Inter-swarm resource allocation in BitTorrent communities

Mihai Capotă*, Nazareno Andrade*†, Tamás Vinkó*, Flávio Santos‡, Johan Pouwelse* and Dick Epema*
*Delft University of Technology, The Netherlands
†Universidade Federal de Campina Grande, Brazil
‡Universidade Federal do Rio Grande do Sul, Brazil

***Abstract*—A considerable body of research shows that BitTorrent provides very efficient resource allocation inside single swarms. Many BitTorrent clients also allow users to participate in multiple swarms simultaneously, and implement inter-swarm resource-allocation mechanisms that are used by millions of people. However, resource allocation across multiple swarms in BitTorrent has received much less attention. In this paper, we investigate whether currently prevalent inter-swarm resource allocation mechanisms perform acceptably or call for improvements. We use data from two BitTorrent communities and present results from trace-based simulations. Two use-cases for allocation mechanisms drive our evaluation: (1) file-sharing communities, whose objective is maximizing throughput, and (2) video-streaming communities, whose objective is maximizing the number of users receiving sufficient resources for uninterrupted streaming. To put the results from the analyzed mechanisms into perspective, we devise theoretical efficiency bounds for inter-swarm resource allocation, for which we map the resource allocation problem to a graph-theoretical flow network problem. In this formalism, the goal of the file-sharing use-case, throughput maximization, is equivalent to maximizing the flow in the network. The goal of the video-streaming use-case translates into finding a max-min fair allocation for BitTorrent downloading sessions, a problem for which we devise a new algorithm.**

I. Introduction

A large body of research (e.g., [1], [2]) shows that BitTorrent provides resource allocation mechanisms to create efficient and scalable peer-to-peer swarms for content distribution. However, nearly all evidence of BitTorrent's efficiency has been found exclusively in the context of single-swarm operation. At the same time, measurements show that most BitTorrent users participate in multiple swarms, or *torrents*, simultaneously [3]. It is customary for users to download, or *leech*, multiple files concurrently, and to continue uploading the files they finished downloading, or *seed*, at the same time. In order to enable this multiple-swarm operation, designers of BitTorrent clients have introduced two mechanisms to perform *inter-swarm* resource allocation: the selection of torrents to seed in, and the allocation of upload bandwidth across all torrents in which a peer participates, either as a seeder, or as a leecher. Although these mechanisms are routinely used by millions of users, it is presently unknown how they perform, and whether they can be improved upon. In this paper, we provide a thorough analysis of their performance.

The authors would like to thank Adele Lu Jia for her inspiring suggestions. This work was funded by the EU Seventh Framework Programme projects P2P-Next and QLectives (grants 216217 and 231200). Nazareno Andrade is also supported by the Brazilian National Council of Technological and Scientific Development (CNPq), through grant 480855/2010-2.

The evaluation of inter-swarm resource allocation mechanisms presumes a theoretical understanding of the problem they address. We thus build a foundation for this work by formally defining the inter-swarm resource allocation problem (Section II). Our formalization shows that this problem is NP-hard. We find that to make it tractable, it is necessary to divide it into two parts, torrent selection and bandwidth allocation, similarly to what has been done in BitTorrent clients. Such a relaxation allows us to derive theoretical upper bounds for the performance of resource allocations in a BitTorrent community–a group of users that access torrents through the same site [2], [4]. This is done by modeling the community as a flow network and using graph theory (Section IV). In this work, we consider two types of communities: (1) file-sharing communities, which target maximizing aggregate download speed, and (2) video-streaming communities, which target maximizing the number of users receiving sufficient download speed for streaming. The performance upper bound for a file-sharing community is obtained by solving the maximum flow problem; for video-sharing communities, we introduce a novel algorithm to find the max-min fair allocation of flow.

The theoretical results provide the context for the evaluation of the inter-swarm resource allocation mechanisms presently deployed in BitTorrent clients. Our analysis focuses on the mechanisms of uTorrent and Azureus, two clients that together account for 80% of BitTorrent's usage [2]. We describe their behavior and recreate it with simulators for torrent selection and bandwidth allocation (Section III). To obtain realistic results, we use traces of real-world BitTorrent usage from two BitTorrent communities to drive our simulations. Each trace captures the behavior of nearly 100 000 users over several months, representing large-scale real instances of the inter-swarm resource allocation problem (Section V).

Our results (Sections VI and VII) show that it is possible to significantly improve current inter-swarm resource allocation mechanisms, but that efforts should focus on bandwidth allocation. The present torrent selection mechanism does not hamper performance, partly because of a reduced space of possibilities for selection algorithms in practice. In contrast, the current bandwidth allocation mechanism performs poorly compared to the optimal: as low as 50% for file sharing, and 25% for video streaming.

II. The inter-swarm resource allocation problem

In this section we formulate and formalize the problem of inter-swarm resource allocation in BitTorrent. We first describe the need for inter-swarm resource allocation. Next, a formulation of the general resource allocation problem is presented. Finally, we define the two goals for resource allocation that are considered in this paper.

A. Background

Consider a BitTorrent community with a set of users I and a set of torrents T. A user participating in multiple torrents simultaneously is said to have a *session* in each of these torrents, and these torrents are said to form the user's *active torrents* set. Users downloading a torrent are called *leechers*, and their sessions are called *leeching sessions*, while users that have finished downloading a torrent and are only uploading it are called *seeders*, and their sessions are called *seeding sessions*. We denote by S_t and L_t the sets of seeders and leechers in a torrent t, respectively.

Efficiently participating in a torrent requires a BitTorrent client to maintain a certain number of open TCP connections. However, too many simultaneous connections may reduce the overall upload or download speed of the client. Therefore, most BitTorrent clients have a configuration parameter to set the maximum number of active torrents, thus limiting the total number of open TCP connections in the client. Furthermore, limiting the number of active torrents ensures that disk activity is also limited, thus preventing disk trashing.

A BitTorrent client decides which torrents to keep active at any given moment. Clients usually always keep their leeching sessions active. If a client has fewer leeching sessions than the maximum number of active torrents, it will start seeding sessions in torrents from its *seeding library*, i.e., the set of torrents that have been fully downloaded and that have not been removed. The seeding library of user i is denoted by Λ_i. The maximum number of seeding sessions of a client, called its *seeding capacity* and which is denoted k_i, is determined by the maximum number of active torrents, the number of leeching sessions, and the size of the seeding library.

