Centralized and Distributed Protocols for Tracker-based Dynamic Swarm Management

György Dán and Niklas Carlsson

***Abstract*—With BitTorrent, efficient peer upload utilization is achieved by splitting contents into many small pieces, each of which may be downloaded from different peers within the same swarm. Unfortunately, piece and bandwidth availability may cause the file sharing efficiency to degrade in small swarms with few participating peers. Using extensive measurements, we identified hundreds of thousands of torrents with several small swarms for which re-allocating peers among swarms and/or modifying the peer behavior could significantly improve the system performance. Motivated by this observation, we propose a centralized and a distributed protocol for dynamic swarm management. The centralized protocol (*CSM*) manages the swarms of peers at minimal tracker overhead. The distributed protocol (*DSM*) manages the swarms of peers while ensuring load fairness among the trackers. Both protocols achieve their performance improvements by identifying and merging small swarms, and allow load sharing for large torrents. Our evaluations are based on measurement data collected during eight days from over 700 trackers worldwide, which collectively maintain state information about 2.8 million unique torrents. We find that *CSM* and *DSM* can achieve most of the performance gains of dynamic swarm management. These gains are estimated to be up to 40% on average for small torrents.**

***Index Terms*—Peer-to-peer overlay management, Tracker-based protocol, BitTorrent performance, Content popularity**

I. Introduction

While there has been some controversy around the use of file sharing applications for distributing copyrighted content, peer-to-peer file sharing remains highly popular and is responsible for a large fraction of the overall network traffic. Today, BitTorrent is the dominant peer-to-peer file sharing protocol. BitTorrent has proven to be highly scalable due to its distributed peer-to-peer architecture and its splitting of content (files) into many small pieces that each may be downloaded from different peers. The content and the set of peers distributing it is usually called a *torrent*. To facilitate efficient file sharing, BitTorrent relies on a set of *trackers* to voluntarily maintain state information about all peers currently having pieces of a particular file. A client that wants to download a file can learn about other peers that share the same content by contacting a *tracker*.

In order to increase system availability [1], BitTorrent allows multiple trackers to be associated with each file (Multitracker Metadata Extension, BEP-0012 [2]). To avoid overloading trackers, the protocol only allows a peer to be associated with one tracker per file that it is downloading. Consequently, every peer that participates in sharing a file is member of a *swarm*, which is tracked by a tracker, and multiple swarms associated to a single file can coexist in parallel. The set of all swarms and thus all peers sharing the same file is referred to as a *torrent*.

While BitTorrent allows peers to effectively share pieces of popular content with many peers in a swarm, the performance of small torrents and swarms is sensitive to fluctuations in peer participation. Measurements and analysis have shown that peers in small (less popular) swarms achieve lower throughput on average (e.g., Figure 6, Figure 7 and [3], [4]). Most swarms are unfortunately small; several sources indicate that the popularity distribution of peer-to-peer content follows a power-law form, with a "long tail" of moderately popular files (see Figure 1(b) and, e.g., [5], [6], [7], [8]). At the same time, the measurement data we present in this paper shows that many torrents consist of several swarms (see Figure 2). Potentially, if one could dynamically re-allocate peers among the trackers such that multiple small swarms of a torrent are merged into a single swarm, then one could improve the file sharing performance of the peers belonging to these torrents.

Motivated by these observations, the goal of our work is to evaluate the feasibility and the potential gains of dynamic swarm management for BitTorrent trackers. We propose a dynamic swarm management method that is based on merging small swarms periodically. We formulate two versions of the problem of merging small swarms as optimization problems, and propose a centralized and a distributed swarm management protocol, called *CSM* and *DSM*, respectively, to solve the problems. We use the two protocols to evaluate the feasibility and the potential gains of dynamic swarm management based on eight days of measurements of 721 trackers, which collectively maintain state information of a combined total of 2.8 million unique torrents and approximately $20-60$ million concurrent peers. Collectively, these torrents have supported approximately 4.52 billion downloads. Based on the protocol evaluation we argue that dynamic swarm management could lead to a significant performance improvement in terms of peer throughput at a very low cost in terms of overhead.

Throughout the paper we use BitTorrent terminology to refer to the components of the content distribution system, and use BitTorrent to quantify the achievable gain of our protocols. Nevertheless, the proposed protocols are in general applicable to tracker-based content delivery systems that aim to (i) maintain high content availability by using redundant, highly autonomous trackers, (ii) minimize the performance

G. Dán is with the School of Electrical Engineering, KTH Royal Institute of Technology, Stockholm, Sweden. E-mail: gyuri@ee.kth.se

N. Carlsson is with Linköping University, Linköping, Sweden. E-mail: niklas.carlsson@liu.se

[0]The work was supported by the ACCESS Linnaeus Centre at KTH and by CENIIT at Linköping University.

penalty due to partitioned swarms of peers, and (iii) avoid high traffic overhead.

The remainder of the paper is organized as follows. Section II describes the basics of tracker-based overlay management and our system model. Section III presents results from a large scale measurement of BitTorrent that motivate our work. Section IV formulates our design objectives based on the measurement dataset. In Section V we describe the proposed protocols to perform dynamic swarm management. Sections VI and VII present analytical, simulation based and trace-based performance results. Related work is discussed in Section VIII, and Section IX concludes the paper.

II. System Description

A. Peers, Swarms and Torrents

BitTorrent is a popular peer-to-peer file-sharing protocol that has been shown to scale well to very large peer populations. With BitTorrent, files are split into many small pieces, each of which may be downloaded from different peers. Peers that have the entire file (or content) and only upload content are called *seeds*, while peers that only have parts of the file and are downloading are called *leechers*. The set of peers that share a particular file is usually called a *torrent*.

Trackers are used to maintain state information about the peers currently having pieces of a particular file. A client that wants to download a file can learn about other peers that share the same file by contacting a *tracker* node at its *announce URL*. Upon request, the tracker provides the peer with a subset of the known peers. In exchange, the peer has to provide the tracker with information about its download progress. Additionally, the peers must inform the tracker about changes in their status (i.e., when they join, leave, or finish downloading). Trackers also have a *scrape URL* at which they answer scrape requests to provide information about a swarm: the number of seeds, the number of leechers, and the number of download completions.

The set of peers that share a particular file and are tracked by the same tracker are called a *swarm*. In order to increase the availability of files BitTorrent allows several trackers to track peers sharing the same file; i.e., several swarms can coexist within the same torrent. The set of trackers is determined by the creator of the torrent, who includes the list of trackers in the *torrent metadata file*. Such redundancy allows trackers to be run on relatively cheap, commodity hardware and network equipment. Upon joining, a peer knows about the list of trackers from the *torrent metadata file* and it has to contact *one* tracker at random (if the tracker is no longer available then a new tracker must be contacted). This way the load is balanced between the available trackers, and the tracker load is kept low. *As a consequence of these design choices, peers in different swarms are typically not aware of each other, even though they share the same content.*

Besides the trackers, peers may learn about other peers in the *same* swarm using Peer Exchange (PEX), and can use a distributed hash table (DHT) to learn about peers in the same torrent. In the case of private torrents, which are torrents meant for sharing content by registered users only, PEX and DHT are disabled in order to avoid unauthorized peers from joining the torrent, and thus trackers are the only means of peer discovery.

Parameter	Definition
$\mathcal{R}$	Set of trackers
$\mathcal{T}$	Set of torrents
$\mathcal{R}(t)$	Set of trackers that track torrent t
$\mathcal{T}(r)$	Set of torrents that are tracked by tracker r
$\mathcal{T}(r,r')$	Set of torrents tracked both by tracker r and r'
N_r	Number of peers tracked by tracker r
x_t	Number of peers associated with torrent t
$x_{t,r}$	Number of peers of torrent t that are tracked by tracker r
$\tilde{x}$	Threshold parameter

TABLE I: Frequently used notation

B. System Model

We model the system with a set of trackers $\mathcal{R}$, with a combined set of torrents $\mathcal{T}$. We will denote by $\mathcal{R}(t)$ the set of trackers that track torrent $t \in \mathcal{T}$, and by $\mathcal{T}(r)$ the set of torrents that are tracked by tracker $r \in \mathcal{R}$. Every torrent is tracked by at least one tracker (i.e., $\mathcal{T} = \bigcup_{r\in\mathcal{R}} \mathcal{T}(r)$), but the subsets $\mathcal{T}(r)$ are not necessarily pairwise disjoint. For two trackers r and r' we denote the set of torrents that both trackers track by $\mathcal{T}(r,r') = \mathcal{T}(r) \bigcap \mathcal{T}(r')$. The trackers form a graph $\mathcal{G} = (\mathcal{R},E)$ in which there is an edge between two vertices (trackers) r and r' if $\mathcal{T}(r,r') \neq \emptyset$. We will refer to this graph as the *tracker graph*. Finally, let us denote the number of peers tracked by tracker r for torrent t by $x_{t,r}$, the total number of peers tracked by tracker r by N_r, and the total number of peers associated with torrent t by x_t. Table I summarizes the notation used in the paper.

