across its set S of social peers using equation (5).

$$R_s(t_i, c) = \begin{cases} 1 & \text{if } \sum_{s \in S} A(s) = 0 \\ 1 - \frac{\sum_{s \in S} A(s) * V(t_i, c, s)}{\sum_{s \in S} A(s)} & \text{otherwise} \end{cases} \quad (5)$$

This equation allows each node to weigh the priority of each chunk proportionally with the altruism expressed towards each of the node's online and in-swarm social peers. The chunks that are rare to social peers connected by shorter SocialDistances, and thus higher levels of altruism, are assigned a higher priority than the chunks that are rare to peers connected by higher SocialDistances.

6) *Non-social Rarity for Each Given Chunk*: A peer calculates the rarity of a given chunk across its set N of non-social peers using equation (6).

$$R_n(t_i, c) = \begin{cases} 1 & \text{if } |N| = 0 \\ 1 - \frac{\sum_{n \in N} V(t_i, c, n)}{|N|} & \text{otherwise} \end{cases} \quad (6)$$

C. SocialSwarm Algorithm

SocialSwarm varies the behavior of standard BitTorrent in three basic ways. Table II lists these three changes with their respective input heuristics. Each of these actions and heuristics is described in detail below:

1) *Adaptive Bandwidth Allocation:* SocialSwarm leverages munificence between social peers by dynamically allocating a portion of available bandwidth toward free bandwidth sharing with social peers.

Karame et al. [21] show that combining locally optimal solutions of the smaller social teams would give a globally optimal solution for the entire social network. Hence, we introduce a concept of social rarity that is unique to different cliques in the social graph and is easy to compute. We also incorporate the overall rarity of a chunk to get a fair representation of actual rarity.

As the allocation level of bandwidth for social peers increases and is actively used, the number of unchoke slots available for bartering with non-social swarm peers decreases. To determine what portion of its available upload bandwidth a client should allocate to social peers, SocialSwarm uses the average rarity of chunks across social peers as a heuristic. Effectively the assessment of the rarity of all chunks across social peers is how SocialSwarm determines the stage of the game.

A SocialSwarm client estimates the average social rarity for all chunks at each t_i by normalizing the social rarity of all individual chunks by the chunk count using equation (7).

$$R_A(t_i) = \frac{\sum_{c \in C} R_s(t_i, c)}{|C|} \quad (7)$$

A SocialSwarm client then allocates a certain maximum percentage of its bandwidth ($\text{MaxSocialBandwidth} = (1 - R_A(t_i))$) for use with its social peers. Using levels of altruism towards social peers, a SocialSwarm client will put its social peers into an ordered list. Starting at the top of this list (those peers with the highest altruism), a peer will unchoke its social peers one by one until either the predetermined $\text{MaxSocialBandwidth}$ percentage of upload capacity has fully been consumed by its social peers or a maximum limit on unchoked social peers is reached. This method ensures that the peers with the highest amount of aggregate social altruism—typically those peers who have a higher degree of connectivity—are allocated the greatest bandwidth and thus are potentially able to receive the file faster than peers with lower degrees of social connectivity.

All bandwidth not allocated or consumed by social peers is allocated to traditional BitTorrent unchoke slots (of reasonable size) and used for bandwidth bartering.

2) *Chunk Prioritization:* When social clients initially join swarms and when social bandwidth available from social peers is scarce, they must barter for bandwidth and chunks with non-social peers. Initially, clients will target chunks that are rare across non-social peers. As social peers acquire an increasing percentage of chunks, the average rarity of chunks across social nodes decreases and more bandwidth is allocated toward social purposes. As social peers increase their usage of this bandwidth, a client will increasingly target chunks that are rare across social peers (as opposed to chunks that are rare across non-social peers).

SocialSwarm is thus analogous to a real-world tribe or clan whose members initially barter with non-clan members

for goods not yet available in the clan. As more goods are obtained by clan members, they gradually decrease their external bartering and increase the amount of free sharing of goods within the clan. The amount of bartering with external, self-interested entities is thus determined by the availability of goods (chunks) within the clan. Here, availability is defined as both chunk possession and ability to share (bandwidth availability).

A SocialSwarm client accomplishes this collaboration by varying its calculation of chunk rarity based on the percentage of bandwidth actively being used by social peers, denoted by $B_s(t_i)$, and the percentage of bandwidth used by non-social peers, denoted by $B_n(t_i)$. Both $B_n(t_i)$ and $B_s(t_i)$ may be 0% concurrently if none of a node's bandwidth is being used by any of its peers.

The level of influence that social peers' chunk holdings exert over a node's concept of chunk rarity increases as the level of bandwidth sharing among social peers increases. When the majority of its bandwidth is used for bartering with non-social peers (when $B_n(t_i)$ is large), a SocialSwarm client will focus mostly on chunks that are rare across non-social nodes by making $R_n(t_i, c)$ dominant. Alternatively, when the majority of its bandwidth is used for collaboration with social peers (when $B_s(t_i)$ is large), a SocialSwarm client will focus mostly on chunks that are rare across social nodes by making $R_s(t_i, c)$ dominant. When little of its bandwidth is in use (when both $B_n(t_i)$ and $B_s(t_i)$ are small), a SocialSwarm client will use the traditional BitTorrent algorithm of focusing on chunks rare to the swarm overall by making $R_o(t_i, c)$ dominant.