B. Problem formulation

An *inter-swarm resource allocation* represents the decisions of all users in the community about which leeching sessions they serve, and how much bandwidth they assign to each of these leeching sessions. An inter-swarm resource allocation must satisfy two constraints. Constraint C1 is the *seeding capacity constraint*: users cannot exceed their seeding capacity k_i, and Constraint C2 is the *bandwidth constraint*: users cannot offer more bandwidth than their upload bandwidth μ_i, and they cannot accept more bandwidth than their download bandwidth δ_i. More formally, a resource allocation is a function $\mathcal{A} : P \to \mathbb{R}$ which satisfies:

(C1) $\forall i \in I : \left|\{t \in \Lambda_i \mid \exists j \in L_t \text{ s.t. } \mathcal{A}(i,j,t) > 0\}\right| \leq k_i$; and

(C2) $\forall i \in I$: $\sum_{t \in T, j \in L_t} \mathcal{A}(i,j,t) \leq \mu_i \wedge$
$\forall j \in I$: $\sum_{t \in T, i \in S_t \cup L_t} \mathcal{A}(i,j,t) \leq \delta_j$,

where P is the set of triplets (i,j,t) representing the potential data transfer connections, $P := \{(i,j,t) \in I \times I \times T \mid (i \in S_t \vee i \in L_t) \wedge j \in L_t\}$.

The **Resource Allocation Problem** (RAP) is finding a resource allocation that achieves a specific goal. Such a goal can be either the optimization of a metric or satisfying a set of constraints. The RAP for a particular community may be to maximize the total throughput in the community, whereas another community may be interested in maximizing the median download speed across all leeching sessions. Regarding constraint satisfaction, there can be communities interested in guaranteeing a certain minimum download speed for all users, or communities aiming at achieving some form of max-min fairness. We give precise definitions for goals we consider in the next subsection.

In general, a RAP is a mixed-integer (non-)linear optimization problem, which is, regardless of the goal, NP-hard. Nevertheless, it is possible to divide RAP into two subproblems, which are the seeding sessions allocation problem and the bandwidth allocation problem. This decomposition may lead to an approximative solution for a RAP, and maps parts of the problem to more tractable equivalents.

The **Seeding Sessions Allocation Problem** (SSAP) consists in selecting a subset P_s of P, such that the number of torrents in which any seeder uploads does not exceed its seeding capacity: $\forall i \in I : \left|\{t \in \Lambda_i \mid \exists j \in L_t \text{ s.t. } (i,j,t) \in P_s\}\right| \leq k_i$. Solving SSAP yields a set of possible data transfer connections P_s that satisfies the seeding capacity constraint C1. This corresponds to the torrent selection done by BitTorrent clients.

We define a *bandwidth allocation* as a function $\mathcal{B} : P_s \to \mathbb{R}$ such that the bandwidth constraint C2 holds. Because of the definition of P_s, the bandwidth allocation also satisfies C1. The **Bandwidth Allocation Problem** (BAP) is then finding a bandwidth allocation that achieves a RAP goal. BAP is a tractable relaxation of RAP. It is possible to map BAP to equivalent problems in flow networks, as we show in Section IV.

Framing a RAP as being composed of SSAP and BAP also allows us to derive upper bounds for its solution. Applying an algorithm that optimally solves BAP to the complete set P will produce an upper bound to RAP. This is equivalent to relaxing RAP by ignoring C1. Although this upper bound is not necessarily a feasible solution of RAP, it can be used as a reference for the performance of heuristic solutions.

C. Maximizing download speed and optimizing streaming

We consider two goals for BitTorrent communities in this paper. The first one is suitable for a community interested in maximizing the average download speed of its users. This goal, named *Maximum throughput*, is in line with many existing file-sharing communities. It is formally defined as finding an allocation $\mathcal{A}$ that maximizes $\sum_{(i,j,t) \in P} \mathcal{A}(i,j,t)$.

The second goal reflects the requirements of video-streaming systems. In this case, the community intends to provide as many users as possible with enough download speed for streaming. One way to formalize this objective is to aim at providing a certain minimum streaming rate to as many leeching sessions as possible. However, we opt for a stronger formulation named *Max-min fairness*. In a *max-min fair allocation*, the download speed of a leeching session can only be increased at the cost of decreasing the download speed of another leeching session that has a lower speed. Formally, an allocation $\mathcal{A}$ is max-min fair iff

$\forall \mathcal{A}'$: if $\exists p \in P$ s.t. $\mathcal{A}'(p) > \mathcal{A}(p)$ then $\exists q \in P$ s.t. $\mathcal{A}(q) \leq \mathcal{A}(p)$ and $\mathcal{A}'(q) < \mathcal{A}(q)$.

Intuitively, the allocation should provide the highest possible streaming rate for the lowest-bandwidth user, then the highest possible streaming rate for the second lowest-bandwidth user and so on. With the resulting allocation, users that download at a rate lower than the streaming rate will experience some startup delay, but will still have the best possible quality of experience. Furthermore, max-min fairness enables the community to work with multiple streaming rates of varying qualities and to minimize the number of users experiencing low-quality streams.

III. Simulators for current RAP solutions

We now describe in detail the inter-swarm resource allocation mechanisms currently deployed in BitTorrent clients. In addition, we introduce simulators that replicate their behavior, and present a simulator validation experiment.

A. *Current mechanisms in BitTorrent clients*

Current BitTorrent communities tackle RAP in a decentralized manner using various heuristics implemented by BitTorrent clients. We investigate for this analysis the two most popular clients, *uTorrent* and *Azureus*, which account for 80% of BitTorrent usage [2]. Examining the configuration and documentation of these clients shows that they solve SSAP and BAP using a similar behavior.

With regard to SSAP, or torrent selection, these clients employ a heuristic based on the proportion of leechers in each swarm. First, all torrents in the user's seeding library are sorted according to their proportions of leechers. Then, the torrents with the highest proportions of leechers are chosen to be part of the active torrents set. The torrents that fall outside the seeding capacity are paused. This heuristic relies on the assumption that the proportion of leechers is a good approximation for the bandwidth need in a torrent. Note that this heuristic does not take into account the bandwidth of seeders and leechers. However, the impact of this omission on the quality of the solution will depend on the problem instance at hand.

The solution to BAP, or bandwidth allocation, involves three steps. First, the client allocates the same number of upload connections to each active torrent, five by default. Second, the connections are allocated to leechers inside each torrent. This is done differently by seeders and leechers. Seeders allocate connections in a round-robin fashion to all leeching sessions. Leechers allocate one connection randomly in order to bootstrap the discovery of new peers. The rest of the connections are allocated to the fastest reciprocating peers following a tit-for-tat strategy. In the third step, after connections are allocated, each upload connection receives an equal share of the peer's total upload bandwidth.

There are various reasons for the current BAP solution. The equal division of connections across torrents stems from the assumption that a small fixed number of connections is sufficient for good performance in a torrent. The round-robin allocation of upload connections used by seeders gives every leecher an equal share of the seeder's bandwidth. The allocation of upload connections inside a torrent by leechers incentivizes cooperation. Note that in principle tit-for-tat leads to an emergent clustering of peers by upload bandwidth: peers tend to exchange data with others with similar bandwidth. However, the randomly allocated connection, called *optimistic unchoke* in BitTorrent terminology, affects the clustering: a low-bandwidth leecher receiving a random connection from a high-bandwidth leecher will frequently reciprocate with a regular connection, potentially disconnecting a leecher with similarly low bandwidth. This bias of slow peers towards faster uploaders is well documented in the literature [5], [6].