III. A Measurement Study of Swarms and Torrents

In this section we present the measurement data motivating our work. We first describe the measurement methodology used to collect the data, then present swarm and torrent level statistics based on the observed popularity of approximately three million BitTorrent torrents.

A. Measurement Methodology

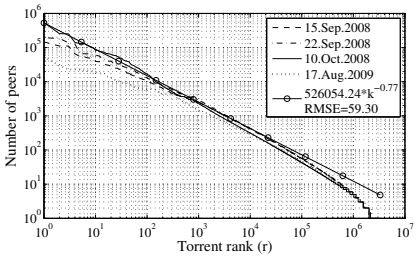
We used two kinds of measurements to obtain our dataset. First, we performed a screen-scrape of the torrent search engine *www.mininova.org*. In addition to claiming to be the largest torrent search engine, *mininova* was the most popular torrent search engine according to *www.alexa.com* during our measurement period (Alexa-rank of 75, August 1, 2008). From the screen-scrapes we obtained the sizes of about 330,000 files shared using BitTorrent, and the addresses of 1,690 trackers.

Second, we scraped all the 1,690 trackers for peer and download information of all the torrents they maintain. For the tracker-scrapes we developed a Java application that scrapes the scrape URL of each tracker. By not specifying any info-hash, the tracker returns the scrape information for all torrents that it tracks. This allowed us to efficiently obtain the number of leechers, seeds, and completed downloads as seen by all trackers that we determined via the screen-scrape of *mininova*. Scrapes were performed in parallel, the longest scrape took about 20 minutes, so our dataset is in fact a sequence of snapshots of BitTorrent content popularity.

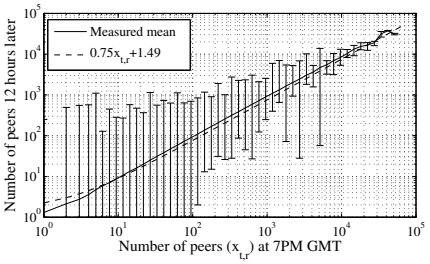
We performed the tracker-scrapes weekly from September 15, 2008 to August 17, 2009, and daily between October

Item	Value
Total trackers	1,690
Unique trackers	721
Unique torrents	2,864,073
Unique swarms	3,309,874
Number of leechers	21,088,533
Number of seeds	16,985,796
Total downloads	$4.52 \cdot 10^9$

(a) Summary of a complete snapshot on Oct. 10, 2008.



(b) Torrent popularity vs. rank at 4 different dates



(c) Change of popularity between 7pm GMT on Oct. 10, 2008 and 12 hours later.

Fig. 1: Basic properties of the multi-torrent, multi-swarm system: (a) summary of a snapshot, (b) content popularity on four dates, and (c) its change.

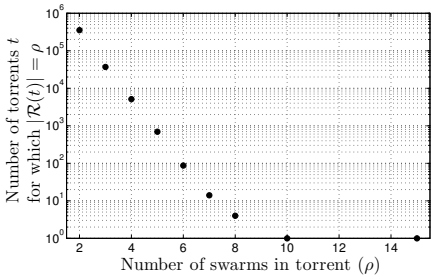


Fig. 2: Number of torrents vs. number of swarms per torrent on Oct. 10, 2008.

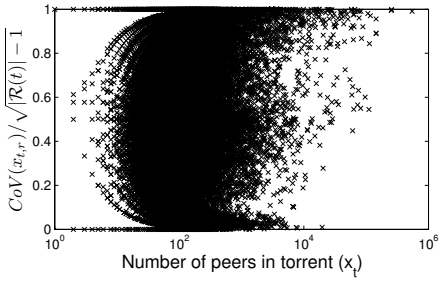


Fig. 3: Normalized coefficient of variation of swarms sizes on Oct. 10, 2008.

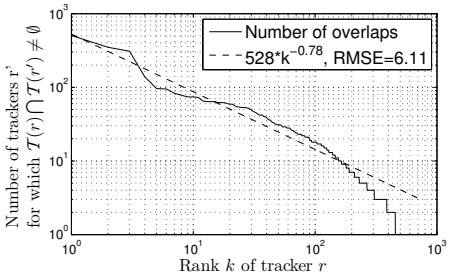


Fig. 4: Number of trackers that have overlapping torrents vs. rank on Oct. 10, 2008.

10, 2008 and October 17, 2008. All scrapes were performed at 7pm GMT. Additionally, we performed hourly scrapes between September 24 and October 15, 2008. To identify redundant tracker-scrapes (stemming from trackers with multiple hostnames, for example) in our traces, we performed a correlation test between every pair of scrapes, and tagged scrapes that had an overlap of at least 90% in terms of the observed torrents, and a correlation coefficient above 0.9 in terms of the number of leechers and seeds as redundant. This way we removed redundant information about the same swarms of peers, and identified 721 independent trackers. The table in Fig 1(a) summarizes the dataset obtained on October 10, 2008.

B. Swarms and Torrents

Figures 1 to 5 summarize some of the main characteristics of the world of trackers and torrents captured by our measurements. Figure 1(b) shows the rank plot of the number of peers per swarm at four different dates during our measurement period. The number of swarms observed did not change significantly in 11 month, neither did the shape of the rank popularity curves. The figure also shows the least-squares best-fit Zipf distribution and the respective root mean squared error for October 10, 2008. We refer to [9] for a detailed study of the swarm size distribution and for a discussion of whether the distribution has a power-law tail. Nevertheless, similar to previous content popularity studies (e.g., [5], [6], [7], [8]), it stands out that there is a substantial number of torrents with moderate popularity: about 2.84 million of the 2.86 million torrents observed at 7pm GMT on October 10, 2008 have less than 200 peers, and about 50% of the peers are in these torrents. The ratio of peers in torrents with less than 200 peers changes over time; it is significantly higher (up to 75%) in the morning hours when torrent popularities are lower. Figure 1(c) illustrates the change of content popularity over 12 hours. Swarms were put into 50 logarithmically spaced bins based on the number of peers at 7pm GMT on October 10, 2008, and the solid curve shows the average number of peers per swarm 12 hours later. The minimum and maximum number of peers are shown by the error bars for each category. The least-squares best linear fit shows that the number of peers in the swarms is 25 percent lower on average at 7am GMT than 12 hours earlier, and interestingly, the decrease is close to uniform independent of the original swarm size. Similar diurnal fluctuations were reported recently in [10].

Figure 2 shows the number of torrents with a given number of unique swarms (after removing duplicates). Clearly, there are a substantial number of torrents that are served independently by multiple trackers. In order to get an impression of how the peers are dispersed over the swarms of the same torrent, we calculated the normalized coefficient of variation of the swarm sizes of each multi-tracked torrent as

$$CoV_t^* = \frac{CoV(x_{t,r})}{\sqrt{|\mathcal{R}(t)-1|}}, \qquad (1)$$

where $CoV(x_{t,r})$ is the coefficient of variation of the swarm sizes of torrent t, i.e., the standard deviation of the swarm sizes divided by their mean. Due to the normalization $CoV_t^* = 0$ if all swarms of a torrent have the same size, and $CoV_t^* = 1$ if all peers are in one of the many swarms of a torrent.

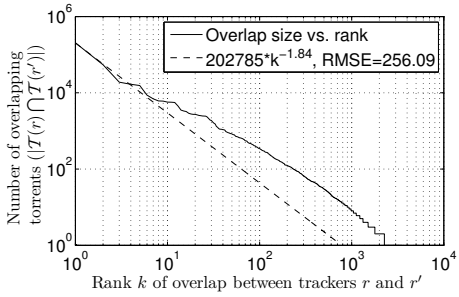


Fig. 5: Number of overlapping torrents between pairs of trackers vs. rank on Oct. 10, 2008.

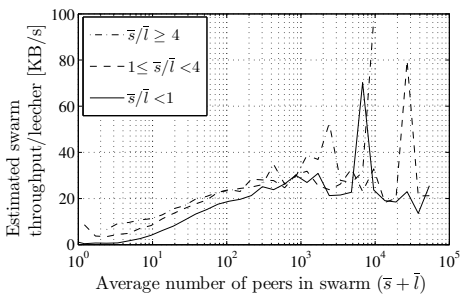


Fig. 6: Throughput estimates for three classes of swarms starting at 7pm GMT, Oct. 10, 2008.

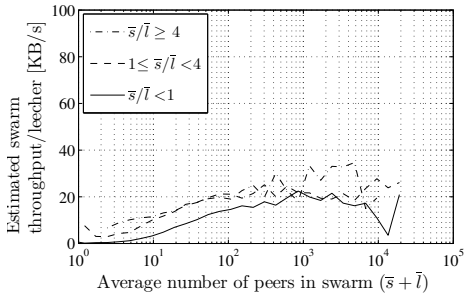


Fig. 7: Throughput estimates for three classes of swarms starting at 7am GMT, Oct. 11, 2008.

Figure 3 shows the normalized coefficient of variation of the swarm sizes versus the torrent size, of each observed torrent. While the normalized CoV only can take a finite number of values (hence the distinguishable patterns), this figure shows that there are many small torrents in which the peers are almost equally spread among the swarms.