Thus using its current $B_s(t_i)$ and $B_n(t_i)$ bandwidth percentages as weights, a SocialSwarm client combines the social, non-social, and overall rarities to form a combined weighted rarity for each given chunk as follows:

$$R_w(t_i, c) = R_n(t_i, c) * B_n(t_i) + R_s(t_i, c) * B_s(t_i) + R_o(t_i, c) * (1 - B_n(t_i) - B_s(t_i)) \quad (8)$$

A SocialSwarm client prioritizes the download of chunks from its connected peers based on their combined weighted rarity, $R_w(t_i, c)$. This allows a client to coordinate its collection of socially rare chunks with its social peers.

3) *Optimistic Unchoke Candidate Selection:* Typical BitTorrent implementations use either random selection or a longest-since-unchoke heuristic in deciding which peer should be optimistically unchoked for the next round. SocialSwarm instead probabilistically selects a peer out of a prioritized list ordered on availability of rare chunks at each peer. Thus a peer will target a peer with the largest group of *rare chunks* at each time interval t_i by calculating the level of rare chunks held by each peer using equation (9).

$$\text{For } p \in S \cup N, \\ H(t_i, p) = \frac{\sum_{c \in C} V(t_i, c, p) \times R_w(t_i, c)}{|C|} \quad (9)$$

Using its list of social peers ordered on $H(t_i, p)$, a peer will randomly choose the next peer for probabilistic unchoke using proportional selection based also on $H(t_i, p)$.

IV. IMPLEMENTATION AND TEST SETUP

In this section, we first present details of our SocialSwarm implementation. Second, we provide an overview of our test infrastructure. Next, we discuss the social network data set used to drive our tests, and finally, we analyze the performance of SocialSwarm in comparison with the standard BitTorrent protocol.

A. Implementation

We implement the SocialSwarm algorithm as an extension to the Rasterbar libtorrent library [6] version 0.13.1. Libtorrent is a library leveraged by a variety of different GUI- and text-based front ends to provide full BitTorrent functionality. Enhancing libtorrent with SocialSwarm as an extension allows SocialSwarm to be used with a variety of existing BitTorrent clients. To evaluate the SocialSwarm BitTorrent extension, we use an unmodified version of qBittorrent v1.1.0, a Qt-based libtorrent front end [26].

SocialSwarm-enabled libtorrent receives a list of known social peers, including relative SocialDistances for each peer and the peer’s most recent known global IP address. SocialSwarm compares its list of known social peer IP addresses with the IP addresses of each of the peers in BitTorrent swarms as received from the BitTorrent tracker to find social peers who are participating in each swarm. Once a social peer is identified, SocialSwarm-enabled libtorrent uses a new flag on the BitTorrent extended peer handshake to determine if the social peer is SocialSwarm-enabled. If a social peer is identified, but does not support the SocialSwarm protocol, then SocialSwarm libtorrent will treat the peer as a non-social peer. Apart from matching IP addresses and checking its SocialSwarm flag, SocialSwarm currently does no other social peer verification. A social network analyzer is developed to take a set of user interactions within the social network—in this case Facebook wall postings—and first calculate proportional levels of direct altruism between the Facebook users in the data set and then calculate levels of indirect altruism. More details are found in the next section.

B. Social Network Data Set

To evaluate SocialSwarm, we use an anonymized data set from interactions—wall postings—of 500 Facebook users [27]. For each social network member, we analyze the number of reciprocal wall postings within a given time period. Each pair of reciprocal postings is considered a single interaction. These interactions are used to determine the single-hop/direct levels of altruism between Facebook users. We use the inverse of this altruism as our single-hop/direct SocialDistance between peers. Using these single-hop SocialDistances, we calculate the multi-hop altruism and SocialDistances using a *HopDecay* of 0.95 with the method described in Section III. After constructing the social tree for all users in the Facebook

data set, we choose a single peer and do a breadth first walk over the tree until a total of 500 social peers is selected (traversed). We then assign each of these 500 social peers to a virtual SocialSwarm client for our evaluation. Each node has knowledge only of its own social peers (rather than global knowledge) and considers all other nodes outside of its maximum SocialDistance as “non-social.”

Each client is assigned a unique virtual machine with a unique IP address. We make the assumption that the social network analyzer has a method of determining a public IP address for known social peers. This IP address determination would occur either via extraction from existing social network interactions or some extension to those interactions that enables extraction of peer IP addresses for direct and indirect social peers. IP address identification of users is dependent on the social network being used. Email, for example, commonly includes the IP address of the original sender as one of the headers. This is also true for certain webmail systems, such as Hotmail and Yahoo mail. There are methods of obtaining users’ IP addresses on MySpace [28], [29] and until recently, IP addresses of Facebook users could be directly extracted from the email notifications sent on activity between social peers, such as wall postings [30]. This information, when coupled with the peer interaction data set, could uniquely give the IP addresses of the users helping to bootstrap our social network. To incorporate dynamically changing IP addresses, Koolean [31] proposes a solution whereby each user is associated with a permanent identifier and coupled with a challenge-response mechanism with the social peers, the user is verified, and the current IP address of the user is deemed authentic. The updated connection details of the newly joined peer are distributed across the social network. Because the main focus in our work is to demonstrate faster file distribution, we have not yet incorporated automatic identification of social peer IP addresses into the social network analyzer.