Allocation mechanisms for network resources must also interact with lower network layers. In the context of large data transfers, a paramount issue is the interplay between the application and transport layers. BitTorrent clients typically use TCP connections, hence in case a leecher's download connection gets congested, TCP congestion control mechanisms come into effect, interfering with the bandwidth allocation of the BitTorrent client. TCP congestion control divides the bandwidth of a congested download connection equally among all uploading connections.

The interplay of the current BAP solution and TCP congestion-control has non-obvious effects on the overall resource allocation. For instance, consider a scenario in which there are two torrents, each with one leecher. There are also two seeders, s_1 and s_2. The upload and download bandwidth of all peers is c. Seeder s_1 has seeding capacity 1 and is thus only active in one torrent, while s_2 has seeding capacity 2 and is active in both torrents. If the seeders allocate their bandwidth according to the current BAP solution, the leecher served by both seeders will be a bottleneck, and the download capacity of this leecher will be divided equally among the two seeders. The other leecher would be served by a seeder with spare capacity $0.5c$. The resulting aggregate download speed will be $1.5c$, instead of the maximum of $2c$. If the system aims at maximizing throughput, this is a suboptimal solution.

Finally, note that the SSAP and BAP solutions we describe in this section are decentralized. Each peer acts autonomously based on local information about other peers. We create simulators to determine the resource allocation that results from applying these decentralized solutions by all peers. The next two subsections explain the simulators for the current SSAP and BAP solutions, respectively.

B. *A simulator for the current SSAP solution*

We approximate the current SSAP solution with a simulator that calculates the allocation to which the community eventually converges given its instantaneous configuration. The algorithm for this simulation is named *cSSAP* and is presented in Algorithm 1. The simulator repeatedly iterates over the peers with seeding sessions and runs the observed torrent selection heuristic for each of these peers. The information available to the seeders about leecher proportions in the torrents is updated after each seeder decision. The simulation stops when the peers stop changing their allocations.

Algorithm 1 cSSAP – Current SSAP solution simulator

```
$\forall i \in I, \Sigma_i := \emptyset$
repeat
    $consensus$ := true
    for all seeder i do
        order-descending-by-proportion-of-leechers($\Lambda_i$)
        $\Sigma'_i$ := top-$k_i$($\Lambda_i$)
        if $\Sigma'_i \neq \Sigma_i$ then
            $consensus$ := false
        end if
        $\Sigma_i$ := $\Sigma'_i$
    end for
until $consensus$
```

Note that we are interested in the effect of the current SSAP mechanism on the composition of the active torrents sets. We isolate this effect by examining the solution to which the mechanism converges; we do not consider the convergence time. In reality, peers get information about the proportions of leechers in torrents only periodically. Depending on the rate of state changes in the system, this may hamper convergence. Nevertheless, this is essentially an information dissemination concern, which is outside the scope of this work.

C. A simulator for the current BAP solution

Similar to SSAP, our analysis is concerned with the result of the current BAP solution after convergence. To approximate this result, we use a simulator named *cBAP*, that repeatedly performs the following steps:

1) Allocate the available upload bandwidth of each peer according to the current BAP solution without considering the download bandwidths of leechers, excluding congested leechers where the uploader has a share of the download bandwidth equal to the other uploaders;
2) Check every leecher for download congestion: if there is no congestion, subtract the allocated bandwidth of uploaders from their available bandwidth; if there is congestion, subtract only an equal share of the leecher's download bandwidth from the available bandwidth of every uploader.

The simulation stops when for each peer, either there is no more upload bandwidth available, or every leecher the peer is uploading to is congested and the peer has a share of its download bandwidth that is at least equal to the average share of the other uploading peers.

We validate cBAP with an experiment using regular BitTorrent clients that are instrumented to provide detailed reports

TABLE I
PARAMETERS FOR THE VALIDATION EXPERIMENT. L AND S INDICATE IF A PEER IS A LEECHER OR A SEEDER IN A TORRENT, RESPECTIVELY.

Peer ID	Download bandwidth (KiB/s)	Upload bandwidth (KiB/s)	Torrent A	B	C
1	512	64	S		
2	1024	128	L		
3	2048	256	L	L	
4	2048	256	L	S	
5	2048	256		L	L
6	512	64			L
7	512	64	L	L	S
8	2048	256	L	L	L
9	1024	128	L	L	L
10	512	64	L	L	L

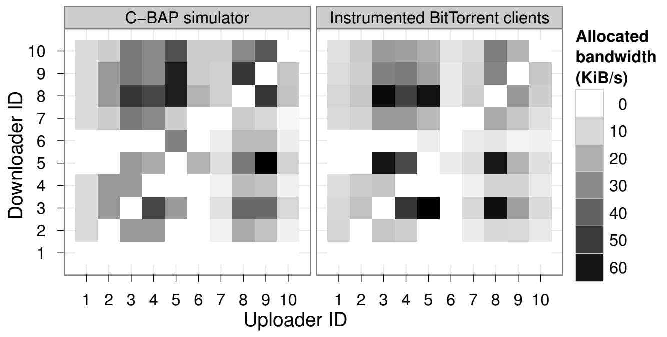


Fig. 1. A comparison of bandwidth allocations produced by the cBAP simulator and an experiment with real BitTorrent clients.

on the data exchanges. The output logs of clients contain sufficient information to determine the bandwidth allocations among the peers during download. The validation consists of simultaneously starting ten peers that participate in three torrents. Some of these peers participate in multiple torrents, creating the need for inter-swarm resource allocation. Moreover, peers have heterogeneous bandwidths, so that clustering can be observed in the experiment. The characteristics of all peers are summarized in Table I. Each peer uses an asymmetric connection with a download bandwidth equal to 8 times the upload bandwidth, and the size of the files distributed in each torrent is 200 MiB.

To compare the cBAP simulator to the actual BitTorrent clients, we discard the warm-up and end phases of the experiment. The warm-up phase is the period before all peers have downloaded at least 10% of the torrent. During this period, it is the piece availability—and not the RAP solutions—that chiefly determines the resource allocation. The end phase of the experiment is the period after which one leecher has become a seeder. When this happens, the configuration of the community has changed, and must be compared against another solution by the simulator.

A comparison between the results of the simulation and the experiment, using the same configuration, is shown in Figure 1. Overall, the bandwidth allocation patterns are similar, with the experiment displaying a marginally less stable clustering. Also the absolute values of the bandwidths allocated by the peers are close, with the highest pairwise transfer speed ~60 KiB/s in both cases.

