Delft University of Technology
Parallel and Distributed Systems Report Series

Inter-swarm resource allocation in BitTorrent communities

Mihai Capotă, Nazareno Andrade, Tamás Vinkó,
Flávio Santos, and Johan Pouwelse
mihai@mihaic.ro

Completed December 2010.

Report number PDS-2010-010

ISSN 1387-2109

Published and produced by:
Parallel and Distributed Systems Section
Faculty of Information Technology and Systems Department of Technical Mathematics and Informatics
Delft University of Technology
Zuidplantsoen 4
2628 BZ Delft
The Netherlands

Information about Parallel and Distributed Systems Report Series:
`reports@pds.twi.tudelft.nl`

Information about Parallel and Distributed Systems Section:
`http://www.pds.twi.tudelft.nl/`

Abstract

Resource allocation in BitTorrent has been extensively studied at swarm-level. However, many BitTorrent users participate in multiple swarms simultaneously, making inter-swarm resource allocation necessary. Mechanisms for inter-swarm allocation have received less attention from the research community, and it is unclear if currently implemented mechanisms best serve users' needs. In this paper, we evaluate the prevalent inter-swarm resource allocation mechanisms using data from two BitTorrent communities and trace-driven simulation. We consider two use-cases: (1) current file-sharing communities, whose objective is to maximize throughput; (2) video-streaming communities, whose goal is maximizing the number of users receiving sufficient resources for uninterrupted streaming. We compare the results of the analyzed mechanisms with efficiency bounds. Such bounds are computed by mapping the resource allocation problem to a graph-theoretical flow problem and using centralized algorithms. In this formalism, throughput maximization is equivalent to regular flow maximization, a problem with well-known solutions. The goal of the video-streaming use-case translates to finding a max-min fair allocation for BitTorrent downloading sessions, a problem for which we devise a new algorithm.

Contents

List of Figures

List of Tables

1 Introduction

BitTorrent's popularity can be attributed, to a great extent, to its efficient resource allocation when distributing a file to multiple machines. A large body of research shows that BitTorrent algorithms are able to create scalable peer-to-peer swarms that distribute content with high efficiency while providing incentives for downloaders to contribute upload bandwidth.

Notwithstanding this understanding of the efficiency of BitTorrent's intra-swarm resource allocation, inter-swarm resource allocation has received limited attention. At the same time, measurements show that most BitTorrent users participate in multiple swarms, or *torrents*, simultaneously [6]. It is customary for users to concurrently download multiple files and continue uploading files they finished downloading. Because of this usage pattern, BitTorrent clients have devised various mechanisms for inter-swarm resource allocation. These mechanisms determine which torrents are kept active at a given moment and how the bandwidth is distributed across these torrents.

In this paper, we examine the performance of currently deployed inter-swarm resource allocation mechanisms using traces of real-world BitTorrent usage. These traces come from two BitTorrent communities – sites which require user registration for access to a collection of torrents [1, 14].

To evaluate how present resource allocation methods perform in the context of these communities, we first describe the algorithms implemented in prevalent BitTorrent clients and introduce methods to simulate them. Next, we devise solutions that provide performance upper bounds for resource allocations. Finally, we compare the performance of current methods with the these upper bounds in problem instances derived from the traces of BitTorrent communities.

In our evaluation, we consider two possible goals for BitTorrent communities: maximizing throughput and achieving a fair bandwidth allocation. The former maps to currently popular file-sharing communities, and the latter to emerging BitTorrent-based video-streaming systems.

Our contributions in the remainder of this paper are as follows:

- a formulation and characterization of the inter-swarm resource allocation problem in its general form (Section 2);
- the characterization of currently prevalent inter-swarm resource allocation mechanisms (Section 3);
- a formulation of the bandwidth allocation problem in BitTorrent communities that maps it to a graph-theoretical flow problem, allowing for well-known approaches to be used to calculate optimal efficiency bounds for this part of the general resource allocation problem (Section 4);
- an algorithm to find the max-min fair allocation of bandwidth to download sessions in a BitTorrent community (Section 4);
- a method to derive resource allocation problem instances occurring in BitTorrent communities based on traces of system usage (Section 5);
- an evaluation of how currently deployed inter-swarm resource allocation algorithms perform regarding the goal of maximizing throughput in a BitTorrent community (Section 6); and

- an evaluation of how presently predominant resource allocation algorithms perform when catering for the video-streaming use case (Section 7).

2 The inter-swarm resource allocation problem

We start by formalizing the problem of allocation resources in a multi-swarm system. This section describes what resource allocation is necessary in a BitTorrent community, formulates the general resource allocation problem, and presents a definition of the two goals for resource allocation that we consider in this paper.

2.1 Background

Let I be the set of users and T the set of torrents in a BitTorrent community. Each user $i \in I$ has upload and download bandwidth capacities μ_i and δ_i, respectively. Each user may participate in multiple torrents simultaneously. In this case, the user i has a *session* in each of these torrents, and these torrents comprise i's set of *active torrents* Γ_i. We say a session is a *leeching session* if i does not own all the content in the torrent and is hence a *leecher* in the torrent. If i is owns all the content and is therefore a seeder in the torrent, the session is a *seeding session*. Each user must divide the upload bandwidth μ_i among the leechers in torrents $t \in \Gamma_i$.