Finally, we look at the characteristics of the tracker graph measured on October 10, 2008. There are 550 trackers that have an overlap with at least one tracker, and thus, the tracker graph has 550 vertices, 3,535 edges and an average node degree of 12.8. The average of the local clustering coefficient [11] of the nodes is 0.66, which indicates that there are many cliques of size three. The diameter of the graph is 5, its radius is 3, and its characteristic path length is 2.11. These low values could be a sign of the scale-free nature of the graph [12]. Figure 4 shows the number of trackers $|\{r'|\mathcal{T}(r)\cap\mathcal{T}(r')\neq\emptyset\}|$ with which a tracker r has overlapping torrents ranked in decreasing order, that is, it shows the rank degree statistic of the tracker graph. There are a few trackers that have overlapping torrents with many other trackers, but there are many trackers with a few overlapping trackers. The number of overlapping trackers decreases slower than exponentially with the rank, but it does not clearly follow a power-law distribution. As a comparison we plotted the least-squares best-fit Zipf distribution and its root mean squared error (RMSE). These characteristics of the tracker graph topology may influence the effectiveness of swarm management protocols. For example, in a distributed protocol, the communication overhead of a tracker might be proportional to its degree. We will discuss the importance of the trackers that have an overlap with many trackers, and of the tracker graph topology in general, in Section VII-A.

Figure 5 shows the number of torrents that are in common between any pair of trackers that have overlapping torrents (i.e., $|\mathcal{T}(r,r')|$) as a function of their ranks. The rank statistics approximately follows a power-law distribution, the least-squares estimates of the parameters of the least-squares best-fit Zipf distribution and the corresponding root mean squared error are shown in the figure. We note that there are many small overlaps, that is, there are many pairs of trackers that have only a few torrents in common. The power-law characteristics also have the desirable property that a few of the trackers (and their overlaps) are responsible for the majority of the overlaps. In Section VII-A we will leverage these observations in order to decrease the total amount of tracker-to-tracker communication and to speed up our protocols.

C. Throughput Estimation

To estimate the throughput of any particular swarm we used the number of seeds, leechers, and cumulative number of downloads measured at two time instants. Using the two snapshots we estimated the download throughput per leecher, as the file size L divided by the estimated download (service) time S. Using Little's law the download time can be expressed as $S=\bar{l}/\mathtt{l}$, where $\bar{l}$ is the average number of leechers being served in the system and $\mathtt{l}$ the average peer arrival rate. Due to flow conservation, the peer arrival rate can be estimated as the overall download completion rate, equal to the number of download completions D between two consecutive measurements, divided by the time T between the two consecutive measurements. To summarize, we have an estimated throughput of $\frac{LD}{\bar{l}T}$. Finally, throughput estimates for any particular swarm size were obtained by taking the average over all swarms of that particular size.

Figures 6 and 7 show throughput estimation results based on two $T=2$ hours long intervals starting at 7pm GMT on October 10, 2008 and at 7am GMT on October 11, 2008, respectively. Swarms are classified based on the average number of peers in the swarm $\bar{s}+\bar{l}$ and the seed-to-leecher ratio $\frac{\bar{s}}{\bar{l}}$; 60 bins are used for the swarm sizes and three curves are used for the different seed-to-leecher ratios. Note that the possible ranges of $\bar{s}+\bar{l}$ values are different for different seed-to-leecher ratios, and hence result in different start values for the curves. We note that the results for swarms up to just over 1,000 peers are similar in both figures and are consistent with intuition and previous studies [3], [4]. Both figures show that the estimated throughput increases with the swarm size. The measurement data over these two intervals are also consistent in that the average number of downloads per leecher in swarms with at least 200 peers is almost twice the average number of downloads per leecher in swarms with less than 200 peers. Assuming that the size of a file does not influence its popularity, this can be translated to an approximately two times higher average throughput in swarms with at least 200 peers. While our focus in this paper is not to investigate the reason of this phenomenon, we attribute it to two reasons. First, as the number of peers in a swarm increases, peers can find more nearby peers to exchange data with. As a consequence, the

round trip times between the peers are lower on average, which allows higher TCP throughput to be achieved. Second, when peer upload rates are heterogeneous, the probability that peers with similar upload rates exist increases with the swarm size. For small swarms, piece availability can also become an issue.

For larger swarm sizes, however, the results are rather noisy. The reason is that the estimation for large swarms is less accurate due to fewer samples. For example, while the number of swarms smaller than $1,000$ peers for which we could obtain content size information from *mininova.org* is around $350,000$ for both intervals, there are only about 500 and 1500 swarms that are larger than $1,000$ peers on the two intervals, respectively. The lack of statistical significance for swarm sizes above $1,000$ peers is thus the reason for the large fluctuations of the curves in both figures for big swarm sizes.

In the rest of the paper our focus is on swarms smaller than 1000 peers for which the throughput estimates are similar in the two figures: both figures show that bigger swarms achieve higher throughput on average, irrespective of the seed-to-leecher ratio and the time of the day.

IV. Dynamic Swarm Management

In the following we describe the rationale for dynamic swarm management and a high level overview of its operation. We then define the key performance metrics and formulate two versions of dynamic swarm management as optimization problems.

A. The Rationale for Dynamic Swarm Management

We can make three important observations based on the measurement dataset presented in Section III. First, the majority of swarms and torrents are rather small and a large fraction of the peers belong to such small swarms: in our dataset 99 percent of torrents have less than 200 peers and about 50 to 75 percent of the peers are in these torrents, depending on the time of the day. Second, many torrents consist of several swarms tracked by different trackers: in our dataset there are about 300 thousand multi-tracked torrents. Third, irrespective of the seed-to-leecher ratio, small swarms achieve lower throughput on average than larger swarms as shown in Figures 6 and 7.

At the same time, in order to increase system availability it may be advantageous to split the responsibility of maintaining per-peer state information across multiple trackers; i.e., to allow several swarms to coexist. If peers follow the BitTorrent protocol and only associate with one tracker upon arrival then peers in different swarms will not be aware of each other. Alternatively, if peers do not conform to the protocol but associate with all trackers then they will know about each other, but the trackers' load increases proportional to $\sum_{t\in\mathcal{T}} x_t(|\mathcal{R}(t)|-1)$, i.e. the number of swarms per torrent weighted by the torrent sizes.

Based on the above observations, we formulate the goal of dynamic swarm management as being able to merge swarms belonging to a torrent if they become too "small". In general, it is hard to define when a swarm should be considered "small", but for simplicity, we assume that a swarm can be considered "small" if it has less than $\tilde{x}$ participating peers, for some threshold $\tilde{x}$. Non-"small" swarms are likely to have high piece diversity and are more resilient to fluctuations in peer participation. Intuitively, one would like to minimize the number of swarms (t,r) for which $x_{t,r} < \tilde{x}$.

A straightforward solution to avoid "small" swarms is to merge all swarms of every torrent. This solution has two potential drawbacks. First, it can lead to many peers being re-assigned unnecessarily between trackers and to large amounts of control traffic. Second, it can lead to an increase of the number of peers assigned to certain trackers, which can be a problem if trackers are managed by autonomous entities.

A better solution is to periodically merge small swarms and to split big swarms on-demand. The splitting of individual swarms is motivated by the potential issue that a single tracker might see an increase of its load because of a single torrent becoming very popular. An easy way to handle such instances is that a tracker stops being the sole responsible for its biggest torrent(s) when its load reaches a threshold ($\hat{x}$). New peers that arrive to the torrent can be redirected to other trackers. The problem of merging swarms is, however, more complex and the impact of merging on the system performance can be noticeable as it involves the reallocation of peers between trackers. In the rest of the paper we focus on the problem of merging small swarms and leave the problem of splitting big swarms to be a subject of future work. In the following we give a mathematical formulation of the problem of merging small swarms.

B. Performance Metrics

We start the problem formulation with the definition of three metrics that we use to capture the efficiency of swarm management: the number of "small" swarms, the overhead of swarm management and the tracker load.

Number of "Small" Swarms: The primary goal of dynamic swarm management is to minimize the number of "small" swarms. Let us denote by $y_{t,r}$ the number of peers tracked by tracker r for torrent t after dynamic swarm management. Then the number of small swarms belonging to torrent $t \in \mathcal{T}$ can be expressed as

$$|\{r|y_{t,r} < \tilde{x}, y_{t,r} \neq 0\}|. \tag{2}$$

Management Overhead: Dynamic swarm management re-assigns peers between trackers. Reassigning peers between trackers leads to increased control traffic. We quantify the management overhead as the number of peers reassigned between trackers. For a torrent $t \in \mathcal{T}$ the management overhead can be expressed as

$$\sum_{r\in\mathcal{R}(t)} [(x_{t,r} - y_{t,r})]^+. \tag{3}$$

Tracker Load: As dynamic swarm management reassigns peers between trackers it may change the number of peers assigned to the individual trackers, and hence the traffic load of the trackers. We quantify the traffic load of a tracker $r \in \mathcal{R}$ with the number of peers it tracks

$$N_r = \sum_{t\in\mathcal{T}(r)} x_{t,r}. \tag{4}$$

C. Swarm Management as an Optimization Problem

We consider two versions of the problem of merging swarms motivated by two application scenarios. The first scenario is a system managed by a single entity. The goal is to merge the swarms at minimal overhead, and merging can be done in a centralized or in a decentralized fashion. The second scenario is a system of autonomous trackers. In this scenario merging should maintain the tracker loads unchanged, and due to the trackers' autonomy, a distributed solution is more likely to be of practical interest. In the following we use the above three metrics to formulate the two versions of dynamic swarm management as optimization problems.