IV. Optimal bandwidth allocation in BitTorrent communities

In this section, we introduce a graph-theoretical model of resource allocation in BitTorrent communities that allows us to map BAP to flow network problems. This mapping, in turn, permits us to apply graph-theory solutions for BAP targeting throughput maximization in the community, and to devise an algorithm to find the max-min fair allocation of bandwidth to the leeching sessions in the community.

A. A graph-theoretical model for BitTorrent communities

We model the state of a BitTorrent community at a certain instant as a flow network $G = (V, E, f, c)$, where V is the set of vertices, E is the set of edges, f is the flow function, and c is the capacity function. The flow network G is a directed tripartite graph, with V being the union of three disjoint subsets of vertices, U, L and D, and with each edge in E connecting two vertices that are in distinct subsets. These three sets of vertices are defined in the following way:

- the upload nodes $U = \{u_1, \dots, u_m\}$ represent the upload potential of the m users (both seeders and leechers) who are active in the community at the instant considered;
- the leeching sessions nodes $L = \cup_i L_i$ represent the presence of leechers in torrents, where $L_i = \{l_i^t \mid i \in L_t\}$ is the set of leeching sessions of user i; and
- the download nodes $D = \{d_1, \dots, d_n\}$ represent the download potential of the leechers.

In the graph G, a user i is represented by the set of nodes $\{u_i, d_i\} \cup L_i$. Figure 2 shows an example mapping.

The set of edges E represents the transfer potential from the upload nodes to the leeching sessions and from the leeching sessions to the download nodes. If user i is leeching in a torrent t, then there are edges from u_i to the leeching sessions of all other users in torrent t. Using the notation introduced in Section II, this formally means that $\forall t \in T, \forall i, j \in L_t \ (i \neq j) : (u_i, l_j^t) \in E$. Similarly, if a user i is seeding in a torrent t then there are edges from u_i to the leeching sessions of all other users in torrent t. Formally, $\forall t \in T, \forall i \in S_t, \forall j \in L_t : (u_i, l_j^t) \in E$. All of the edges thus defined are called *upload edges*. Finally, to represent downloading, the graph also has edges from the leeching session nodes to download nodes: $\forall t \in T, \forall i \in L_t : (l_i^t, d_i) \in E$. These edges are called *download edges*.

The capacity and flow functions of G are defined as follows:

- the capacity function $c : U \cup L \cup D \to \mathbb{Z}$ represents the bandwidth of peers, where $c(u_i) := \mu_i$ is the upload bandwidth of user i, $c(l_i^t) := \infty$, and $c(d_i) := \delta_i$ is the download bandwidth of user i, and
- the flow function $f : E \to \mathbb{R}$ represents the bandwidth allocation, having the property of flow conservation: $\forall l_j^t \in L, \sum_{u_i \in U} f(u_i, l_j^t) = f(l_j^t, d_j)$.

It is easy to see that any flow in G is equivalent to a bandwidth allocation $\mathcal{B}$, and that the Seeding Sessions Allocation Problem is equivalent to selecting a subset $E' \subseteq E$ such that $\forall i \in I : |\{t \in T \mid (u_i, l_j^t) \in E\}| \leq k_i$.

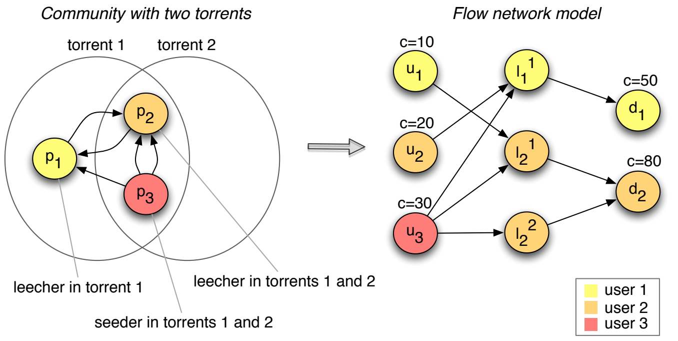


Fig. 2. An example of a BitTorrent community with three users and two torrents and its representation as a flow network.

B. Maximizing throughput

Using the flow network model defined above, solving BAP to maximize throughput is equivalent to solving the maximum flow problem for G, a problem for which several algorithms exist [7]. In this paper, a linear programming (LP) problem formalization is used, which we denote *MaxFlow*:

$$\begin{array}{lll} & \max \sum_{(u_i, l_j^t) \in E} f(u_i, l_j^t), & \\ \text{subject to} & \sum_{t,j} f(u_i, l_j^t) \leq \mu_i & \forall u_i \in U, \\ & \sum_t f(l_j^t, d_j) \leq \delta_j & \forall d_j \in D. \end{array}$$

Note that the flow is not required to take integer values. Because the capacity function does take integer values, the integral flow theorem states that there exists an integral maximum flow; this solution can be found in polynomial time using an LP solver. In our experiments we use MOSEK [8].

C. Max-min fairness algorithm

Our second goal from Section II-C is to find the max-min fair allocation. In order to do so, we establish an iterative algorithm, which we denote *MaxMin*, that maintains an increasing set F of download edges for which the flow values are fixed to their proper value in the max-min fair allocation. This set F is initially empty, and in every iteration, new download edges are added to F. In every iteration k, our algorithm solves the following linear programming problem MM_k:

$$\begin{array}{lll} & \max \phi_k, & \\ \text{subject to} & f(l_i^t, d_i) \geq \phi_k & \forall (l_i^t, d_i) \in E_k, \\ & \sum_{t,j} f(u_i, l_j^t) \leq \mu_i & \forall (u_i, l_j^t) \in E, \\ & \sum_j f(l_j^t, d_j) \leq \delta_j & \forall (l_j^t, d_j) \in E, \end{array}$$

where E_k is the set of edges whose flows have not yet been fixed in previous iterations, i.e., $E_k = E \backslash F$. The MaxMin algorithm stops when F contains all download edges.

In the main loop of our algorithm, having solved the actual instance of MM_k, an LP solver returns with a flow f on the graph. In this flow, there may be multiple download edges with the max-min flow value ϕ_k. These edges are collected into a set Φ, which contains the candidate edges to be added to the set F. However, among the edges in Φ, there may be edges on which the flow value can be increased. In order to check this property, the algorithm continues with two inner loops; to explain these, we use the following terminology. We say that an upload node u_i node is *saturated* if $\sum_{t,j} f(u_i, l_j^t) = \mu_i$, that download node d_i has the *max-min property* if there is no torrent t for which $f(l_i^t, d_i) > \phi_k$, and that a download node d_i is *saturated* if $\sum_t f(l_i^t, d_i) = \delta_i$.

We now describe the two inner loops. The first one checks for all elements of Φ whether they should actually be included in F. We only keep an edge (l_i^t, d_i) for which either a) the download node d_i is saturated and has the max-min property, or b) all upload nodes u_j connected to it (through an upload edge (u_j, l_i^t)) are saturated and all other upload edges $(u_j, l_{i'}^{t'})$ with positive flows are connected to download edges $(l_{i'}^{t'}, d_{i'})$ for which the flow is $\leq \phi_k$ (i.e., u_j cannot be desaturated).