Most BitTorrent clients have a configuration parameter to set the maximum number of active torrents. This happens because efficiently participating in a torrent typically requires a certain number of TCP connections; too many simultaneous connections may reduce the overall upload or download performance of the client. Limiting the number of active torrents limits the total number of TCP connections opened by the client.

A BitTorrent client decides which torrents are active at any given moment. Most clients always keep the leeching sessions active. If i has fewer than $|\Gamma_i|$ leeching sessions, the client will try to fill Γ_i with torrents where i can be a seeder. The set of torrents where i is a seeder at a moment is i's *active seeding set* S_i. The size of the active seeding set of user i at an instant, denoted by k_i, is called the *seeding capacity* of user i.

The torrents in which i can be a seeder at time τ have been fully downloaded by i before τ and kept in i's machine until τ. These torrents form i's *seeding library* Λ_i at τ. The active seeding set S_i is a subset of Λ_i. BitTorrent clients automatically pause torrents in the seeding library that are not in the active seeding set.

2.2 Problem formulation

Let t_S be the set of seeders and t_L the set of leechers in torrent t. The set of *possible user connections* P_Λ is the set of triplets representing potential P2P connections in torrents: $P_\Lambda := \{(i, j, t) \in I \times I \times T \mid (i \in t_S \vee i \in t_L) \wedge j \in t_L\}$.

A **resource allocation** represents the decisions of all users in the community about which leeching sessions they serve and how much bandwidth they offer to each of these leeching sessions. Resource allocation must satisfy two constraints. Constraint C1 is the bandwidth constraint: users cannot offer more than their upload bandwidth allows and they cannot receive more than their download bandwidth allows. Constraint C2 is the seeding capacity constraint: users cannot offer to upload from more seeding sessions than their seeding capacity allows. More formally, a resource allocation is a function $\mathcal{A}_\Lambda : P_\Lambda \to \mathbb{R}$ such that:

(C1) $\forall i \in I \sum_{j \in I, t \in T} \mathcal{A}_\Lambda(i,j,t) \leq \mu_i \wedge \forall j \in I \sum_{i \in I, t \in T} \mathcal{A}_\Lambda(i,j,t) \leq \delta_j$, and

(C2) $\forall i \in I, |\{\mathcal{A}_\Lambda(i,j,t) > 0 | i \in t_S\}| \leq k_i$.

The **Resource Allocation Problem** (RAP) is finding a resource allocation that achieves a specified goal. Such a goal can be either the maximization 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 a second community may be interested in maximizing the median download speed across all leeching sessions. Regarding RAP to satisfy a set of constraints, there can be communities interested in guaranteeing a certain minimum download speed for all users, or communities aiming at maxmin-fair allocations.

Note that 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: bandwidth allocation and seeding sessions allocation. This relaxation leads to an approximative solution for RAP and maps parts of the problem to more tractable equivalents.

We first define a *bandwidth allocation* as a function $\mathcal{A} : P \to \mathbb{R}$ such that the bandwidth constraint C1 holds, where P is a given subset of P_Λ that satisfies C2. Because of P's property, the bandwidth allocation also satisfies C2.

The **Bandwidth Allocation Problem** (BAP) is then finding a bandwidth allocation that achieves an RAP goal. BAP is a tractable relaxation of RAP. It is possible to map BAP to equivalent problems with well-known solutions, as we show in Section 4.

Nevertheless, a solution for RAP presumes a set P. The **Seeding Sessions Allocation Problem** (SSAP) is choosing a subset P of P_Λ, such that $\forall i \in I, |\{(i,j,t) \in P | i \in t_S\}| \leq k_i$. Solving this problem yields a set of possible user connections P that satisfies the seeding capacity constraint C2. Note that in this formulation, SSAP is not necessarily a standalone problem, but a preceding step to BAP.

Framing RAP as composed by BAP and SSAP also allows us to derive upper bounds for its solution. Applying an algorithm that optimally solves BAP to P_Λ will produce an upper bound to RAP. This is equivalent to relaxing RAP by ignoring C2. 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.

2.3 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* (MT), is in line with many existing file-sharing communities. MT is formally defined as finding an allocation $\mathcal{A}_\Lambda$ that maximizes $\sum_{(i,j,t) \in P_\Lambda} \mathcal{A}_\Lambda(i,j,t)$.

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 minimum streaming rate r to as many sessions as possible. However, we opt for a stronger formulation named *Max-min fairness* (MMF): the allocation should provide the highest possible r for the lowest-capacity user, the highest possible r' for the second lowest-capacity 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, MMF enables the community to work with multiple streaming rates of varying qualities and to minimize the number of users experiencing low-quality streams. We define MMF as an adaptation of maxmin fair flow allocations in computer networks. In a max-min fair allocation, the download speed of a leeching session can only be increased by decreasing the download speed of another leeching session that has a lower speed. Formally, an allocation $\mathcal{A}_\Lambda$ is max-min fair iff

$$\forall \mathcal{A}'_\Lambda,\ if\ \exists p \in P_\Lambda\ s.t.\ \mathcal{A}'_\Lambda(p) > \mathcal{A}_\Lambda(p)\ then\ \exists q \in P_\Lambda\ s.t.\ \mathcal{A}_\Lambda(q) \leq \mathcal{A}_\Lambda(p)\ and\ \mathcal{A}'_\Lambda(q) < \mathcal{A}_\Lambda(q). \quad (1)$$

3 Simulating de facto solutions in BitTorrent communities

3.1 Current solutions in BitTorrent clients

Current BitTorrent communities tackle RAP in a decentralized manner using various heuristics implemented by BitTorrent clients. To characterize the solutions resulting from such heuristics, we investigate the implementation of prevalent BitTorrent clients. We consider for this analysis two most popular clients, *uTorrent* and *Azureus*, which account for 80% of BitTorrent usage [14]. Examining the configuration and documentation of these clients shows they solve SSAP and BAP sequentially using a similar behavior.