Minimum Management Overhead (MMO): The first form of the problem assumes that the primary concern is to minimize the number of "small" swarms with the least amount of management overhead possible. The optimization problem can be formulated as

$$\min \sum_{t\in\mathcal{T}} \sum_{r\in\mathcal{R}(t)} [(x_{t,r} - y_{t,r})]^+, \tag{5}$$

s.t.

$$|\{r|y_{t,r} < \tilde{x}, y_{t,r} \neq 0\}| \leq \min(1, \max(0, \tilde{x} - \sum_{r\in\mathcal{R}(t)} x_{t,r})) \quad \forall t \in \mathcal{T}$$

$$\sum_{r\in\mathcal{R}(t)} y_{t,r} = x_t, \quad \forall t \in \mathcal{T}.$$

Note that for a particular torrent t if $\sum_{r\in\mathcal{R}(t)} x_{t,r} < \tilde{x}$ then the minimum number of "small" swarms is 1, otherwise it is 0.

Minimum Management Overhead with Load Maintenance (MMO-LM): The second form of the problem also considers the tracker load, but also ensures that the load of the individual trackers remains unchanged after performing swarm management. The optimization problem can be formulated as

$$\min \sum_{t\in\mathcal{T}} \sum_{r\in\mathcal{R}(t)} [(x_{t,r} - y_{t,r})]^+, \tag{6}$$

s.t.

$$|\{r|y_{t,r} < \tilde{x}, y_{t,r} \neq 0\}| \leq \min(1, \max(0, \tilde{x} - \sum_{r\in\mathcal{R}(t)} x_{t,r})) \quad \forall t \in \mathcal{T}$$

$$\sum_{r\in\mathcal{R}(t)} y_{t,r} = x_t, \quad \forall t \in \mathcal{T},$$

$$\sum_{t\in\mathcal{T}(r)} y_{t,r} = \sum_{t\in\mathcal{T}(r)} x_{t,r}, \quad \forall r \in \mathcal{R}.$$

While in the case of MMO the optimization can be performed independently for different torrents, in the case of MMO-LM the optimization has to be performed jointly over all torrents, which makes the optimization problem more complex.

V. Swarm Merging Protocols

We describe two protocols to solve the two optimization problems formulated in the previous section. The first protocol, called Centralized Swarm Management (*CSM*) protocol, is centralized and requires global knowledge to solve the *MMO* problem. *CSM* relies on the Minimum Excess Deficit (MED) algorithm to calculate the solution to the *MMO* problem for each torrent. We then devise a distributed protocol that suits a system without centralized control of the trackers and without global state information. This decentralized swarm management protocol, called the Distributed Swarm Management (*DSM*) protocol, moves peers that belong to the same torrent between pairs of trackers. The algorithm can be used to obtain an approximate solution to the *MMO* and to the *MMO-LM* optimization problems.

A. Centralized Swarm Management Protocol (CSM)

The *CSM* protocol works by first performing a scrape of all trackers to obtain the number $x_{t,r}$ of peers per swarm. The central server then solves the *MMO* problem for every torrent to calculate the number of peers per swarm $y_{t,r}$. Finally, the server informs all trackers $r \in \mathcal{R}$ about the target number of peers $y_{t,r}$ for every torrent $t \in \mathcal{T}(r)$.

In order to solve the *MMO* problem for a torrent, *CSM* uses the Minimum Excess-Deficit (*MED*) algorithm described in the following. The *MED* algorithm considers that swarms can be split arbitrarily, that is, peers in a torrent can be redirected between swarms on an individual basis, and can solve the *MMO* problem in $O(n\log n)$ time, as we show in Section VI. We note that if swarms could not be split arbitrarily then *MMO* would require solving the bin covering problem, which is NP-hard [13]: given a set of positive integers (the swarm sizes $x_{t,r}$) and bins of equal sizes (the threshold $\tilde{x}$) pack the integers into the maximum number of bins such that the sum of the integers in any bin is at least $\tilde{x}$.

Given a set of swarms, in decreasing order of size, the *MED* algorithm calculates the number of non-empty swarms of the *MMO* solution. The algorithm first calculates the excess number of peers that can potentially be moved from 'non-small' swarms to 'small' swarms. Starting from the initial number of non-small swarms it then evaluates whether it is feasible to increase the number of non-small swarms by either moving some excess peers from non-small swarms to small swarms or by merging the small swarms into non-small swarms. The algorithm terminates when it becomes infeasible to increment the number of non-small swarms. The pseudo-code of the *MED* algorithm is shown in Fig. 8.

The MED algorithm returns the optimal number of swarms m that should be non-empty after peer re-allocation. These $m \leq n$ swarms are then filled up, one-by-one, by moving peers from the other swarms. The reallocation of peers among swarms starts with the $n-m$ smallest swarms, which should be empty according to the algorithm, and continues with moving the excess peers from the biggest swarms (with $x_{t,r} > \tilde{x}$), if necessary. Fig 9 illustrates the operation of the *MED* algorithm on a simple example.

Proposition 1: The MED *algorithm solves the* MMO *optimization problem.*

Proof: We have to show that the *MED* algorithm (i) finds a solution with the minimum number of small swarms and that the solution found (ii) requires the fewest possible peers to be reallocated between swarms.

First, consider the number of small swarms. Let us call a peer allocation *valid* if the number of small swarms is minimal.

INPUT: Swarm sizes $x_1 \geq x_2 \geq \ldots \geq x_n$
OUTPUT: Optimal number m of non-empty swarms

1a. if $\min_i(x_i) \geq \tilde{x}$ return n
1b. if $\sum_{i=1}^{n} x_i \leq \tilde{x}$ return 1
2. Let $s = \inf\{i | x_i < \tilde{x}\}$
3. Let $E = \sum_{i=1}^{s-1}(x_i - \tilde{x})$ //excess number of peers
4. **for** $i = s$ to n
5. $D(i) = \sum_{j=s}^{i}(\tilde{x} - x_j)$ //deficit of peers in swarms
6. $M(i) = \sum_{j=i+1}^{n} x_j$ //peers to move from rest of swarms
7. if $M(i) + E < D(i)$ return $i-1$ //infeasible to have i swarms
8. if $M(i) + x_i < D(i)$ return $i-1$ //better to have $i-1$ swarms
9. **end for**
10. return n

Fig. 8: Pseudo-code of the *MED* algorithm used by *CSM* to solve the *MMO* optimization problem. For a set of swarms the algorithm returns the optimal number of non-empty swarms.

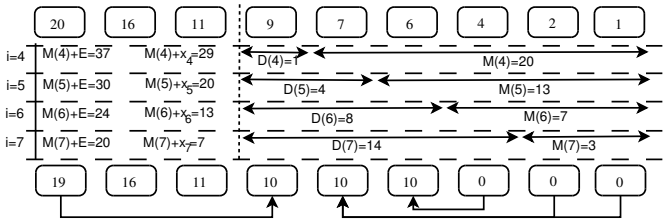


Fig. 9: Example with $n = 9$ swarms. $s = 4$ and $E = 17$. The rows show the iterations and the values of $D(i)$ and $M(i)$ as calculated by *MED*. The algorithm terminates for $i = 7$ because $M(i) + x_i < D$, the optimal number of swarms is $i - 1 = 6$, and a possible reallocation of the peers is shown in the last row.

By definition, if there are less than $\tilde{x}$ peers in the torrent then the solution to *MMO* is to have 1 small swarm. Otherwise, by allocating all peers to the first $i' = \max(1, s-1)$ swarms one can eliminate all small swarms at overhead $M(i') = \sum_{j=i'+1}^{n} x_j$. This is possible, because either swarm i' is not small or there are only small swarms but $\sum_{i=1}^{n} x_i \geq \tilde{x}$. Hence, *MED* starts from a valid allocation. In general, for an allocation with $i \geq i'$ swarms to be valid it must hold that $M(i) + E \geq D(i)$, that is, the number of excess peers plus the peers to be moved from swarms $j > i$ must be sufficient to make all swarms $s \leq j \leq i$ non-small, which requires $D(i) = \sum_{j=s}^{i} x_j$ peers to be moved. Observe that $M(i)$ is a strictly monotonically decreasing function of i and $D(i)$ is a strictly monotonically increasing function of i. The two functions are illustrated in Figure 10. Hence if an allocation with $\hat{i} \geq i'$ swarms is valid then all allocations with $i' \leq i \leq \hat{i}$ swarms are valid. *MED* starts from a valid allocation and by condition (7) it terminates when it encounters a non-valid allocation, hence the solution obtained by *MED* is valid.