The second loop considers all download edges (l_i^t, d_i) in Φ for which condition b) but not condition a) from the first loop holds. We discard those download edges (l_i^t, d_i) for which there exist upload edges (u_j, l_i^t) with a saturated upload node u_j, which has other $(u_j, l_{i'}^{t'})$ edges with positive flows on them in such a way that $(l_{i'}^{t'}, d_{i'})$ is not in Φ, but has $f(l_{l'}^{t'}, d_{i'}) = \phi_k$.

Finally, the MaxMin algorithm takes the elements of Φ, fixes the flows on them, and adds them to F.

Properties of the MaxMin algorithm. We first observe that in each iteration at least one element is added to F, since at least one edge with the minimum flow ϕ_k is kept in Φ after filtering. On the other hand, the flow on these edges $(l_i^t, d_i) \in \Phi$ can be increased only by decreasing flows on those edges (l_j^t, d_j) which have at most flow value ϕ_k, which assures that all edges in Φ belong to the max-min fair allocation. Since the set F of edges with fixed flows is incrementally constructed using the edges from Φ, it follows that the algorithm finds the max-min fair allocation for a given flow network G.

As a consequence, we find that the number of iterations in our algorithm is at most equal to $|E|$. For each iteration there is a linear program, MM_k, to be solved, which happens in polynomial time (we again use MOSEK). The filtering part of the algorithm has complexity $O(|E| \cdot |U| \cdot |L|)$, as it only considers the download edges and edges connected to them through paths of length at most two.

Regarding the existence of the solution of the max-min fair allocation problem, note that our MaxMin algorithm runs on a continuous, convex set (bounded by the finite number of constant capacities), on which the max-min fair allocation exists [9]; moreover, the allocation is unique [10].

V. Datasets

To evaluate current inter-swarm resource allocation mechanisms in realistic conditions, we derive RAP instances from traces of real-world BitTorrent usage. In the following, we describe the datasets we use and the method for extracting problem instances.

A. Communities studied

We use data from two BitTorrent communities: Bitsoup and Filelist[1]. Both traces were collected by periodically crawling web pages published in these communities containing user activity information. These pages include, for each user in each torrent, the user name, the session duration, and the amount of data uploaded and downloaded in the session. The pages were crawled hourly for Bitsoup, and every six minutes, on average, for Filelist. Table II summarizes the datasets. In total, there are around 100 000 BitTorrent sessions active at every moment, allowing us to form large-scale problem instances.

[1]We note that some of this data has been analyzed before (e.g., [4], [11]). Nevertheless, the aspects evaluated in this paper have never been considered.

TABLE II
Summary of datasets. (95% confidence intervals for means).

	Bitsoup	Filelist
Registered users	84 007	91 745
Total torrents	13 741	3 236
Mean active torrents	6 869.6 ± 30.8	512.2 ± 10.2
Mean active sessions	76 370.3 ± 1 135	32 829.4 ± 672.8
Mean seeders/leechers ratio	5.125 ± 0.155	3.65 ± 0.2

BitTorrent communities that require registration such as those we analyze are known to have lower resource contention than open BitTorrent sites such as The Pirate Bay [2]. This is caused by accounting mechanisms used by the community administrators to keep track of the contributions of users and to expel those users who fail to contribute enough. As a consequence, users seed for longer, and the proportion of leechers is low. To investigate if our results are affected by this, we generate problem instances with higher proportions of leechers than those in the traces. In order to do so, we start with the traces and we reduce the seeding capacities of all users to produce a second dataset with an overall ratio of two leeching sessions per seeding session.

B. Extracting seeding capacities and seeding libraries

Given the set of users who are online at a certain instant, we define for each user a seeding capacity and a seeding library. The seeding capacity of a user at a certain time is taken directly from the traces as the number of torrents the user is seeding at that time.

Defining the contents of the seeding libraries of the users is a more complex task because the traces only contain a series of times when the user was seeding a torrent, but no information about when that torrent was removed. We circumvent the absence of such information by considering the two extreme scenarios for reconstructing the seeding libraries. The first one, named *minimal libraries*, assumes a user deletes a torrent immediately after the last time the user is observed seeding it. In this scenario, a torrent is in the seeding library of a user at a certain time only if that user was observed seeding it both before and after that time. The second scenario, named *maximal libraries*, assumes users never delete torrents. Then, a torrent is in the seeding library of a user if the user was observed seeding it at least once in the past. The seeding library size distribution is heavily skewed in both communities. In Bitsoup, using minimal and maximal estimation, the median library sizes are 3 and 6, respectively, while the maximums are 290 and 542, respectively. In Filelist, the medians are 2 and 4, while the maximums are 72 and 566.

To obtain unbiased comparisons using different times in different traces, we define limited time windows for analyzing past and future events relative to each instant. In addition, we use a random selection of instants from one whole week of each of the two BitTorrent community traces. This allows us to account for most short-term seasonality in BitTorrent usage, which is daily or weekly, and to have a time window for seeding library estimation of 28 days in both traces. In Sections VI and VII, we use 55 problem instances derived from Bitsoup and 45 from Filelist.

C. Upload and download bandwidths

The traces do not contain information about the upload and download bandwidths of users. To obtain realistic numbers, we turn instead to a trace obtained by Isdal et al. [12], who measured the upload bandwidths of a large sample of BitTorrent peers using passive measurement tools. We derive random samples from this dataset to assign to the users in our problem instances, preserving the distribution of bandwidths in [12]. Additionally, we consider two types of user connections. If a user's upload bandwidth is less than 100 Mbit/s, we assume the connection to be asymmetric with download bandwidth equal to eight times the upload bandwidth (in line with connections in Europe and North America). On the other hand, if the upload bandwidth of a user is higher than 100 Mbit/s, the user is assumed to have a symmetric connection with equal download and upload bandwidth.

VI. Can current algorithms provide high throughput?

In this section, we evaluate the performance of current SSAP and BAP solutions, as resulting from the cSSAP and cBAP simulators, respectively, in the context of file-sharing communities interested in maximizing aggregate throughput.

A. Baseline

We first establish a baseline for performance of any allocation mechanism. Recall that, for a SSAP solution, the bandwidth allocation that maximizes aggregate download speed of all users is that given by MaxFlow (as discussed in Section IV). Our baseline is thus the performance of MaxFlow on an unconstrained input, where all seeders can use their whole seeding libraries–a relaxed version of RAP. The results represent an upper bound for solutions of any RAP instance where the seeding capacity constraint holds.