With regard to SSAP, 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 proportion of leechers. Then, torrents with the highest proportion of leechers are chosen to form the active seeding 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. It is unclear to what extent this omission impacts the quality of the solution.

The solution to BAP involves three steps. First, the client allocates the same number of uploading connections to each active torrent, five by default. Second, the connections are allocated to leechers inside each torrent. This happens differently for seeders and leechers: while seeders allocate connections in a round-robin fashion to all leecher sessions, leechers allocate most connections to the fastest reciprocating peers and a few connections randomly. In the third step, each upload connection receives a fair share of the peer's total upload bandwidth.

In case a leecher's download connection gets congested, TCP congestion control mechanisms come into effect, interfering with BitTorrent clients' bandwidth allocation. TCP congestion control divides the leecher's congested download bandwidth equally among all the uploading TCP streams. When this happens, each uploader must repeat step three of the allocation.

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 allocation of upload connections inside a torrent for leechers implements a variant of tit-for-tat to encourage cooperation. The round-robin upload connection allocation used by seeders gives every leecher a fair share of the seeder's bandwidth.

The interplay of the current BAP solution and TCP congestion-control has non-obvious effects on the overall resource allocation. For instance, consider the following scenario. Let two torrents have one leecher each. Consider also two seeders, s_1 and s_2. Seeder s_1 is only active in one torrent, while s_2 is active in both. Upload and download capacity of all peers is c. If the seeders allocate their bandwidth according to the current BAP solution, the leechers served by both seeders will be a bottleneck, and the resulting throughput will be $1.5c$, instead of the maximum $2c$. This is a problem if the goal is maximizing throughput.

The SSAP and BAP solutions we describe in this section are decentralized. Each peer acts autonomously based on local information about other peers. We turn to simulation to determine the overall community resource allocation that results from applying these decentralized solutions. The next two subsections detail this simulation.

3.2 Approximating current SSAP solutions

We are primarily interested in the effect of the current SSAP heuristic on allocations. To isolate this effect, we examine the solution to which the heuristic converges, ignoring the convergence time. In reality, peers only get information about the state of torrents periodically. Depending on the rate of state-changes in the system, this may hamper convergence. Nevertheless, this would be an effect of information dissemination, which is outside the scope of this work.

To approximate the current solutions, our simulation repeatedly iterates over users with seeding sessions and runs the observed torrent choice heuristic for each of these users. The information available to the seeders about leecher proportion in torrents is updated after each seeder decision. The simulation stops when users stop changing their allocations. Algorithm 1 presents the pseudocode for the simulation.

```
$\forall i \in I, S_i = \emptyset$
**while** *not consensus* **do**
    consensus := true
    **foreach** *seeder* **do**
        order-descending-by-proportion-of-leechers($\Lambda_i$)
        $S'_i$ := choose-top-$k_i$($\Lambda_i$)
        **if** $S'_i \neq S_i$ **then**
            consensus := false
        **end**
        $S_i := S'_i$
    **end**
**end**
```

Algorithm 1: Simulation of proportion-of-leechers seeding sessions allocation

3.3 Approximating current BAP solutions

Similar to SSAP, we are interested in the result of current BAP solutions after convergence. We repeatedly perform the following steps:

1. Allocate each user's available upload bandwidth according to current BAP solution without considering download capacities for leechers, but excluding congested leechers where the uploader already has a fair share of the download bandwidth;

2. Check leechers for download congestion: if there is no congestion, remove the uploaders' allocated bandwidth from their available bandwidth; if there is congestion, remove only a fair share of the leecher's download bandwidth out of the available bandwidth of every uploader.

The simulation stops when, for each uploading user, either there is no more available bandwidth, or all the leechers the user is uploading to are congested and the user has a fair share of their download bandwidth.

We validate this approach, we using an experiment with regular BitTorrent clients. This experiment relies on instrumented clients that report piece exchanges. The output log of a set of such clients contains the necessary information to estimate bandwidth allocation between peers during the experiment.

The experiment consists of starting ten peers simultaneously and form three torrents. Some of the peers participate in multiple torrents, creating the need for inter-swarm resource allocation. Moreover, peers have heterogeneous capacities, so that clustering is observable in the experiment. All peers' characteristics are summarized in Table 1. Each peer uses an asymmetric connection with download bandwidth 8 times larger than his upload capacity, and the content distributed in each torrent is a 200 MiB file.

To compare our simulator against the validation experiment, we discard the warm-up and end of the experiment. The warm-up is the period before all peers have downloaded at least one piece and are able to upload data. During this period, the availability of pieces chiefly determines resource allocation, which is an effect out of our scope. The end of the experiment is the period after which one leecher has become a seeder. When this happens, the configuration of the system has changed, and we cannot compare it with the same results from the simulator.

A comparison between the result of the experiment and the output from the simulator for the experiment scenario is shown in Figure 1. Overall, resource allocation is similar in both results, with the experiment displaying somewhat less stable allocations.