The 500 users in our experiments were extracted from a much larger data set of Facebook wall postings which contained over 43,000 nodes in total. We thus utilized less than 1.2% of the total social network data set. Unlike our evaluation nodes, real-world users of SocialSwarm would not be restricted to searching for social peers among an isolated group of 500 nodes, but rather could search for and utilize any available peers on their full social network within their chosen max SocialDistance. For example, by increasing the number of social network nodes that we included in our analysis to 5000 and given no variation in maximum SocialDistance, we found a 55% average increase in the number of peers a node would consider as its peers. Fig. 2 shows the number of single-hop/direct and multi-hop peers each node considers as part of its social network given a maximum SocialDistance. Fig. 3 shows the number of peers each node considers as part of its social network when the subset of the data is expanded to include 5000 nodes total. Although our experiments constrained each node to search only for social peers within a very small subset of the social network, and given the differences between Fig. 2 and Fig. 3, it is clear that in real-world deployments of

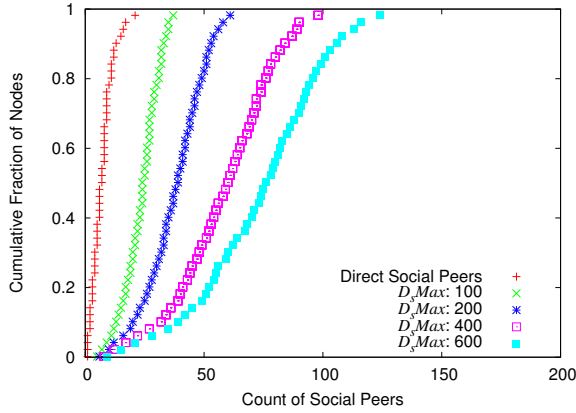


Fig. 2: CDF of Social Peer Count for 500 Nodes

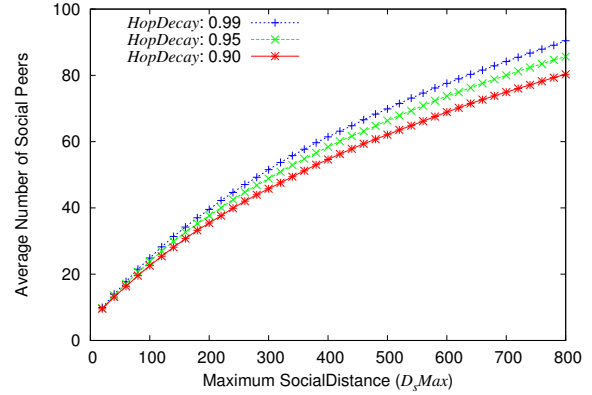


Fig. 4: Number of Social Peers in Network Based on Max SocialDistance

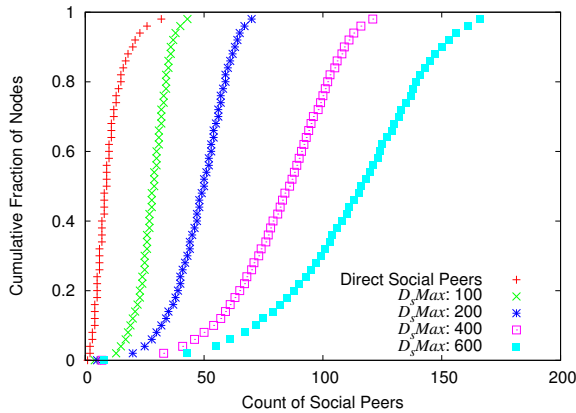


Fig. 3: CDF of Social Peer Count for 5000 Nodes

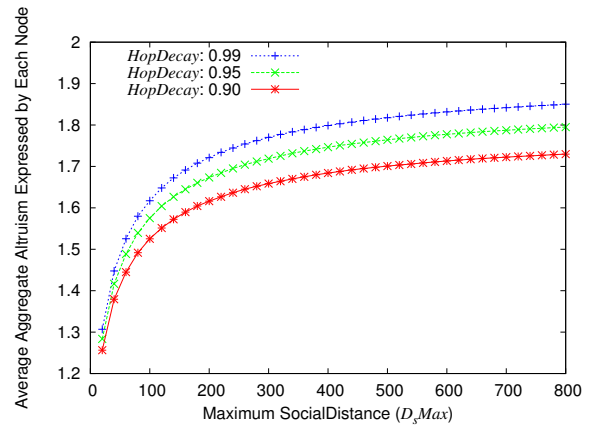


Fig. 5: Average Aggregate Altruism from Each Peer Based on Maximum SocialDistance

SocialSwarm nodes would have many additional peers from which to choose.