Figure 3 depicts the mean download speed across leeching sessions, considering different leecher proportion levels, and library estimation methods. We observe that changes in leecher proportion have a sizeable effect in Filelist, but little effect in Bitsoup. This implies that a considerably larger fraction of seeding sessions in Bitsoup is not contributing to the maximum flow in the community. Removing these seeding sessions has no effect on community performance. At the same time, the mean download speed in Filelist is 2–3 times the speed in Bitsoup, suggesting the configuration of Filelist is such that leechers and seeders are balanced more evenly in torrents.

It is also notable that library estimation method has no significant influence on the results. Maximal library estimation

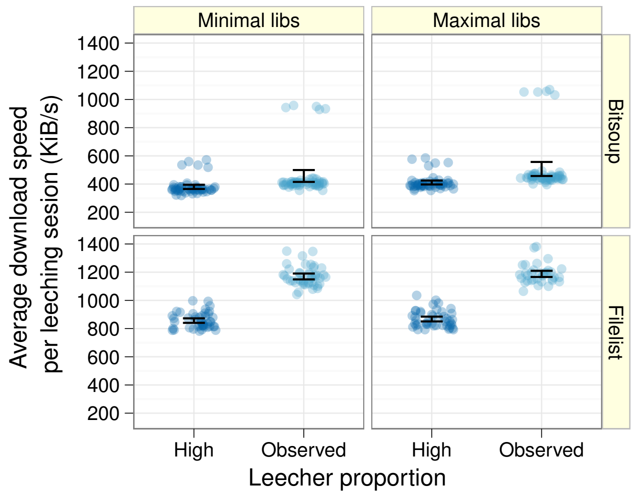


Fig. 3. Upper bound throughput for all scenarios, as produced by MaxFlow in relaxed RAP instances (means with 95% confidence intervals).

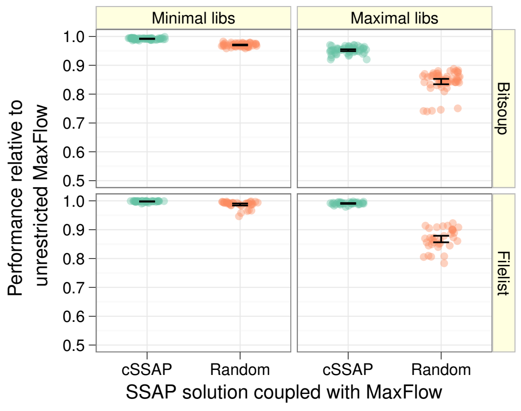
provides considerable more option for peers, but this does not lead to higher performance. Investigating the allocations produced by MaxFlow shows that in all solutions, the majority of seeders does not use any file from their libraries. This suggests that it is possible to attain the maximum flow in the RAP instances we consider even if most peers do not allocate bandwidth to seeding sessions.

B. Torrent selection

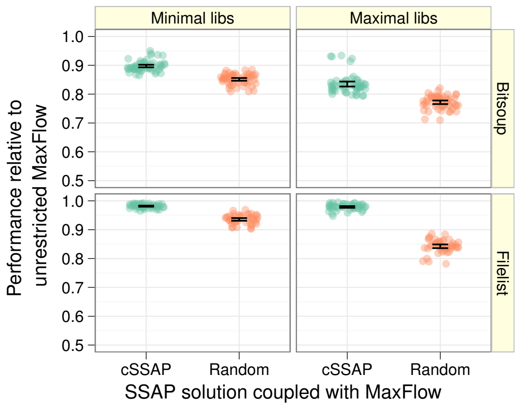
Our next experiment assesses whether the current solution for SSAP limits the performance of solutions based on it for the complete resource allocation problem. Thereafter, we use the notation *Algorithm1*+*Algorithm2* to denote a RAP solution composed by the combination of two algorithms *Algorithm1* and *Algorithm2*, that address SSAP and BAP, respectively.

This experiment compares the performance of cSSAP+MaxFlow running in succession to the established baseline–which is an upper bound. The solution cSSAP+MaxFlow would be equivalent to having clients solve torrent selection in a decentralized manner, and then obtaining from an oracle the optimal bandwidth allocation for their SSAP solution. If cSSAP+MaxFlow performs similarly to the baseline, it is possible to affirm that the current torrent allocation does not limit the performance of a complete solution for RAP. At the same time, It may happen that cSSAP+MaxFlow performs well in the experiment because the space of possible allocations does not allow a different outcome. To test for this, we examine the performance of a random torrent allocation coupled with MaxFlow.

Results comparing the performance upper bound, cSSAP+MaxFlow and Random+MaxFlow are presented in Figure 4. Performance is measured as the aggregate download speed of all peers as relative to the baseline. For Filelist, there are negligible or no differences between solutions from cSSAP+MaxFlow and the upper bound. Moreover, this happens regardless of the library estimation and the proportion of leechers. In Bitsoup, cSSAP+MaxFlow is equivalent to the upper bound in most scenarios, but 10-15% worse than the upper bound for the scenarios with



(a) Observed proportion of leechers



(b) High proportion of leechers

Fig. 4. Throughput produced with (a) observed and (b) high leecher proportions by current and random SSAP solutions coupled with MaxFlow and relative to the performance upper bound for each RAP instance (means with 95% confidence intervals).

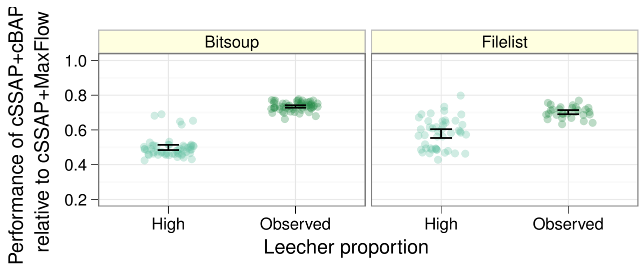


Fig. 5. Throughput of cBAP coupled with cSSAP relative to optimal, cSSAP+MaxFlow (means with 95% confidence intervals, minimal libraries).

high leecher proportion. However, we cannot know if the upper bound performance is attainable in a given scenario, so we cannot conclude cSSAP is affected by the change in the proportion of leechers or if the upper bound is not achievable for the high leecher proportion scenario. Finally, there is a slight difference in the performance obtained with minimal and maximal libraries in Bitsoup. These differences can be attributed to an effect of larger choice on cSSAP, because all solutions possible with minimal library estimation are also possible with maximal library estimation.

Overall, it follows that it is possible to attain optimal or nearly optimal performance using the current torrent selection mechanism. This is notable given that this mechanism ignores bandwidth information. Our results suggest that an efficient bandwidth allocation algorithm can cope with this limitation. Finally, an analysis of the Random+MaxFlow results shows that a heuristic that does not consider any torrent information can only hamper the performance of an efficient BAP solution to a limited extent. Our results thus suggest a reduced space of possibilities for torrent selection.