4 Optimal allocations in BitTorrent communities

In the following we introduce a graph-theoretical model of resource allocation in BitTorrent communities that allows us to map BAP to network flow problems. This mapping, in turn, permits us to (i) apply well-known solutions for BAP targeting throughput maximization in the community, and (ii) devise an algorithm to find the max-min fair allocation of bandwidth in the community.

Table 1: Parameters for the validation experiment. An *L* or an *S* in a cell of the table indicate a peer is participating as a leecher or a seeder in a torrent, respectively.

		Peer									
Id		1	2	3	4	5	6	7	8	9	10
Bandwidth (KiB/s)		512	1024	2048	2048	2048	512	512	2048	1024	512
Torrent	A	S	L	L	L			L	L	L	L
	B			L	S	L		L	L	L	L
	C					L	L	S	L	L	L

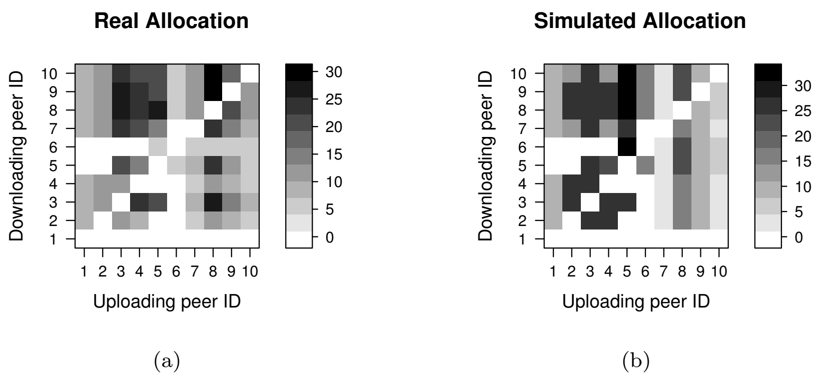

Figure 1: Bandwidth allocation in an experiment with client software compared to our simulator for current BAP solutions. Darker regions represent higher bandwidths, and bandwidth is expressed in KiB/s.

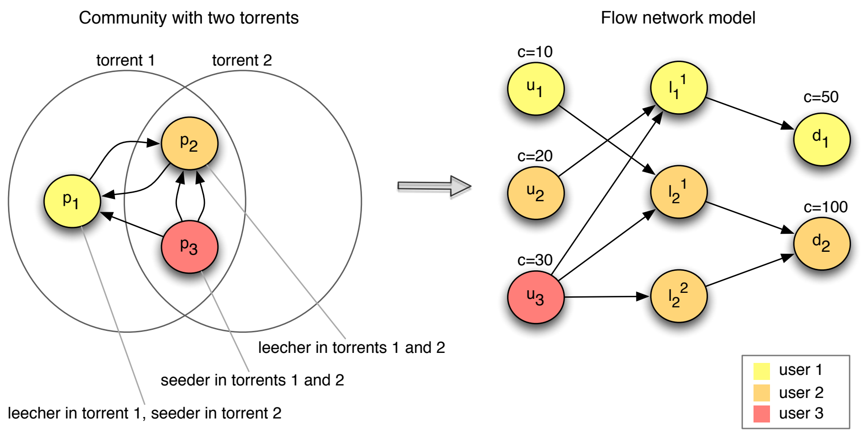

Figure 2: Example community with three users and two torrents and its representation as a flow network.

4.1 A graph-theoretical model for inter-swarm resource allocation

Using graph theory, we model a BitTorrent community as a flow network. Each user $i \in I$ is represented by a set of nodes $\{u_i, d_i, L_i\}$, where u_i and d_i are the upload and download potential of user i, respectively, and $L_i = \{l_i^t \mid i \in t_L\}$ is the set of nodes representing the user's leeching sessions.

Upload potential in the community is represented in the flow network by the set of edges E. If two users are both leeching in a torrent, then there is an edge between them. Formally, $\forall t \in T, \forall i, j \in t_L : (u_i, l_j^t) \in E$. Similarly, a user is connected to all the leechers in all torrents in the user's seeding library: $\forall t \in T, \forall i \in t_S, j \in t_L : (u_i, l_j^t) \in E$. The network also has edges between all l nodes and the d node of a user i: $\forall t \in T, i \in t_L : (l_i^t, d_i) \in E$. Figure 2 depicts an example mapping from a community state to a graph.

We now define the flow network $G = (U, D, L, E, f, c)$, where:

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

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

4.2 Maximizing throughput

Using the flow network model, solving BAP to maximize throughput is equivalent to solving the maximum flow problem for G. Several well-known algorithms exist for solving maxflow [4]. In this paper we use the linear programming problem formalization, which is

$$\begin{aligned} &\max \textstyle\sum_{(u_i,l_j^t)\in E} f(u_i,l_j^t),\\ \text{s.t.}\quad &f(u_i,l_j^t) \le \mu_i \text{ and } f(l_j^t,d_j) \le \delta_j \qquad (\forall (u_i,l_j^t)\in E \wedge \forall (l_j^t,d_j)\in E). \end{aligned}$$

Note that we do not require the flow to be integer. Doing that would imply an integer linear programming problem, which is, in general, NP-hard. Due to the fact that the capacity function is integer, the so-called integral flow theorem states that there exists an integral flow. This can be found in polynomial time using a linear solver. We use MOSEK [10], which is able to solve our maxflow instances efficiently with special options for solving network problems.