A peer continues a shortest-path-first search of its social network adding new social peers until some maximum allowable SocialDistance is reached. Based on this Facebook data set, Fig. 4 shows the average number of social peers each user has in relation to the maximum allowable SocialDistance. Although the number of additional peers continues to increase at a reasonable rate with greater SocialDistances, the relative altruism expressed toward those additional peers significantly degrades. Fig. 5 shows the average altruism expressed by each node to its social peers as the maximum SocialDistance is varied. This figure shows a drop off in altruism increase beyond a maximum SocialDistance of 150 to 200.

It is important to note that SocialSwarm is not dependent on any specific social peers being online or available, SocialDistance is used for prioritizing bandwidth offered. Any unused bandwidth due to offline peers will be offered to other social peers who are online, even if they have a higher SocialDistance. If no social peers are online or if those that are online are not consuming bandwidth, then SocialSwarm will revert to interacting with non-social peers exclusively.

C. Test Infrastructure

Our testbed consists of 20 high-performance servers. Each server has 24GB of RAM and 8 Intel-based Xeon CPU cores (two quad core Xeon L5420 processors per system). All servers are fully connected through a gigabit switch with a fully connected 68Gbps back-plane. A torrent tracker is run on a separate machine that also has full gigabit connectivity to the same network switch. On each server, we create 25 virtual clients for a total of 500 virtual clients. Each client runs Debian Linux version 5.0.3 inside of an OpenVZ virtual container. The storage for each virtual container is located on a set of high performance SAN arrays with 15K rpm drives to ensure that the probability of I/O contention is minimized. Network tuning and shaping is put in place so that each virtual client can be tuned independently with each of the following metrics: incoming maximum throughput, outgoing maximum throughput, incoming packet latency, and outgoing packet latency.

CPU, network, memory, and disk I/O are monitored on all servers to ensure that resource contention between the virtual environments did not occur.

TABLE III: Baseline Test Parameters

Parameter	Value
File Size	25 MB
RTT Inter-peer Latency	48 ms
Altruism <i>HopDecay</i>	0.95
Maximum SocialDistance (<i>D_sMax</i>)	400
Maximum Number of Concurrently Unchoked Social Peers	30
Leeches (Non-contributing Peers)	0
Seed Bandwidth	2.5MB/sec

V. EVALUATION

We evaluate SocialSwarm using the infrastructure described in section IV-C. We now provide an overview of our testing methodology, followed by the results of our evaluation.

A. Evaluation Methodology and Criteria

In order to evaluate the flash-file distribution speeds of SocialSwarm in comparison with standard BitTorrent, we pre-load a single peer in the system with the file contents, making it the sole seed for the system. We then start all clients within 10 seconds of each other. We assume that in real-world use, external mechanisms for communicating torrent availability and automatic triggering of swarm participation exist. Likely such a mechanism would use messaging capabilities of the social networks themselves. In all experiments, we use the parameters, shown in Table III, as input to each of the tests, unless otherwise specified.

The network configuration (including latency between peers) was made independent of each node’s social connectivity. A vast majority (489 peers) of the 500 peers are assigned a maximum upload bandwidth of 256Kbit (32KB) per second and a maximum download bandwidth of 1Mbit (128KB) per second. The remaining 11 peers are assigned a maximum upload bandwidth of 2.5MB/second and a maximum download bandwidth of 5MB/second. One of these 11 high-speed peers is chosen as the seed. These bandwidth capabilities attempt to simulate a mix of home users with slower Internet connections combined with a few corporate/educational/FTTH (fiber-to-the-home) users (including the seed) with much faster Internet connections [32]. Each data point provided in this section represents an average across 10 runs, with each run using an identical configuration of nodes including seeds.

B. Comparison of Basic Download Time

In this section, we evaluate the average download time of SocialSwarm compared to that of standard BitTorrent. One of our first tests is to compare the average time required for a single file to be dispersed to all participating peers.

Fig. 6 provides a cumulative density function (CDF) of the 500 peer file distribution time for a fully socially enabled run as well as a non-socially enabled run.

As shown in Table IV, the average download time of SocialSwarm for the 499 peers is reduced by 25.7% compared to BitTorrent. The performance gain (33.5%) for the most

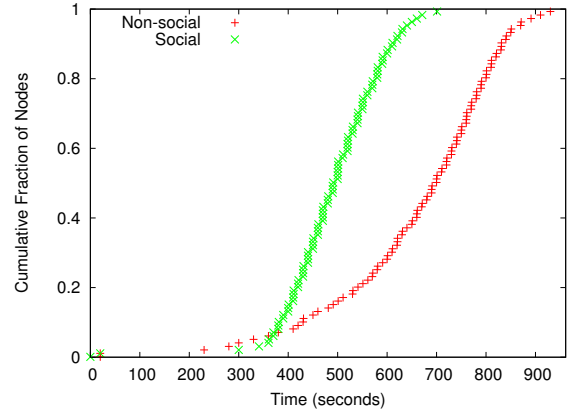


Fig. 6: Social vs Non-social CDF of 25MB file and 0 Leeches

TABLE IV: Average Download Time & Percent Improvement with a 99% confidence interval

	Non-social	Social	Top 1% Social	Bottom 1% Social
Download Time (sec)	654±11	486±3	435±17	551±22
Improvement (%)	base	25.7	33.5	15.7

socially connected peers (top 1%) is greater than the one (15.7%) of the least socially connected peers (bottom 1%).