Second, consider the minimum overhead to achieve a valid peer allocation. Any valid allocation with i non-empty swarms requires that either $M(i)$ or $D(i)$ peers are moved, whichever is larger. $M(i)$ peers must be moved away from swarms $j > i$ to ensure that there are no small swarms, and $D(i)$ peers must be moved to swarms $i' \leq j \leq i$ to ensure that these have at least $\tilde{x}$ peers. The minimal overhead is hence achieved by having $m = \arg\min_i \max\{D(i), M(i)\}$ non-empty swarms. Because of the strict monotonicity of $D(i)$ and $M(i)$ the overhead $\max\{D(i), M(i)\}$ has a global minimum. Let us define $i^* = \min\{i | D(i) > M(i)\}$. Then the minimal overhead is achieved for $m = i^*$ if $D(i^*) < M(i^*) + x_{i^*}$ and $m = i^* - 1$ otherwise, because $M(i-1) = M(i) + x_i$ as illustrated in Figure 10. *MED* will hence terminate as soon as the minimal overhead has been identified. ■

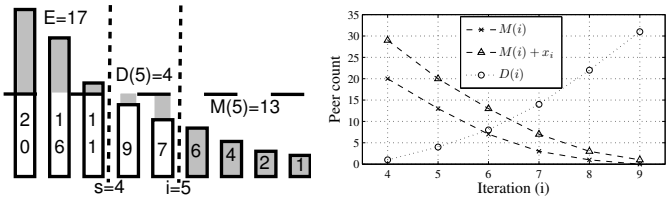


Fig. 10: Example with $n = 9$ swarms. $s = 4$ and $E = 17$. On the left, D and M as calculated by *MED* for $i = 5$. On the right, the evolution of D, M and $M + x_i$ as a function of i.

B. Distributed Swarm Management Protocol (DSM)

The *DSM* protocol is composed of two components. It relies on a distributed protocol to determine the order in which trackers should perform pairwise balancing. On a pairwise basis, trackers then exchange information about the number of active peers associated with each torrent they have in common (e.g., by performing a scrape), and determine the number of peers per swarm that the trackers should be responsible for.

In the following, we provide a description of these two components.

1) Distributed Negotiation: The distributed negotiation protocol assumes that each tracker r knows the set of torrents $\mathcal{T}(r, r') = \mathcal{T}(r) \cap \mathcal{T}(r')$ that it has in common with other trackers $r' \in \mathcal{R}$ for which the trackers' torrents are not disjoint (i.e., for which $|\mathcal{T}(r, r')| \neq 0$). Note that this information is available through the torrent metadata files, which are uploaded to the tracker when the torrents are registered with the tracker. In the following, we describe two alternatives that can be used to establish the order of pairwise balancings.

- *DSM-Max:* Tracker r invites for pairwise balancing the trackers r' for which the overlap in tracked torrents, $\mathcal{T}(r, r')$ is maximal among the trackers with which it has not yet performed the pairwise balancing. A tracker r' accepts the invitation if its overlap with tracker r is maximal. Otherwise, tracker r' asks tracker r to wait until their overlap becomes maximal for r' as well. *DSM-Max* was originally considered in [14].
- *DSM-Rand:* The second protocol is randomized. Tracker r invites for pairwise balancing any tracker r' with which it has not yet performed a pairwise balancing. A tracker r' accepts the invitation if it is not currently performing a pairwise balancing with another tracker.

Both protocols guarantee that all pairs of trackers with an overlap in torrents will perform a pairwise balancing once and only once during the execution of the protocol. However, as discussed in Section VII, the number of balancing rounds can be substantially lower using *DSM-Rand*.

INPUT: Swarm sizes $x_{t,r}, \forall t \in \mathcal{T}(r)$ and $x_{t,r'}, \forall t \in \mathcal{T}(r')$
OUTPUT: Updated swarm sizes $x_{t,r}$ and $x_{t,r'}$

1. **for** $\forall t \in \{\mathcal{T}(r,r') | x_{t,r} + x_{t,r'} < 2\tilde{x}\}$
1.1. $a \leftarrow \text{argmax}_r\, x_{t,r}$; $b \leftarrow \{r,r'\} \setminus \{a\}$
1.2. $x_{t,a} \leftarrow x_{t,r} + x_{t,r'}$; $x_{t,b} \leftarrow 0$
1. **end for** // merged small swarm into the less "small" swarm
2. **for** $\forall t \in \{\mathcal{T}(r,r') | x_{t,r} + x_{t,r'} \geq 2\tilde{x}, \min[x_{t,r}, x_{t,r'}] < \tilde{x}\}$
2.1. $a \leftarrow \text{argmax}_r x_{t,r}$; $b \leftarrow \{r,r'\} \setminus \{a\}$
2.2. $\mathtt{d} \leftarrow \tilde{x} - \min[x_{t,r}, x_{t,r'}]$
2.3. $x_{t,a} \leftarrow x_{t,a} - \mathtt{d}$; $x_{t,b} \leftarrow x_{t,b} + \mathtt{d}$
2. **end for** // re-balanced small swarms when possible
3. $a \leftarrow \text{argmax}_r(N_r - {}_t x_{t,r})$; $b \leftarrow \{r,r'\} \setminus \{a\}$
4. **while** $N_a > {}_t x_{t,a},\ \exists t | (x_{t,a} = 0, x_{t,b} \leq (N_a - {}_t x_{t,a}))$
4.1. $\mathtt{D} \leftarrow N_a - {}_t x_{t,a}$
4.2. $t \leftarrow \text{argmax}_t(x_{t,b} | x_{t,a} = 0, x_{t,b} < 2\tilde{x}, x_{t,b} \leq \mathtt{D})$
4.3. $x_{t,a} \leftarrow x_{t,b}$; $x_{t,a} \leftarrow 0$
4. **end while** // load adjusted using merged swarms
5. **while** $N_a > {}_t x_{t,a},\ \exists t | (x_{t,a} + x_{t,b} \geq 2\tilde{x}, x_{t,b} - x_{t,a} \geq 2)$
5.1. $\mathtt{D} \leftarrow N_a - {}_t x_{t,a}$
5.2. $t \leftarrow \text{argmax}_t(x_{t,b} - x_{t,a} | x_{t,a} + x_{t,b} \geq 2\tilde{x}, x_{t,b} - x_{t,a} \geq 2)$
5.3. $\mathtt{d} \leftarrow \min[\frac{x_{t,b} - x_{t,a}}{2}, \mathtt{D}]$
5.4. $x_{t,a} \leftarrow x_{t,a} + \mathtt{d}$; $x_{t,b} \leftarrow x_{t,b} - \mathtt{d}$
5. **end while** // load adjusted using split swarms
6. $\mathtt{D} \leftarrow N_a - {}_t x_{t,a}$ // adjust any remaining imbalance
7. find$\{t,t'\}$ *s.t.* $\min_{\{t,t' | x_{t,a} = x_{t',b} = 0\}} |\mathtt{D} - x_{t,b} + x_{t',a}|$
7.1. $x_{t,a} \leftarrow x_{t,b}$; $x_{t,b} \leftarrow 0$; $x_{t',b} \leftarrow x_{t',a}$; $x_{t',a} \leftarrow 0$

Fig. 11: The pairwise balancing algorithm for *MMO-LM*.

2) Pairwise Balancing: DSM can be used to obtain an approximate solution to the *MMO* and to the *MMO-LM* problems by using different pairwise balancing algorithms. In the case when *DSM* is used to solve the *MMO* problem, swarms associated with the two trackers can be merged according to the *MED* algorithm described in Section V-A.

For the case of the *MMO-LM* problem, we propose a three-step greedy algorithm to determine the peer allocation between the two trackers. Figure 11 shows the pseudo-code of the algorithm. First, torrents are treated independently and peers are tentatively shifted based only on information about each individual torrent. For all torrents t that require merging (i.e., for which $x_{t,r} + x_{t,r'} < 2\tilde{x}$), all peers are tentatively shifted to the tracker that already maintains information about more peers. (Lines 1-1.2.) For all torrents that should be re-balanced (i.e., for which $\min[x_{t,r}, x_{t,r'}] < \tilde{x}$ and $x_{t,r} + x_{t,r'} \geq 2\tilde{x}$), the minimum number of peers $\mathtt{d} = (\tilde{x} - \min[x_{t,r}, x_{t,r'}])$ needed to ensure that both trackers have at least $\tilde{x}$ peers are tentatively shifted to the tracker with fewer peers for that torrent. (Lines 2-2.3.)