4.3 Max-min fairness algorithm

Our second goal from Section 2.3 is to find the Max-min fair allocation. In order to do so, we establish Algorithm 2, which, in every iteration step k, solves the following linear programming problem MM_k:

$$\begin{aligned} &\max f_{\min}^k,\\ \text{s.t.}\quad &f(u_i,l_j^t) \le \mu_i \text{ and } f(l_j^t,d_j) \le \delta_j \qquad (\forall (u_i,l_j^t)\in E \wedge \forall (l_j^t,d_j)\in E),\\ &f(l_i^t,d_i) \ge f_{\min}^k \qquad (\forall (l_i^t,d_i)\in E \;\text{ and } f(l_i^t,d_i) \text{ is not fixed}). \end{aligned}$$

Note that fixing flows on edges means that those fixed flow values are considered as constants and, as such, are not subject to change by the algorithm. As in the maxflow case, we used MOSEK to solve MM_k (with the network optimization options). It was again not required to find integer solutions for MM_k, due to the inefficiency of this approach and also to the fact that in that case the max-min fair allocation is not unique.

The algorithm runs on a finite, continuous, convex set (bounded by the finite many constant capacities), on which the maxmin fair allocation exists [12]; moreover, we know that it is unique [2].

Observe that in the first for-cycle we put those edges from MinSet into FixedSet for which: either (a) their d node is saturated and there exists no (l',d) edge for which $f(l',d) > f_{\min}^k$ or (b) all the u nodes, for which (u,l) exist are saturated in a way that for all other existing (u,l') edge $f(u,l') > 0$ and the is no (l',d') edge for which $f(l',d') > f_{\min}^k$. Now, in the second (filtering) cycle we consider edges from FixedSet for which *only* the second condition holds from the previous step (i.e., the d node is not saturated or the d node has other incoming edges on which the flow value is greater than the current minimum flow, $f_{\min}^k$). We take out those (l,d) edges for which there exist (u,l) edges, with a saturated u node, which has other (u,l') edges with strictly positive flow on them in such a way that (l',d') is not in FixedSet but in MinSet. On such an (l,d) edge the flow value can be increased.

This means, first of all, that we always have at least one element in FixedSet$_k$, otherwise it contradicts the optimization algorithm (i.e., there was possibility for improving $f_{\min}^k$). On the other hand, the flow on these $(l,d) \in$ FixedSet$_k$ edges can be increased only with decreasing flows on those (l',d') edges which have at most

```
k:=1;  FixedSet := $\emptyset$;
repeat
    f := solve($MM_k$);  $f^k_{\min} := \min_{(l,d)} f(l,d)$;
    FixedSet$_k$:=$\emptyset$; MinSet:=$\{(l^t_i, d_i) \in E \mid f(l^t_i, d_i) = f^k_{\min})\}$;
    for all $(l^t_i, d_i) \in MinSet$ do
        if  $((\sum_{l^t_i} f(l^t_i, d_i) = \delta_i) \wedge (\forall l^{t'}_{i'} : f(l^{t'}_{i'}, d_i) \leq f_{\min}))$
        $\vee$ $((\forall (u_j, l^t_i) \in E : \sum_{i',t'} f(u_j, l^{t'}_{i'}) = \mu_j) \wedge$
        $(\forall (u_j, l^t_i) \in E, \forall (u_j, l^{t'}_{i'}) \in E$ s.t. $f(u_j, l^{t'}_{i'}) > 0 : (f(l^{t'}_{i'}, d_{i'}) \leq f_{\min}))$ then
            add $(l^t_i, d_i)$ to FixedSet;
        end
    end
    repeat
        removed:=FALSE;
        for all $(l^t_i, d_i) \in FixedSet_k$ do
            if $(((\sum_{l^t_i} f(l^t_i, d_i) < \delta_i) \vee (\exists l^{t'}_{i'} : f(l^{t'}_{i'}, d_i) > f_{\min}))$ then
                if $(\exists (u_j, l^t_i) \in E \wedge \exists (u_j, l^{t'}_{i'}) \in E$  such that $f(u_j, l^{t'}_{i'}) > 0 \wedge$
                $(l^{t'}_{i'}, d_{i'}) \notin FixedSet_k \ \wedge (l^{t'}_{i'}, d_{i'}) \in MinSet)))$ then
                    remove $(l^t_i, d_i)$ from FixedSet$_k$;
                    removed:=TRUE;
                end
            end
        end
    until not removed ;
    for all $(l^t_i, d_i) \in FixedSet_k$ do
        fix the flow $f(l^t_i, d_i)$ to be $f_{\min}$;
    end
    FixedSet := FixedSet $\cup$ FixedSet$_k$;
    k := k+1;
until $(FixedSet = |\{(l^t_i, d_i) \in E\}|)$ ;
```

Algorithm 2: Max-min Fairness algorithm

flow value $f^k_{\min}$, which assures that all edges in FixedSet belong to the max-min fair allocation. Thus, we conclude that the algorithm finds the max-min fair allocation for a given flow network.

As a consequence, we also know that our algorithm makes at most $|\{(l,d) \in E\}|$ number of steps. For each step there is a linear program, MM_k, to be solved, which happens in polynomial time. The filtering part of the algorithm also has linear complexity as it is considering only the (l,d) edges and all associated paths which go through on them.

5 Datasets

To evaluate current resource allocation methods in realistic conditions, we derive RAP instances from traces of BitTorrent usage. Each problem instance is based on the state of the community at an instant in time. In the following, we describe the datasets we use and the method for extracting problem instances.