Fig. 7 shows the average download rate per peer over time. The first minute or so of the experiment shows a significant spike and fluctuation in the download rate for all peers. This is due to the fact that all peers are initiating connections with the tracker as well as with each other. All peers are sharing chunk availability maps with every other peer with which they initiate connections. After about 180 seconds, the non-social peers level out in their sustained bandwidth usage. The social peers, however, slowly allocate more bandwidth to social peers as the average social rarity of chunks decreases; this is shown in Fig. 8. It is this bandwidth surge—the peak of which is

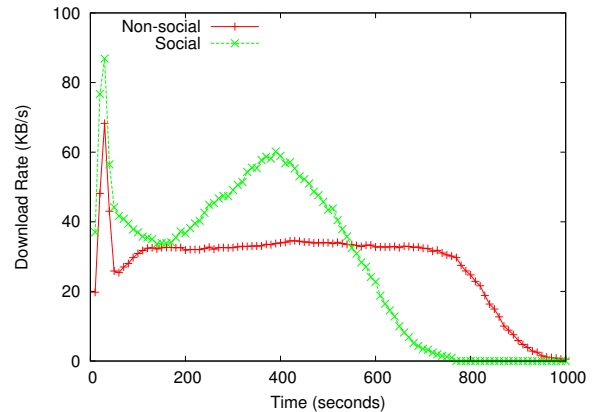


Fig. 7: Client Download Rate Comparison

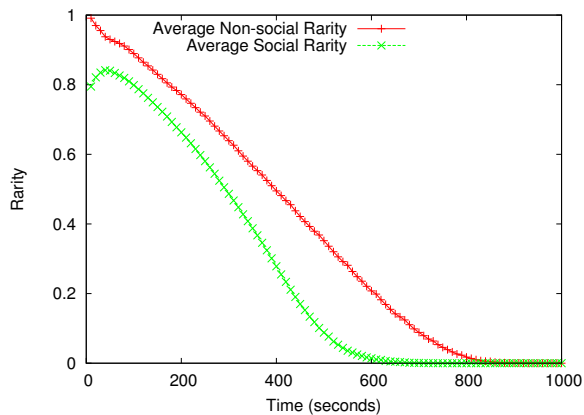


Fig. 8: Chunk Rarity Reduction Comparison

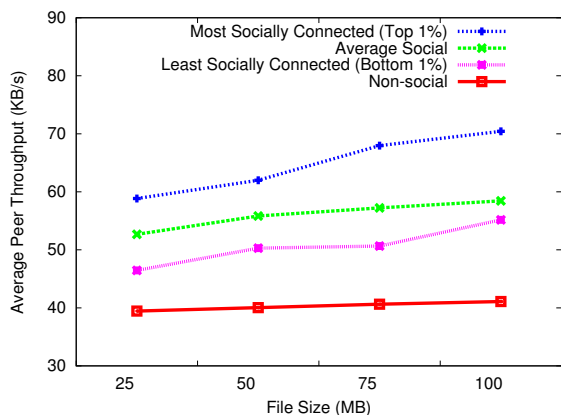


Fig. 9: Effect of File Size on Peer Throughput

around 400 seconds into the test—that allows social peers to complete earlier and turn into seeds earlier.

The results of the first 60 seconds of average social rarity are inaccurate due to lack of social peer and chunk availability information during system start-up and initialization. Social peers must find and establish connections with other social peers, then receive piece availability bitmaps from those social peers before declaring that chunks are truly socially rare.

These figures show that for both social and non-social swarms, rarity of chunks nears zero around 200 seconds before the download rate nears zero. This is because rarity is averaged across both downloading clients and seeds. Swarm participants that become seeds earlier do not necessarily decrease the average bandwidth used per node. The unchoke slots vacated by these newly formed seeds are quickly reoccupied by other peers, and the new seeds reduce the average rarity of chunks.

C. Effect of File Size

In order to see the impact of file size on the performance of flash file distribution, we use four different file sizes from 25M to 100M, increased by 25M, as shown in Fig. 9. The x-axis represents the file size and the y-axis shows the average peer throughput (KB/s). With an increase of file size from 25MB

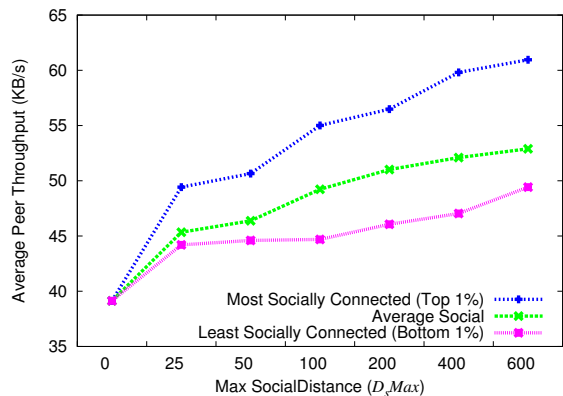


Fig. 10: Effect of Maximum SocialDistance on Peer Throughput

TABLE V: Average Download Time and Improvement for Two Seeds

	Non-Social	Social
1 Seed (sec)	654	491
2 Seeds (sec)	649	486
Improvement (%)	0.6	1.1

to 100MB the performance of standard BitTorrent increased by 4% on average and the performance between social nodes increased by 9% on average.