Second, in order to achieve load conservation of the total number of peers N_r tracked by each tracker r, the peer responsibility of some torrents may have to be adjusted. Using a greedy approach, the tentative allocations are shifted towards the tracker that saw a decrease in peer responsibility (if any). (This tracker is identified in line 3.) To avoid increasing the number of peers associated with partial shifts, priority is given to allocations of the torrents that are being merged. (Hence, lines 4-4.3 are executed before lines 5-5.4.) Among the merged torrents, the algorithm selects the torrent that results in the largest load adjustment and that does not cause the imbalance in overall load to shift to the other tracker. (Lines 4-4.3.) Among the balanced torrents, the algorithm selects the torrent that results in the largest load adjustment and that does not cause the imbalance in overall load to shift to the other tracker. (Lines 5-5.4.) By sorting torrents based on their relative shift, this step can be completed in $O(|\mathcal{T}(r,r')| \log |\mathcal{T}(r,r')|)$ steps.[1]

Finally, if overall load conservation is not yet fully achieved, additional load adjustments can be achieved by swapping the peer responsibilities of the pair of torrents that (if the responsibilities of those torrents were swapped) would result in the load split closest to achieving perfect load conservation of N_r, with ties broken in favor of choices that minimize the total shift of peers. (Lines 6-7.1.) Of course, considering all possible combinations can scale as $O(|\mathcal{T}(r,r')|^2)$. However, by noticing that only the torrents with the smallest shift of peers for each load shift are candidate solutions, many combinations can be pruned. By sorting the torrents appropriately, our current implementation achieves $O(|\mathcal{T}(r,r')| \log |\mathcal{T}(r,r')|)$ whenever $\tilde{x}$ is finite.

C. Protocol Implementation

The proposed *CSM* and *DSM* protocols could be implemented with minor extensions to the BitTorrent protocol. The only new protocol message required is a *tracker_redirect* message that can be used by a tracker to signal to a peer that it should contact an alternative tracker for the torrent. The message is used by a tracker r for a torrent t for which $x_{t,r}$ decreases due to the execution of *CSM* or *DSM*. Peers that receive the *tracker_redirect* message should contact another tracker they know about from the tracker file. Once two swarms are merged, one of the trackers becomes responsible for the merged swarms and all arriving peers will register with the responsible tracker. For this reason neither *CSM* nor *DSM* are equivalent to trackers merging their peer lists periodically: merging peer lists would lead to trackers announcing already departed peers unless merging is done very frequently. *CSM* and *DSM* avoid this problem by delegating swarms between trackers dynamically.

VI. Protocol Analysis and Performance

A. Protocol Overhead

The communication overhead of *CSM* is dominated by the collection of swarm popularity information from the trackers (4 bytes per swarm independent of the actual number of peers $x_{t,r}$ in the swarm), by the notifications sent to the trackers (4 bytes per swarm) and by the redirection messages sent by the trackers to the peers. The amount of redirection messages sent by tracker r is bounded by ${}_{t \in \mathcal{T}(r)} x_{t,r}$

The communication overhead of *DSM* is dominated by the exchange of torrent popularity information between the trackers and by the redirection messages sent to the peers. Distributed negotiation involves one tracker scrape before every pairwise balancing, and the corresponding exchange of the results of the balancing. The amount of data exchanged

[1] This sorting can be done just before lines 4 and 5 of the algorithm, respectively. With a sorted list the sequence of argmax operations can easily be translated into a single "linear" for (or while) loop.

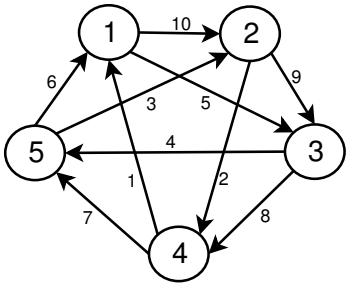


Fig. 12: Complete graph with $|\mathcal{R}| = 5$ vertices on which *DSM-Max* requires $|\mathcal{R}|(|\mathcal{R}|-1)/2 = 10$ balancing rounds to finish. The arrows show the execution order of *DSM-Max* starting from vertex 1. *DSM-Rand* requires $2\lfloor(|\mathcal{R}|+1)/2)\rfloor - 1 = 5$ balancing rounds only.

between trackers r and r' is hence $O(|\mathcal{T}(r,r')|)$. [2] The amount of redirection messages sent by tracker r is proportional to the number of peers shifted between swarms tracked by r and r', and is bounded by $\sum_{t\in\mathcal{T}(r,r')} x_{t,r}$. Once two swarms are merged, one of the trackers becomes responsible for the merged swarms and all arriving peers will register with the responsible tracker, hence there will be no overhead related to these swarms when the next pairwise balancing is executed between the two trackers (unless the swarm increases significantly, and has to be split into two).

B. Protocol Complexity

We now turn to the complexity of the *MED* algorithm, and that of the distributed negotiation used in *DSM*. We first show that *MED* solves *MMO* in linearithmic ($O(n\log n)$) time.

Proposition 2: The computational complexity of the MED *algorithm is* $O(n\log n)$.

Proof: The algorithm first sorts the swarms in decreasing order of size, which has complexity $O(n\log n)$. Once sorted, the algorithm terminates after at most n iterations. The summations used in each iteration can be obtained using a single operation from the value of the corresponding sum as used in the previous iteration. Hence, the algorithm has computational complexity $O(n\log n)$. ■

We already provided a complexity analysis of pairwise balancing in Section V-B2, so here we focus on distributed negotiation. We analyze the complexity of distributed negotiation using the assumption that pairwise balancing can be performed between distinct pairs of trackers simultaneously, and show that the complexity of *DSM-Max* is in the worst case almost a factor $|\mathcal{R}|/2$ higher than that of *DSM-Rand*.

Proposition 3: The worst case performance of *DSM-Max* on a graph with $|E|$ edges is $O(|E|)$ balancing rounds. Hence on a complete graph it is $O(|\mathcal{R}|^2)$ balancing rounds.

Proof: We first show that the worst case performance is $O(|E|)$ balancing rounds. Consider a tracker graph (as defined in Section II-B), in which the maximum overlaps between neighboring trackers, when ordered, form an Eulerian path. By definition, the length of the Eulerian path is $|E|$, and there will be only one pairwise balancing executed at once. Because the number of edges in an undirected complete graph is $|\mathcal{R}|(|\mathcal{R}|-1)/2$, the number of balancing rounds in the worst case is $O(|\mathcal{R}|^2)$. ■

For example, in the complete graph with 5 vertices shown in Fig. 12 the maximum overlaps form an Eulerian path starting at vertex 1 and following the overlaps indicated on the edges in decreasing order. *DSM-Max* will start by performing a pairwise balancing between vertices 1 and 2, followed by 2 and 3, etc. All remaining nodes wait until the nodes with the highest overlaps perform their pairwise balancings.

DSM-Rand requires significantly less balancing rounds, as shown by the following result.

Proposition 4: The worst case performance of *DSM-Rand* on a graph with $|\mathcal{R}|$ vertices is $2\lfloor(|\mathcal{R}|+1)/2)\rfloor - 1$ balancing rounds, that is, $O(|\mathcal{R}|)$.

Proof: Let us consider first a complete graph with an even number of vertices $|\mathcal{R}|$. The number of vertices that can be involved in a pairwise balancing at any moment is exactly $|\mathcal{R}|$. Each vertex has to perform $(|\mathcal{R}|-1)$ pairwaise balancings, hence the number of balancing rounds needed is $|\mathcal{R}|-1$.

Consider now a complete graph with an odd number of vertices $|\mathcal{R}|$. The number of vertices that can be involved in a pairwise balancing at any moment is $|\mathcal{R}|-1$. The total number of pairwise balancings is $|\mathcal{R}|(|\mathcal{R}|-1)/2$, but in any balancing round only $(|\mathcal{R}|-1)/2$ pairwise balancings can be performed, hence the number of balancing rounds is $|\mathcal{R}|$.

For a non-complete graph with $|\mathcal{R}|$ vertices the number of balancing rounds cannot be more than that for a complete graph, hence the result. ■

Consequently, on a complete graph the number of balancing rounds needed using *DSM-Max* can be up to $|\mathcal{R}|/2$ times higher than using *DSM-Rand*.

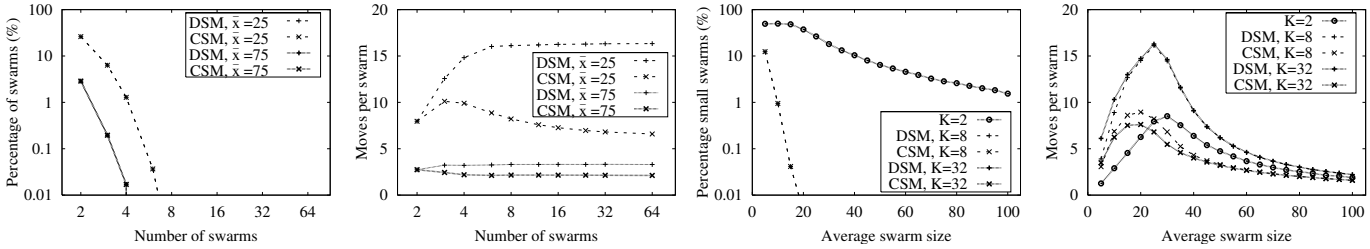
C. Simulation-based Performance Evaluation

In the following we show simulation results that compare the number of balancing rounds and the effectiveness of *DSM* to that of the solution of the centralized *CSM* protocol.

Figures 13(a) and 13(c) show the percentage of small swarms after applying the centralized and the distributed protocols as a function of the number of swarms K and as a function of the average swarm size $\bar{x}$, respectively. Both figures show results for the case when swarm sizes are uniformly distributed, and the threshold for a small swarm is set at $\tilde{x} = 50$. The distributed protocol used is *DSM-Rand* with the *MMO*-based pairwise balancing algorithm. Figures 13(b) and 13(d) show the average number of peers that had to be moved per swarm to merge the swarms for the same scenarios. Figure 14 shows the corresponding results for the case when the original swarm sizes are Zipf distributed with exponent 0.9 instead of uniformly distributed.