Table 2: Characteristics of the datasets with 95% CI for averages.

Trace	Torrents		Users	Sessions
	total	avg. w/ users	total	avg. active
Filelist	3 236	512.2 ±10.2	91 745	32 829.4 ±672.8
Bitsoup	13 741	6 869.6 ±30.8	84 007	76 370.3 ±1 135.5

5.1 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 users' activity information. Such pages include, for each user in each torrent: a user name, session duration, and the amount of data uploaded and downloaded in the session. For Bitsoup, all statistics pages were crawled hourly; for Filelist, crawling happened every six minutes on average. Table 2 summarizes the datasets.

5.2 Extracting torrent libraries and seeding capacities

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

Defining the contents of users' libraries from the traces is a more complex task. The traces do not contain information about all files stored in a user's machine. Instead, they contain only a series of times at which a user was observed seeding a file. This indicates moments in time when the user had the file, but does not inform us if and when it was deleted.

We circumvent the absence of such information by considering the two alternatives for users' libraries that delimit the extreme possibilities from the perspective of the problems we study. The first scenario, named *minimal libraries*, assumes a user deletes the file from the seeding library immediately after the last time the user is observed seeding this file. In this scenario, a file is in the seeding library of a user at a time if that user was observed seeding this file both before, and after that time. In the second scenario, named *maximal libraries*, the libraries have the maximum possible size. In this scenario, users never delete files. A file is in the seeding library of a user if that user is observed seeding it at least once in the past. To obtain unbiased comparisons of problem instances derived from different times in different traces, we define limited time windows for analyze past and future events relative to each instant.

5.3 Upload and download capacities

The traces do not have information about the upload and download capacities of users. We turn instead to an additional trace to derive realistic bandwidth capacities for users in each problem instance. Isdal et al. [7] measured the upload capacity of a large sample of BitTorrent users using passive measurement tools. We assume

[1]We note that some of this data has been analyzed before (for example Andrade et al. [1] and Zhang et al. [13]). Nevertheless, the aspects evaluated in this paper have never been considered using these datasets.

this dataset represents the population of BitTorrent users and derive random samples from it to assign to the users in our problem instances, preserving the distribution of bandwidth capacities observed by Isdal et al.

Finally, we consider two types of connections available for users. If a user's upload capacity is less than 100 Mbit/s, we consider it an asymmetric connection whose download capacity is eight times higher than the upload capacity (this level of asymmetry is in line with connections available in most European and North American countries). On the other hand, if the upload capacity of a user is equal to or higher than 100 Mbit/s, the user is assumed to have a symmetric connection whose download capacity is equal to the upload capacity.

5.4 Sampling

To characterize the performance of different RAP solutions on typical problem instances of the two communities analyzed, one must evaluate the algorithms on a random sample of problem instances from each community. This translates into analyzing a set of problem instances defined at random times. Note that the future and past time windows for seeding library estimation must be contained in the trace for every selected time. Furthermore, one should allow the set of possible times to account for most seasonality in the data. Most short-term seasonality in BitTorrent usage is daily or weekly. We therefore sample instants from a set of instants covering one whole week of each trace. The time window for seeding library estimation is 28 days for both past and future in both traces. Throughout this paper, we use a sample of 45 states for Filelist and 55 states for Bitsoup.

6 Can current algorithms provide high throughput?

In this section, we present experiments for evaluating the performance of current SSAP and BAP solutions in the context of file sharing communities interested in maximizing throughput. The metric used to compare solutions is the aggregate download speed of all sessions.

6.1 Seeding sessions allocation

The first experiment assesses whether the current solution for SSAP limits the performance of solutions for the complete resource allocation problem. We devise an upper bound for RAP solutions by removing the seeding capacity constraint for all users and applying the MaxFlow algorithm on the original possible user connections set.

The experiment then compares the performance of a community using Current SSAP and MaxFlow BAP (Current+MaxFlow) to the established upper bound. If Current+MaxFlow performs similarly to the upper bound, it is possible to affirm that current seeding sessions allocation does not hamper the performance of a complete solution for RAP.

However, comparing the current method for seeding sessions allocation and an upper bound does not allow us to understand if the current method could perform worse than it does. It may happen that the current method performs optimally in the experiment because the space of possible allocations does not allow a different outcome. We test for this possibility by also comparing the results of Current+MaxFlow

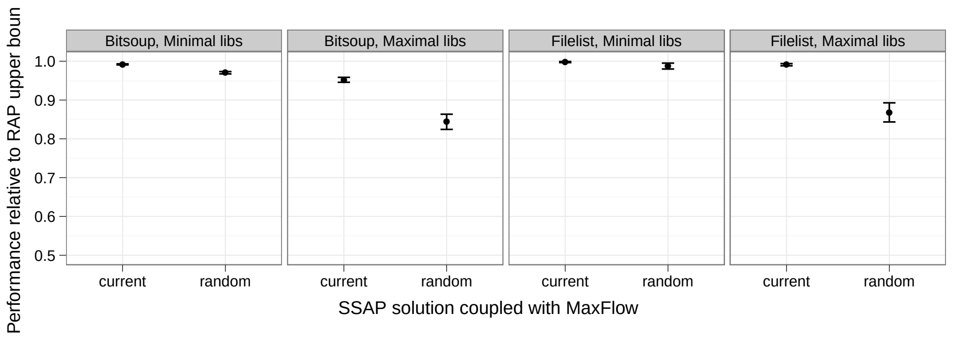

Figure 3: Throughput produced by current and random SSAP solutions coupled with MaxFlow and relative to the performance upper bound for each RAP instance (means with 95% confidence intervals).

to those produced by a random seeding sessions allocation coupled with the MaxFlow BAP algorithm (Random+MaxFlow).