It is observed that the greatest increase in bandwidth is realized by the most socially connected peers. The 1% of peers in the system with the highest degree of social connectivity realized a 16% increase in performance between a 25MB file and a 100MB file.

D. Maximum SocialDistance

Maximum SocialDistance (D_{sMax}) is one of the important parameters in SocialSwarm. By way of review, this is the maximal SocialDistance whereby a peer would consider a peer to be part of its social network. Maximum SocialDistance can thus be considered as a radius from a peer to the perimeter of its social network.

Fig. 10 shows the average per peer throughput as maximum SocialDistance is increased. A maximum SocialDistance of 0 is effectively the same as disabling the SocialSwarm protocol. It can be seen that even low maximums of SocialDistance—such as 25—yield considerable improvements in per-client throughput compared with non-social clients. As bandwidth utilization improves while increasing the maximum SocialDistance, the percentage of improvement decreases at each step.

E. Effect of Additional Seed Capacity

Table V shows the negligible effect of adding a second high bandwidth seed into the system. This reinforces the fact that BitTorrent’s performance is much more dependent on p2p bandwidth and unchoke slot availability than on seed bandwidth.

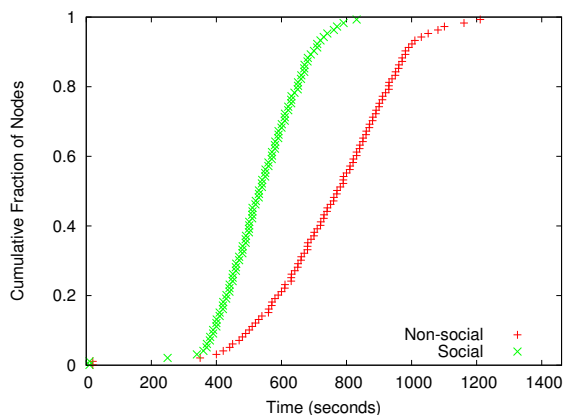


Fig. 11: Social vs Non-social CDF of 25MB File and 100 Leeches

TABLE VI: Average Download Time Based on # of Leeches

Leech Count	Non-Social Time (sec)	Social Time (sec)
0	658	489
100	755	535
200	870	566

F. Effect of Leeches

We conduct several experiments to identify how SocialSwarm compares with standard BitTorrent when faced with varied numbers of non-social leeches (additional peers each consuming bandwidth from the system but not contributing reciprocally). We make the assumption that unless they are infected with malware, SocialSwarm-enabled peers will typically behave properly and share their bandwidth resources altruistically with their social peers (and not leech bandwidth from social peers).

As shown in Fig. 11, the CDF between social and non-social torrent download times follows the same pattern as the baseline 25MB tests (Fig. 6). Table VI compares the average download time between the base run of 0 leeches with the download time when 100 and 200 leeches are present respectively.

Fig. 12 shows the relative throughput degradation as number of leeches is increased (from 0 to 100 and from 0 to 200). The throughput percentages in this graph are relative to runs without leeches in the swarm. Thus, although non-social swarms have a lower throughput than social swarms, the 0 leech mark in this graph is shown at 100%, representing no performance degradation in comparison with each swarm type's base case. Leeches are intentionally configured with a very small level of upload bandwidth capacity. Leeches are added to the swarm before the 500 peers are started. This is done with the goal of intentionally establishing connections to and consuming bandwidth from the seed before other nodes start. In the case of socially enabled swarms, we assume that although leeches may have social relationships with each other, they have no social relationships to other peers within the swarm. It is clear that non-socially enabled peers have the greatest performance degradation (25%) when faced with leeches. The most socially