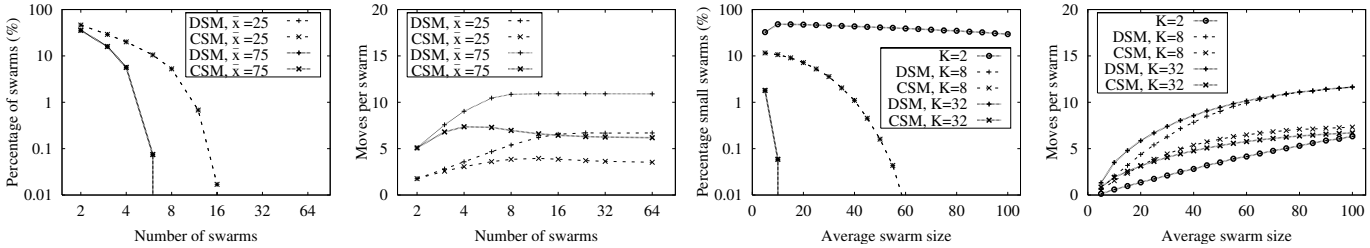
When varying the number of swarms K, we show results for $\bar{x} = 25$ and $\bar{x} = 75$. These values correspond to average swarm sizes that are smaller and bigger than the small swarm threshold $\tilde{x} = 50$, respectively. When varying the swarm size, we show results for $K = 2$, $K = 8$, and $K = 32$. We note that the

[2] Note that the two trackers exchange the number of seeds and leechers (e.g., 4 bytes per torrent), so that the actual number of peers $x_{t,r}$ and $x_{t,r'}$ does not influence the amount of data exchanged.



(a) Small swarms (b) Moves (c) Small swarms (d) Moves

Fig. 13: Percentage of small swarms after applying *DSM* and *CSM*, as well as the average number of moves per swarm needed by these protocols. Swarm sizes are uniformly distributed.



(a) Small swarms (b) Moves (c) Small swarms (d) Moves

Fig. 14: Percentage of small swarms after applying *DSM* and *CSM*, as well as the average number of moves per swarm needed by these protocols. Swarm sizes are Zipf distributed.

operation decisions of the two protocols are identical for the case when $K = 2$, and the curves for *DSM* and *CSM* therefore overlap. The cases $K = 8$ and $K = 32$ are used to illustrate the relative difference when there are more swarms in the system.

We note that *CSM* provides a bound on the best that the *DSM* protocol can perform. While the distributed *DSM-Rand* protocol consistently achieves the same number of small swarms as the centralized *CSM* protocol (i.e., the *MED* algorithm), it typically requires slightly more peers to be moved between swarms. At the same time, the final swarm sizes, including the total number of non-empty swarms, often differ for the two protocols.

Comparing Figures 13 and 14 we note that the protocols are able to achieve a smaller number of small swarms for the case of uniformly distributed original swarm sizes. However, for average swarm size around 30 the efficiency in reducing the number of small swarms comes at the cost of more peers being moved between trackers, as shown by the peak in terms of the number of moves in Figure 13(d). Overall, we find that *DSM* can achieve much of the benefits of *CSM*.

VII. Trace-based Performance Evaluation

In order to illustrate the potential benefits of swarm management we used the *CSM* and the *DSM* protocols on the measurement dataset described in Section III. In the following we present results on the swarm merging efficiency and provide an estimate of the achievable throughput improvements.

A. CSM *and* DSM *protocol performance*

To analyze the performance benefits of dynamic swarm management, we used *CSM* and *DSM* to re-allocate the peers between swarms belonging to the same torrent in the dataset obtained on October 10, 2008. In this snapshot there are 550 trackers that have to participate in swarm management (the rest do not have overlap in torrents). The average overlap between these 550 trackers is $1,800$ torrents.

In total, the dataset contains $765,795$ small multi-tracked swarms (and some non-small ones) if using a threshold of $\tilde{x} = 50$. Using the centralized *CSM* protocol $2,875,363$ peers were moved between trackers, and $398,751$ of the multi-tracked swarms became empty (resulting in a total of $367,044$ small swarms). On average less than 3.8 peers per swarm had to be moved.

Using *DSM* with the *MMO*-based pairwise algorithm, we could achieve the same reduction in terms of the number of small swarms, but requiring slightly more peers to be moved, a total of $2,891,392$ peers. Using *DSM* with the *MMO-LM*-based pairwise algorithm, which provides load balancing, reduced the number of small swarms by $393,002$ only, while requiring $4,080,624$ peers to be moved. This may seem like a high price to pay for load balancing, but for independently run trackers *MMO-LM* is the easiest way to ensure that some trackers do not end up with an unfair portion of the load after merges. The best solution may hence depend on the relationship between the trackers (and who operates them). Since BitTorrent trackers are mostly operated by independent entities, in the following we focus on *DSM* with the *MMO-LM*-based pairwise balancing algorithm.

Figure 15 shows the normalized CoV for all torrents using *DSM-Rand*. Comparing Figure 15 with Figure 3 we note that *DSM* significantly reduces the number of torrents operating in the lower left corner, i.e., the number of small torrents in which peers are spread roughly uniformly among the swarms. It is interesting to note that there are almost no small torrents left with a normalized CoV below 0.5. To understand why,

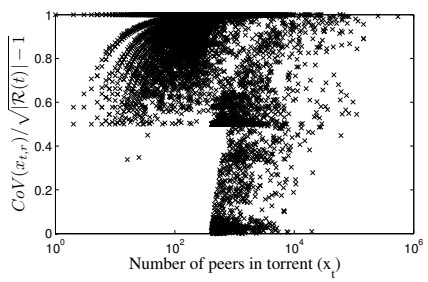


Fig. 15: The normalized coefficient of variation after applying *DSM-Rand* with $\tilde{x} = 200$.

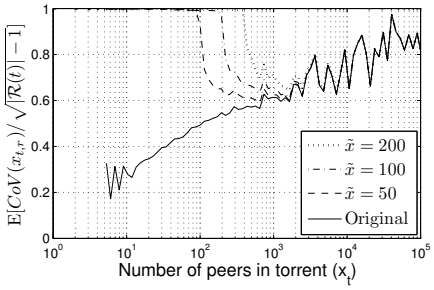


Fig. 16: Average coefficient of variation as a function of the number of peers for *DSM-Max*.

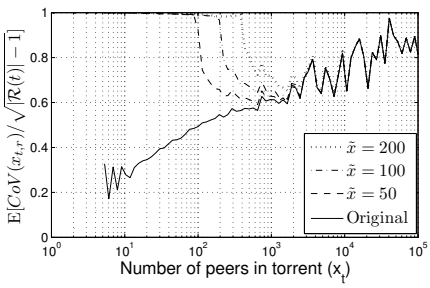


Fig. 17: Average coefficient of variation as a function of the number of peers for *DSM-Rand*.

consider a small torrent that has $\mathcal{R}(t)$ equal sized swarms initially, thus its normalized CoV is 0. After one execution of *DSM* the number of small swarms is at most $\mathcal{R}(t)/2$, and the sizes of the non-empty swarms double. Thus the normalized CoV is $1/\sqrt{\mathcal{R}(t)-1}$, which is at least 0.5 for $\mathcal{R}(t) \leq 5$, and from Figure 2 we know that not many torrents have more than 5 swarms. It is also conspicouos that *DSM* merges quite some torrents that have significantly more than $\tilde{x} = 200$ peers and an original normalized CoV above 0.5. While there still are some torrents that do operate in the lower left corner, Figure 15 suggests that their number is very small.

Figures 16 and 17 show the average normalized CoV as a function of the number of peers for October 10, 2008 for *DSM-Max* and *DSM-Rand*, respectively. Both figures show results for both the original peer allocation, as well as for *DSM* using three different threshold values (i.e., $\tilde{x} = 50, 100, 200$) for pairwise balancing. We note that *DSM* pushes the CoV for torrents smaller than the threshold values (and even a bit beyond) to roughly one. As previously discussed, this is exactly what we want.

In total, both *DSM-Max* and *DSM-Rand* have to perform 3,535 pairwise balancings; i.e., slightly more than six per tracker on average. Because a few trackers have to perform pairwise balancing with a large number of trackers, both *DSM-Max* and *DSM-Rand* require 505 balancing rounds to perform all pairwise balancings if one considers all pairs of overlapping trackers. Nevertheless, in order to perform 50% of the pairwise balancings *DSM-Max* requires 327 rounds, while *DSM-Rand* requires 30 rounds only. Given the higher level of parallelism allowed by *DSM-Rand*, the question is of course whether the random execution order of pairwise balancings affects the balancing performance. Referring back to Figures 16 and 17, we note that the difference between the performance of *DSM-Max* and *DSM-Rand* is negligible. Since *DSM-Rand* is computationally less intensive, we will in the following primarily present results for *DSM-Rand*.