C. Bandwidth allocation

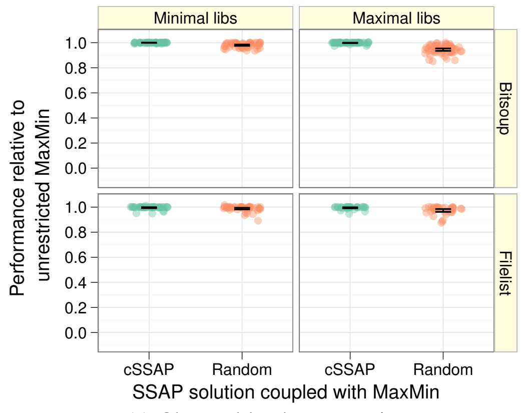
We now examine the efficiency of the current bandwidth allocation mechanism. Our experiment compares cBAP coupled with cSSAP to the optimal solution for the cSSAP allocation, cSSAP+MaxFlow. Figure 5 presents the results of this experiment. Only minimal libraries are considered, as we know from our previous experiment that library estimation has a negligible effect on solutions (Figure 4).

Results are similar for both communities: current BAP solutions achieve less than 80% of the optimal throughput, and the performance of cBAP is affected by resource contention. In the scenarios with high leecher proportion, cSSAP+cBAP achieves only 50-60% of the performance of cSSAP+MaxFlow. Such results highlight that current decentralized mechanisms fall short of the performance that can be achieved in multi-swarm scenarios. Furthermore, note that the decrease in relative performance of cSSAP+MaxFlow suggests that the more resource contention in the community, the further from the optimal current methods are. This is particularly relevant for communities that have less seeding than those we study. Finally, the similarity between relative performance of cSSAP+cBAP in the two communities suggests the performance of these mechanisms is independent from the structure of the community.

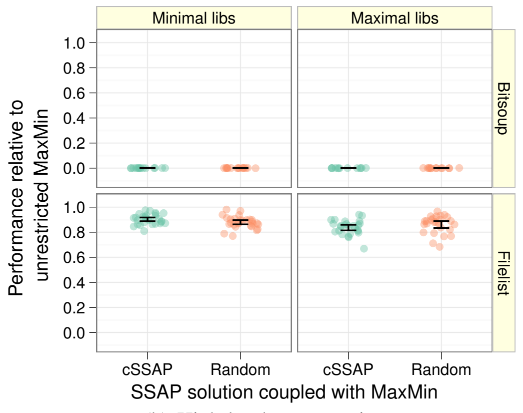
Summary We find that the current torrent selection mechanism does not limit the performance of file-sharing communities, often allowing for optimal solutions if combined with optimal bandwidth allocation. On the other hand, the bandwidth allocation mechanism presently implemented in BitTorrent clients significantly hampers the performance of file-sharing communities, and performs particularly worse in communities with a high leecher proportion. Finally, the upper-bound performance of a file-sharing community is not affected by the size of the seeding libraries and is only slightly affected by the variation in leecher proportion we consider.

VII. Are current algorithms appropriate for video streaming?

We now turn to the video-streaming use-case. In this context, the ideal resource allocation is max-min fair for all leeching sessions. Such an allocation provides the best possible service for the sessions most exposed to streaming interruptions while guaranteeing that the rest of the sessions obtain a service at least as good. We use a performance metric appropriate for video streaming, the download speed of the fifth percentile worst-performing leeching session. If the metric has value v, 95% of the sessions in the community receive a download speed at least equal to v.



(a) Observed leecher proportion



(b) High leecher proportion

Fig. 7. Fifth percentile session download speed with (a) observed and (b) high leecher proportions produced by current and random SSAP solutions coupled with MaxMin and relative to unrestricted MaxMin (means with 95% confidence intervals).

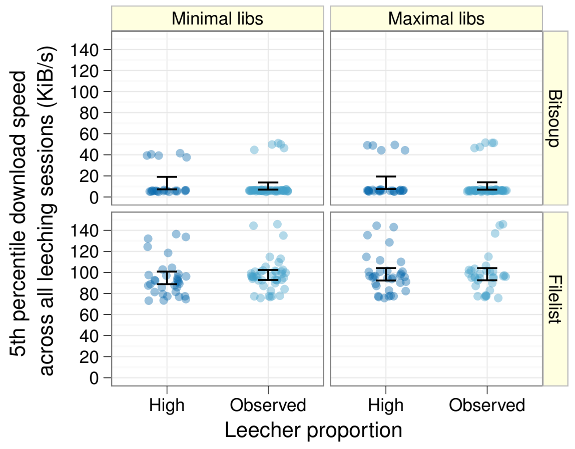


Fig. 6. Upper bound throughput for all scenarios, as produced by MaxMin in relaxed RAP instances (means with 95% confidence intervals).

A. Baseline

The baseline for this experiment is obtained by running the MaxMin algorithm introduced in Section IV on relaxed RAP instances. The average download speed for the fifth percentile leeching session across all problem instances is presented in Figure 6. The performance in Filelist is considerably higher than in Bitsoup, similarly to what we observed for aggregate download speed in Section VI. The fifth percentile download speed in Filelist is ~6 times that in Bitsoup. Within a community, the results are not affected by seeding library estimation nor by the leecher proportion. This implies that the performance of the worst performing leeching sessions cannot be improved just by having seeders participate in more torrents from their libraries.

B. Torrent selection

Analogously to our analysis of aggregate download speed, we first determine whether the current SSAP solution hinders the performance of the optimal BAP solution for video streaming. This is done comparing cSSAP+MaxMin to MaxMin running on the relaxed RAP. At the same time, we establish the extent to which any SSAP solution can affect the overall RAP solution by analyzing the results of a random torrent allocation (Random+MaxMin) in relation to the same unconstrained MaxMin solution. The results are shown in Figure 7.

For the scenarios using the observed leecher proportion, there are only negligible differences between current, random and unrestricted SSAP solutions. This suggests the cSSAP is adequate for maximizing the fifth percentile performance. At the same time, the close-to-optimal result of random selection points to a limited potential for choice; it seems SSAP solutions can only have limited effect on the RAP solution in the problem instances we study. In summary, given the observed proportion of leechers, solving SSAP without bandwidth information does not affect streaming performance when an efficient BAP solution is used–a similar outcome to the throughput maximization use-case.

With regard to the high leecher proportion scenario, Filelist results for cSSAP and random selection show performance drops of 10–20% compared to the baseline, for both seeding library configurations. On the other hand, using Bitsoup data, it is remarkable to see that cSSAP generates a torrent selection where MaxMin cannot provide the lowest five percent of the peers with any bandwidth at all. However, since our baseline applies to an unconstrained RAP, it may be that a higher performance is not attainable by any solution that respects the seeding constraint.

C. Bandwidth allocation

Next, we investigate the performance of cBAP for video streaming. Starting with the torrent allocation produced by cSSAP, we compare the solution of cBAP to the optimal BAP solution generated by our MaxMin algorithm. The results are depicted in Figure 8. Note that only minimal libraries are considered, as we have observed library estimation has no significant effect on the performance that can be achieved by BAP solutions. Moreover, Bitsoup is not considered in the case with is a high proportion of leechers, since cSSAP generates a torrent allocation in which it is impossible to provide any bandwidth to the worst five percent of streaming sessions.