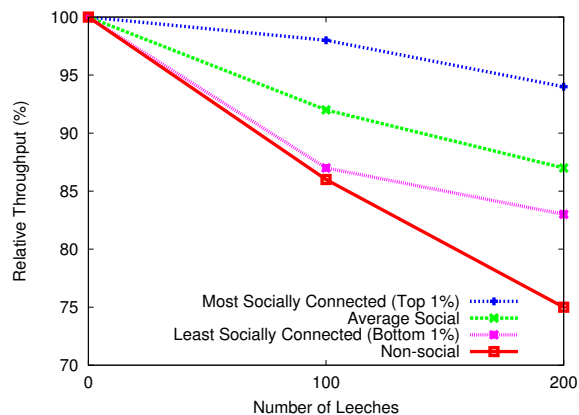


Fig. 12: Effect of Leeches on Received Bandwidth

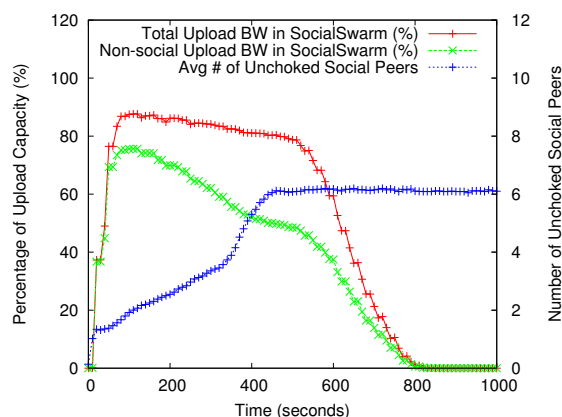


Fig. 13: Bandwidth Allocation & Social Unchokes

connected peers have the least performance degradation (6%). This performance degradation delta is attributed to the fact that peers with higher levels of social connectivity have a larger number of peers with which they may altruistically share bandwidth. Based on the assumption that social peers are less likely to exhibit malicious behavior than unknown non-social peers, SocialSwarm clients target known social peers when deciding those peers with which to establish outgoing and incoming connections. This may increase aversion to leeches. By unchoking a higher number of peers concurrently in comparison with standard BitTorrent, SocialSwarm distributes the upload and download bandwidth used across a larger number of peers thus diversifying the risk that any individual malicious peer might adversely affect a client's performance.

G. Bandwidth Contribution and Unchoke Slot Allocation

Fig. 13 shows, for a given SocialSwarm, the average percentage of bandwidth used for interacting with non-social peers and, stacked on top of that, the additional percentage of bandwidth used for interacting with non-social peers. This figure shows that SocialSwarm does not replace interaction with non-social peers but rather increases the percentage of bandwidth utilized on each peer. Fig. 13 also shows the

average number of social peers a node will unchoke over time. An offer of bandwidth to a social peer in no way guarantees that the bandwidth will actually be used by the offer recipient. The number of social peers that actively use the offered bandwidth is lower than the number of nodes, also shown in Fig. 13, that are offered bandwidth. Clearly, out of the total number of social peers a node might have—as shown in Fig 2—only a very small percentage of those peers would need to be online to allow SocialSwarm to be effective.

VI. CONCLUSIONS AND FUTURE WORK

In this paper, we introduced SocialSwarm, a novel approach to flash file dissemination that exploits SocialDistance, which we extract from altruism between social peers, so as to relax the required, but inefficient, reservation of bandwidth for resource bartering in BitTorrent. We implemented SocialSwarm as an extension to the libtorrent library, applied a social network topology and interaction history obtained from Facebook, and evaluated it on a testbed of 500 independent virtual clients. We showed that SocialSwarm reduces average file download time by 25% to 35% compared to that of standard BitTorrent under varied conditions—file sizes, max SocialDistance, and leech and seed counts.

In the future, we will investigate the effect of socially enabled leeches on SocialSwarm. Given that malicious code commonly uses social networks for propagation, clusters of social peers have the possibility of becoming infected. Our future work will also include finding a dynamic way to modify peer SocialDistance/altruism levels based on observed behavior between individual peers as well as among clusters of social peers.

In our work so far, we have approximated altruism to be proportional to levels of reciprocal interaction between peers. Our future work will include investigation of alternative methods of estimating direct altruism among social peers. We will also evaluate the effect of a varied *HopDecay* by each SocialSwarm participant.