The results of these comparisons are depicted in Figure 3. Performance is measured as the aggregate download performance of all users, and the relative performance of a method in relation to the upper bound is calculated by dividing the performance of that method by the upper bound for each problem instance considered.

Overall, there is negligible difference between Current+MaxFlow and the upper bound in all scenarios. It follows that it is possible to attain high performance using current SSAP solutions. This is notable given that these solutions ignore bandwidth information. Our results suggest that an efficient bandwidth allocation can cope with this limitation.

Observing the limited difference between Random+MaxFlow and the upper bound suggests that Current achieves such high performance mainly due to a reduced space of possibilities. The results are very similar for Bitsoup and Filelist, despite the differences in number of torrents and sessions noted in Table 2.

6.2 Bandwidth allocation

The second experiment examines if the current bandwidth allocation coupled with the current SSAP solution (Current+Current) approximates the maximum throughput achievable starting from the current seeding sessions allocation (Current+MaxFlow). Figure 4 presents the results of this experiment.

In general, the results are significantly lower when Current BAP is applied. Throughput of Current+Current lies between 65% and 80% of Current+MaxFlow. We note that Filelist performs marginally worse than Bitsoup; this could be the consequence of the difference between the ratio of sessions to torrents in the two communities.

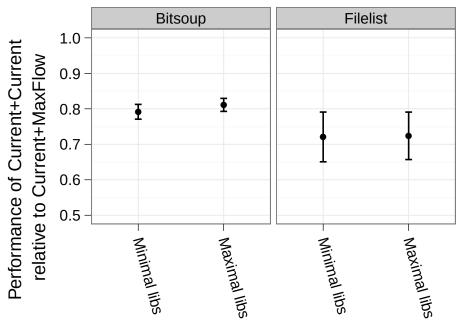

Figure 4: Throughput produced by the current BAP solution coupled with Current SSAP and relative to the performance of Current+MaxFlow (means with 95% confidence intervals).

7 Are current algorithms appropriate for streaming?

This section evaluates how current methods perform when considering the video streaming use case. In this use case, the ideal resource allocation is an allocation that is max-min fair with respect to the n worst performing leeching sessions. Such an allocation provides the best possible service for the n sessions most exposed to streaming interruptions while guaranteeing that the rest of the sessions obtain a service at least as good as the best-performing of the n sessions.

In the experiments in this section, we use the performance of the 5th percentile worst-performing session to compare the allocations to the max-min fair allocation for the same scenario.

7.1 Torrent selection

Our approach is analogous to that used to evaluate total throughput in Section 6. First, we determine whether the current SSAP solution hinders the performance of the optimal BAP solution. For this, we compare Current+MaxMin to an unconstrained solution based on MaxMin that utilizes the complete seeding library of each user as that user's active seeding set. Second, we establish the extent to which the actual SSAP solution can affect the overall RAP solution by analyzing the results of a random seeding sessions allocation (Random+MaxMin) in relation to the unconstrained MaxMin solution.

Inspecting the results in Figure 5, we see negligible differences between current, random and unrestricted seeding sessions allocations. This suggests the Current is adequate for maximizing the 5th percentile performance. At the same time, the random selection results again point to a limited potential for choice.

Similarly to maximizing throughput, solving SSAP without bandwidth information does not affect streaming when an efficient BAP solution is used.

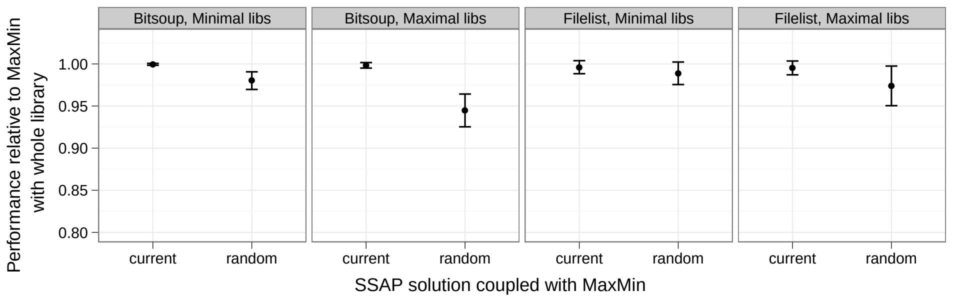

Figure 5: Fifth percentile session download speed produced by current and random SSAP solutions coupled with MaxMin BAP and relative to unrestricted MaxMin (means with 95% confidence intervals).

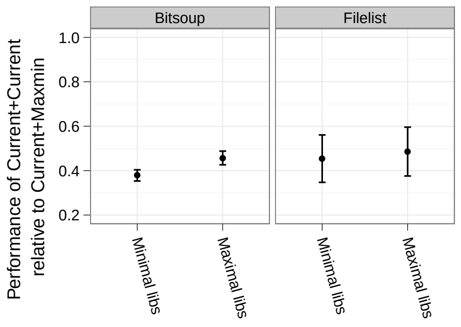

Figure 6: Fifth percentile session download speed produced by the current BAP solution coupled with Current SSAP and relative to the performance of Current+MaxMin (means with 95% confidence intervals).