Results are similar for the other scenarios for both communities: fifth percentile download speeds produced by cBAP are around 30-40% of the baseline values. This shows that current

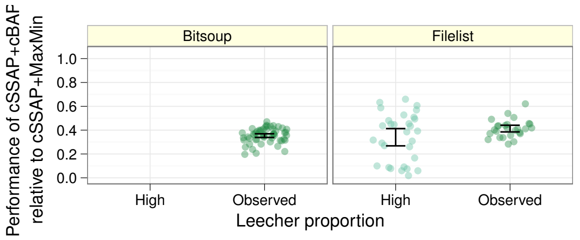


Fig. 8. Fifth percentile leeching session download speed produced by cBAP coupled with cSSAP relative to max-min fair allocation speed cSSAP+MaxMin (means with 95% confidence intervals, minimal libraries).

BAP solutions deployed in BitTorrent are far from ideal in the video-streaming use-case. However, differently from what we observed for aggregate throughput, there is no sizeable effect of leecher proportion on the relative performance of cBAP.

Summary Akin to our results for file sharing, we find that the current torrent selection mechanism does not hamper performance in video-streaming communities. Nevertheless, randomly selecting torrents allows for similar performance using our datasets. Regarding bandwidth allocation, we again find that the current mechanism's performance is substantially worse than the optimal. Finally, the upper bound is not affected by seeding library estimation or leecher proportion.

VIII. Related work

Considerable research and development effort has been invested in designing and evaluating BitTorrent's intra-swarm resource allocation methods. Experimental investigations by Legout et al. suggest that the current algorithms for choosing upload partners inside a swarm need no further improvement [1] and document the high utilization of upload bandwidth inside a swarm [6]. BitTorrent has also been studied at the community level. Zhang et al. [2] show how an entire ecosystem forms around the P2P protocol. Guo et al. [3] and Andrade et al. [4] analyze traces of multiple BitTorrent communities. Nevertheless, previous work investigating multi-swarm systems has not considered the community-level metrics we use, nor evaluated the effect of current inter-swarm resource allocation mechanisms.

More similar to our work, Dunn et al. [13] explore seeding strategies for a BitTorrent-like system centered around a content provider. Their goal is minimizing the bandwidth demand at the provider–equivalent to maximizing P2P throughput. Using synthetic scenarios, they find that the behavior of current BitTorrent algorithms can be improved. Our results do not contradict this finding, but question whether improvements for SSAP solutions are relevant for most real communities. Peterson et al. [14] design a BitTorrent-inspired content distribution system with a central bandwidth allocation algorithm. Similar to us, they envisage different goals for the system, such as guaranteeing a minimum service level in swarms or avoiding starvation. However, they only present results for the throughput maximization goal, for which they also find BitTorrent to perform suboptimally. We corroborate this finding, adding that it happens in real BitTorrent communities. Furthermore, we expand the results of Peterson et al. by examining the video-streaming use-case.

IX. Conclusion

In this paper, we present a performance evaluation of *de facto* mechanisms for inter-swarm resource allocation in BitTorrent communities. This paves the way for informed developments of these mechanisms by identifying requirements and relevant factors that affect the performance of BitTorrent communities from a multi-swarm perspective.

We conclude that, for both file-sharing and video-streaming communities, the present torrent selection mechanism is suitable if coupled with efficient bandwidth allocation algorithms, but the present bandwidth allocation mechanism performs significantly worse than optimal in multi-swarm operation, especially in the case of high leecher proportions. In a way, our results highlight there is currently a price for anarchy in BitTorrent communities: with individuals allocating resources solely in their own interest, they do not fulfill the global objective optimally. Nevertheless, this does not imply that globally optimal mechanisms should not be incentive-compatible. Instead, future work should ideally improve these mechanisms considering multi-swarm incentives.

The observation that maximal libraries do not improve the upper bound performance is relevant for the design of future BitTorrent clients. Our simulations suggest there is little to gain through peer-level caching of user downloaded torrents. Future work should extend the generalizability of our data with a similar analysis of more BitTorrent communities. Furthermore, the development of real-time implementations of our optimal bandwidth allocation algorithms could lead to efficient inter-swarm resource allocation in cooperative scenarios where peers follow the directions of a centralized coordinator.

References

[1] A. Legout, G. Urvoy-Keller, and P. Michiardi, "Rarest first and choke algorithms are enough," in *ACM IMC*, 2006.

[2] C. Zhang, P. Dhungel, and K. Di Wu, "Unraveling the BitTorrent ecosystem," *IEEE TPDS*, 2010.

[3] L. Guo, S. Chen, Z. Xiao, E. Tan, X. Ding, and X. Zhang, "Measurements, analysis, and modeling of BitTorrent-like systems," in *IMC*, 2005.

[4] N. Andrade, E. Santos-Neto, F. Brasileiro, and M. Ripeanu, "Resource demand and supply in BitTorrent content-sharing communities," *Computer Networks*, vol. 53, no. 4, pp. 515–527, 2009.

[5] M. Meulpolder, J. Pouwelse, D. Epema, and H. Sips, "Modeling and analysis of bandwidth-inhomogeneous swarms in bittorrent," in *Proc. of IEEE P2P 2009*, S.N., Ed. Los Alamitos, USA: IEEE Computer Society, September 2009, pp. 232–241.

[6] A. Legout, N. Liogkas, E. Kohler, and L. Zhang, "Clustering and sharing incentives in BitTorrent systems," in *ACM SIGMETRICS*, 2007.

[7] R. Cottle, E. Johnson, and R. Wets, "George B. Dantzig (1914–2005)," *Notices of the AMS*, vol. 54, pp. 344–362, 2007.

[8] MosekApS, "MOSEK optimization software," www.mosek.com.

[9] B. Radunović and J.-Y. L. Boudec, "A unified framework for max-min and min-max fairness with applications," *IEEE/ACM Trans. Netw.*, vol. 15, pp. 1073–1083, 2007.

[10] D. Bertsekas and R. Gallager, *Data networks*. Prentice-Hall, Inc., 1992.

[11] B. Zhang, A. Iosup, J. Pouwelse, D. Epema, and H. Sips, "Sampling bias in bittorrent measurements," in *Euro-Par'10*, 2010, pp. 484–496.

[12] T. Isdal, M. Piatek, A. Krishnamurthy, and T. Anderson, "Leveraging BitTorrent for end host measurements," in *PAM*, 2007.

[13] R. J. Dunn, S. D. Gribble, and H. M. Levy, "The importance of history in a media delivery system," in *IPTPS*, 2007.

[14] R. S. Peterson and E. G. Sirer, "Antfarm: Efficient content distribution with managed swarms," in *NSDI*, 2009.