REFERENCES

- [1] D. Brumley, P. Poosankam, D. Song, and J. Zeng, "Automatic patch-based exploit generation is possible: Techniques and implications," in *IEEE Symposium on Security and Privacy*, May 2008.
- [2] I. Norros, B. Prabhu, and H. Reittu, "On uncoordinated file distribution with non-altruistic downloaders," *Managing Traffic Performance in Converged Networks*, pp. 606–617, 2007.
- [3] D. Levin, K. LaCurts, N. Spring, and B. Bhattacharjee, "Bittorrent is an auction: analyzing and improving bittorrent's incentives," *SIGCOMM Computer Communication Review*, vol. 38, no. 4, pp. 243–254, 2008.
- [4] B. Cohen, "Incentives build robustness in BitTorrent," in *2003 Workshop on Economics of Peer-to-Peer Systems (P2P Econ'03)*, 2003.
- [5] M. Piatek, T. Isdal, T. Anderson, A. Krishnamurthy, and A. Venkataramani, "Do incentives build robustness in bittorrent," in *2007 Symposium on Networked Systems Design and Implementation (NSDI'07)*, 2007.
- [6] A. Norberg, "The opensource libtorrent library." [Online]: <http://www.rasterbar.com/products/libtorrent/>
- [7] C. Gkantsidis, T. Karagiannis, and M. Vojnovic, "Planet scale software updates," *SIGCOMM Computer Communication Review*, vol. 36, no. 4, pp. 423–434, 2006.
- [8] L. J. Camp and A. Friedman, "Good neighbors can make good fences: A peer-to-peer user security system," in *Telecommunications Policy and Research Conference*, Sep 2004.
- [9] S. Shakkottai and R. Srikant, "Peer to peer networks for defense against internet worms," in *2006 Workshop on Interdisciplinary Systems Approach in Performance Evaluation and Design of Computer & Communications Systems (Inter-Perf'06)*, 2006.
- [10] M. Deshpande, B. Xing, I. Lazardis, B. Hore, N. Venkatasubramanian, and S. Mehrotra, "Crew: A gossip-based flash-dissemination system," in *2006 IEEE International Conference on Distributed Computing Systems (ICDCS'06)*, 2006.
- [11] P. G. Lind, L. R. da Silva, Jr, and H. J. Herrmann, "Spreading gossip in social networks," *Physical Review E (Statistical, Nonlinear, and Soft Matter Physics)*, vol. 76, no. 3, 2007. [Online]: <http://dx.doi.org/10.1103/PhysRevE.76.036117>
- [12] T. Locher, P. Moor, S. Schmid, and R. Wattenhofer, "Free riding in bittorrent is cheap," in *2006 ACM Workshop on Hot Topics in Networks (HotNets'06)*, 2006.
- [13] P. Shah and J.-F. Paris, "Incorporating trust in the bittorrent protocol," in *2007 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS'07)*, 2007.
- [14] M. Piatek, T. Isdal, A. Krishnamurthy, and T. Anderson, "One hop reputations for peer to peer file sharing workloads," in *2008 USENIX Symposium on Networked Systems Design and Implementation (NSDI'08)*, 2008.
- [15] Z. Zhu, G. Cao, S. Zhu, S. Ranjan, and A. Nucci, "A social network based patching scheme for worm containment in cellular networks," in *2009 IEEE Conference on Computer Communications (INFOCOM'09)*, 2009.
- [16] A. Friedman, "Good Neighbors Can Make Good Fences," *IEEE Technology and Society Magazine*, vol. 278, no. 0079/07, 2007.
- [17] P. Garbacki, A. Iosup, D. Epema, and M. van Steen, "2fast: Collaborative downloads in p2p networks," in *2006 IEEE International Conference on Peer-to-Peer Computing (P2P'06)*, 2006.
- [18] V. Vishnumurthy, S. Chandrakumar, and E. Siler, "KARMA: A secure economic framework for peer-to-peer resource sharing," in *Workshop on Economics of Peer-to-Peer Systems (P2P Econ'03)*, 2003.
- [19] J. A. Pouwelse, P. Garbacki, J. Wang, A. Bakker, J. Yang, A. Iosup, D. H. J. Epema, M. Reinders, M. R. V. Steen, and H. J. Sips, "Tribler: A social-based peer-to-peer system," in *2006 International Workshop on Peer-to-Peer Systems (IPTPS'06)*, 2006.
- [20] O. Ozkasap, M. Caglar, and A. Alagoz, "Principles and performance analysis of second: A system for epidemic peer-to-peer content distribution," *Journal of Network and Computer Applications*, vol. 32, no. 3, pp. 666–683, 2009.
- [21] G. Karame, M. Cagalj, and S. Capkun, "Small coalitions: Lightweight collaboration for efficient p2p downloads," *2009 IEEE International Symposium on Network Computing and Applications (NCA'09)*, 2009.
- [22] R. Izhak-Ratzin, N. Liogkas, and R. Majumdar, "Team incentives in bittorrent systems," in *2009 IEEE International Conference on Computer Communications and Networks (ICCCN'09)*, 2009.
- [23] W. Galuba, K. Aberer, Z. Despotovic, and W. Kellerer, "Self-organized fault-tolerant routing in peer-to-peer overlays," in *2010 IEEE Consumer Communications and Networking Conference (CCNC'10)*, 2009.
- [24] S. Marti, P. Ganesan, and H. Garcia-Molina, "Sprout: P2p routing with social networks," in *2004 International Conference on Extending Database Technology (EDBT'04)*, 2004.
- [25] F. Walter, S. Battiston, and F. Schweitzer, "A model of a trust-based recommendation system on a social network," *Autonomous Agents and Multi-Agent Systems*, vol. 16, no. 1, pp. 57–74, 2008.
- [26] "qbittorrent client." [Online]: <http://qbittorrent.sourceforge.net/>
- [27] B. Viswanath, A. Mislove, M. Cha, and K. P. Gummadi, "On the evolution of user interaction in Facebook," in *2009 ACM Workshop on Online Social Networks (WOSN'09)*, 2009.
- [28] "Myspace trackers." [Online]: <http://myspacetracker.blogspot.com/>
- [29] J. Cheng, J. Hoffman, T. LaMarche, A. Tavil, A. Yavad, and S. Kim, "Forensics tools for social network security solutions," 2009.
- [30] X. Jardin, "Yet another Facebook privacy risk: Emails Facebook sends leak user IP address," 2010. [Online]: <http://boingboing.net/2010/05/07/yet-another-privacy.html>
- [31] S. Koolen, "Creating and maintaining relationships in social peer-to-peer networks," *Delft University of Technology*, 2007.
- [32] "Industry Analysis and Technology Division - Wireline Competition Bureau of the US FCC", "High-speed services for internet access," 2010. [Online]: http://www.fcc.gov/Daily_Releases/Daily_Business/2010/db0722/FCC-10-129A7.pdf