7.2 Bandwidth allocation

In the second experiment regarding the streaming use case, we investigate performance of current bandwidth allocation solutions. Starting with current SSAP solution, we compare the allocation produced by the current methods (Current+Current) to the optimal allocation produced by our MaxMin algorithm (Current+MaxMin). The results are depicted in Figure 6.

For all scenarios, results are at most 60% of optimal. In case of the minimal libraries scenario in the Bitsoup community, the service received by the 5th percentile session is only 35–40% of the maximum possible. Overall, the current BAP solution is far from ideal for the streaming use case.

8 Related work

Considerable research and development effort has been invested in designing and evaluating BitTorrent's intra-swarm resouce allocation methods. In the paper introducing the BitTorrent protocol, Cohen [3] emphasizes the role of efficient upload bandwidth utilization. Experimental investigations by Legout et al. suggest that the current algorithms for choosing upload partners inside a swarm need no further improvement [9] and documents the high bandwidth utilization of upload bandwidth inside a swarm [8].

BitTorrent has also been studied at the community level. Zhang et al. [14] show how an entire ecosystem forms around the P2P protocol. Guo et al. [6] and Andrade et al. [1] analyze traces of multiple BitTorrent communities. Nevertheless, previous work investigating multi-swarm systems has not characterized the community-level metrics we consider or evaluated the effect of current inter-swarm resource allocation mechanisms.

More similar to our work, Dunn et al. [5] explore seeding session selection 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 present SSAP solutions are relevant for the problem instances most common in real communities.

Peterson et al. [11] 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. Our results corroborate present BAP solutions lead to inefficient outcomes, adding that this happens in the problem instances found in real communities. Furthermore, we expand Peterson et al.'s results examining the video-streaming use case.

9 Conclusion

In this paper, we present an evaluation of present *de facto* solutions for inter-swarm resource allocation in BitTorrent communities. First we formulate the resource allocation problem and its tractable decomposition: seeding sessions allocation combined with bandwidth allocation. Next, we identify the solutions for this problems implemented in prevalent BitTorrent clients. We develop simulations to approximate these solutions.

In order to evaluate current performance, we devise performance upper bounds for the resource allocation problem. This is possible by isolating the bandwidth allocation problem and mapping BitTorrent communities to flow networks, which allows for the use of well-known graph-theoretical techniques.

Considering the file-sharing community use case, our results suggest current SSAP solutions are adequate, while we notice the possibility for improvement in present BAP solutions. In a way, current BAP solutions highlight there is currently a price for anarchy: with individuals allocating resources solely in their own interest, they fulfill the global objective less efficiently than a centralized algorithm is capable of.

For the experiments analyzing the streaming use case, our results point out that current SSAP solutions are also sufficient. On the other hand, present algorithms for BAP perform poorly, with sizeable room for improvement.

References

[1] N. Andrade, E. Santos-Neto, F. Brasileiro, and M. Ripeanu. Resource demand and supply in BitTorrent content-sharing communities. *Computer Networks*, 53(4):515–527, 2009. 4, 14, 19

[2] D. Bertsekas and R. Gallager. *Data networks*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1992. 12

[3] B. Cohen. Incentives build robustness in BitTorrent. In *Workshop on Economics of Peer-to-Peer systems*, volume 6, pages 68–72, 2003. 19

[4] R. Cottle, E. Johnson, and R. Wets. George B. Dantzig (1914–2005). *Notices of the American Mathematical Society*, 54:344–362, 2007. 12

[5] R. J. Dunn, S. D. Gribble, and H. M. Levy. The importance of history in a media delivery system. In *Proceedings of IPTPS 2007*, 2007. 19

[6] L. Guo, S. Chen, Z. Xiao, E. Tan, X. Ding, and X. Zhang. Measurements, analysis, and modeling of BitTorrent-like systems. In *Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*, page 4. USENIX Association, 2005. 4, 19

[7] T. Isdal, M. Piatek, A. Krishnamurthy, and T. Anderson. Leveraging BitTorrent for end host measurements. *Passive and Active Network Measurement*, pages 32–41, 2007. 14

[8] A. Legout, N. Liogkas, E. Kohler, and L. Zhang. Clustering and sharing incentives in BitTorrent systems. In *Proceedings of SIGMETRICS '07*, pages 301–312. ACM, 2007. 19

[9] A. Legout, G. Urvoy-Keller, and P. Michiardi. Rarest first and choke algorithms are enough. In *SIGCOMM '06*, pages 203–216. ACM, 2006. 19

[10] MosekApS. Mosek optimization software. `www.mosek.com`. 12

[11] R. S. Peterson and E. G. Sirer. Antfarm: Efficient content distribution with managed swarms. In *Procedings of NSDI 09*, pages 107–122, 2009. 19

[12] B. Radunović and J.-Y. L. Boudec. A unified framework for max-min and min-max fairness with applications. *IEEE/ACM Trans. Netw.*, 15:1073–1083, October 2007. 12

[13] B. Zhang, A. Iosup, J. Pouwelse, D. Epema, and H. Sips. Sampling bias in bittorrent measurements. In *Euro-Par'10*, pages 484–496, 2010. 14

[14] C. Zhang, P. Dhungel, and K. Di Wu. Unraveling the bittorrent ecosystem. *IEEE Transactions on Parallel and Distributed Systems*, 2010. 4, 7, 19