The large number of trackers with small overlaps, as observed in Figure 5, suggests that the number of rounds can be significantly reduced if the *DSM* protocol did only do pairwise load balancing between trackers with at least some threshold number of torrents in common. The question is then how such thresholding would affect the efficiency of *DSM*. To understand this trade-off, we considered a threshold policy for *DSM*. Figure 18 shows results for the case when pairwise balancing is only performed if $|\mathcal{T}(r,r')| > 50$, 100 or 200. We observe a very small performance penalty using these thresholds, while the savings in terms of the number of pairwise balancings are significant. If one considers overlaps of more than 50 torrents only (i.e., tracker pairs r, r' such that $|\mathcal{T}(r,r')| > 50$), then the number of pairwise balancings is 353, of which *DSM-Max* performs 50% in 68 rounds, and *DSM-Rand* in 27 rounds. The respective numbers for $|\mathcal{T}(r,r')| > 100$ are 233, 36 and 22, and for $|\mathcal{T}(r,r')| > 200$ they are 142, 27 and 16. These results show that both *DSM-Rand* and the threshold-based modifications to the *DSM-Max* protocol can reduce the execution time and the overhead of *DSM*, without significant impact on the performance gain.

When discussing the protocol convergence, it is also important to note that *DSM* requires fewer re-allocations of peers among trackers when the protocol is re-applied on the same set of torrents day after day than when first applied on the original non-balanced dataset. To capture this effect we used daily measurements of the swarm sizes for eight consecutive days. For simplicity, the total torrent size (of a set of swarms) is estimated as the sum across all "measured" swarms for that torrent, and the peers associated with this torrent are split across the swarms such that the estimated individual swarm sizes are proportional to the individual swarm sizes on the preceding day (as determined by *DSM* that day).

Figure 19 shows the average number of swarm changes per peer that is in the system at the time the protocol is applied, as a function of the time for both *DSM-Max* and *DSM-Rand*. While *DSM-Rand* appears to take a bit longer to converge than *DSM-Max*, it is interesting to note that both versions appear to converge relatively quickly to roughly the same value. Furthermore, as expected, the later iterations require fewer changes per peer and the average number of swarm changes per peer is relatively low suggesting that the overhead is low. Finally, and interestingly, we found that *DSM-Rand* performs very similar to *DSM-Max*. Due to its simpler implementation and the fact (as noted in Section V-C) that it typically requires much less balancing rounds, *DSM-Rand* appears to be the preferred tracker based protocol.

B. Estimated throughput improvements

Finally, we use the throughput estimates of different sized swarms presented in Section III to estimate the potential performance gains that small torrents can obtain using our protocols. We rely on the throughput estimates from 7pm GMT shown in Figure 6 to estimate the speedup achieved by the

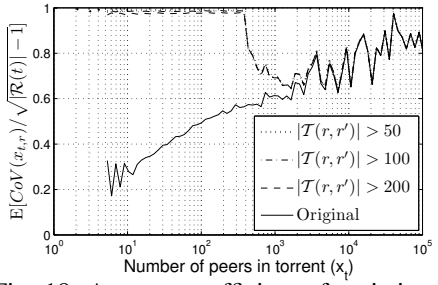


Fig. 18: Average coefficient of variation vs. number of peers for *DSM-Rand* for various lower limits of tracker overlap.

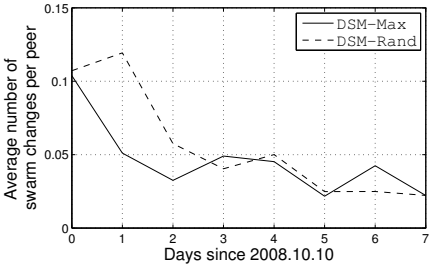


Fig. 19: Time-line simulations using measurement data for each day.

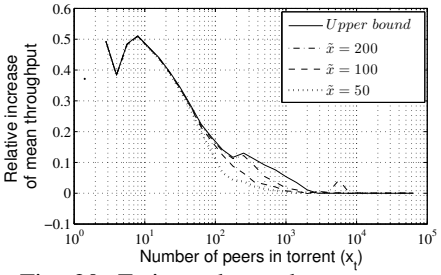


Fig. 20: Estimated speedup vs. torrent size after applying *DSM-Rand*.

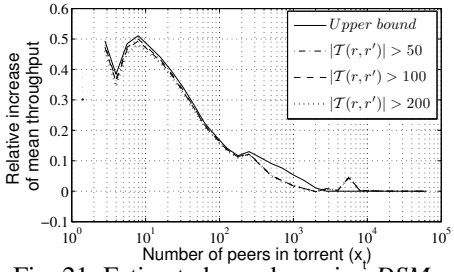


Fig. 21: Estimated speedup using *DSM-Rand* with various thresholds of tracker overlap, $\tilde{x}=200$.

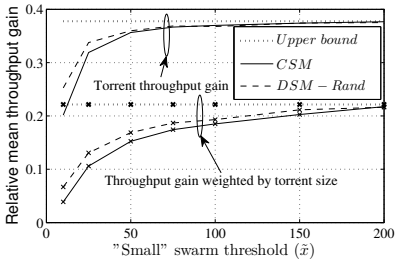


Fig. 22: Estimated speedup for torrents smaller than 300 peers vs. small swarm threshold $\tilde{x}$.

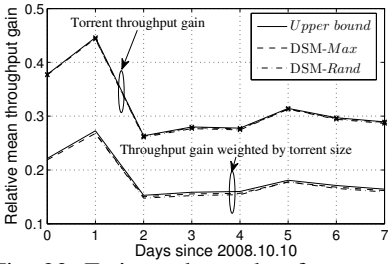


Fig. 23: Estimated speedup for torrents smaller than 300 peers vs. time.

DSM protocol (for swarms below 1,000 peers the estimated throughputs in Figures 6 and 7 coincide). As a benchmark for our protocols we show in the figures an *upper bound*, which was obtained by assuming that all peers of a torrent are in the same swarm. This would be the case, for example, if all peers, upon arrival, registered with all trackers. Such a solution requires trackers to maintain much more state information and have much higher communication overhead than *DSM*, but it enables us to assess the throughput improvement obtained by *DSM*. Alternatively, peers could register with the biggest swarm. This solution could, however, lead to the overload of a tracker if some form of load balancing is not implemented.

Figure 20 shows the relative throughput improvement for leechers as a function of the torrent size for *DSM-Rand* for various values of the threshold $\tilde{x}$. The results show a good match with our objectives: the smallest torrents experience throughput gains up to 50% on average, while torrents above approximately 1000 peers are not affected by our protocol. Figure 21 shows the relative throughput improvement for leechers for *DSM-Rand* for various threshold values on the minimum overlap between trackers and $\tilde{x}=200$. The results support our previous observation about the minor impact of the threshold policy; most of the benefits of *DSM* can be achieved by having only the trackers with the biggest overlaps use *DSM*.

The throughput improvements achieved using *DSM* are close to the upper bound. With *DSM* the throughput is typically increased by 40% or more on average when torrents have less than ten peers, and by 10% or more on average when torrents have less than 200 peers. In general, these results show a good match with our objectives: the smallest torrents experience throughput gains up to 60% on average, while torrents above approximately 200 peers are not affected by our protocols.

Figure 22 shows the relative estimated throughput improvement for leechers in torrents smaller than 300 peers as a function of the small swarm threshold $\tilde{x}$. The throughput gain is only sensitive to the threshold below $\tilde{x}\approx 75$, above that the gain is very close the upper bound, both the average gain per torrent and the gain weighted by the torrent size. Figure 23 shows the relative estimated throughput improvement for leechers in torrents smaller than 300 peers over a week. The throughput gains are rather steady, and show that dynamic swarm management could improve peer throughput significantly. For example, the average torrent with less than 300 peers sees an increase in throughput by 30% on average (and the average peer in such a torrent sees a throughput increase by 15%).

VIII. Related Work

Much research has been done to understand BitTorrent and BitTorrent-like systems. The effectiveness of BitTorrent's tit-for-tat and rarest-first mechanisms was considered in [15], [16], the potential of network coding for content distribution was investigated in [17], and BitTorrent-like systems have also been considered for streaming [18]. Most of these efforts have focused on understanding the performance of single-torrent systems. Other works have analyzed the general characteristics of BitTorrent traffic, and the impact of BitTorrent usage on the amount of inter-ISP traffic (e.g., [9], [10]).

There are a few recent works that consider multi-torrent environments [1], [19], [20], [21], [22]. Neglia *et al.* [1] evaluated the benefits of multiple trackers in terms of improved tracker availability based on measurements. They found that multiple trackers significantly improve the availability, and observed that multiple trackers can reduce the connectivity of the overlay. Guo *et al.* [19] provided measurement results showing that more than 85% of torrent users simultaneously

participate in multiple torrents (of different files). The authors also used measurements and analytical models to illustrate that the "life-time" of torrents can be extended if a node that acts as a downloader in one torrent also acts as a seed in another torrent. Yang *et al.* [20] proposed rate-based incentives that motivate users to act as seeds for other torrents than they currently are downloading. Menasche *et al*. [4] showed that "bundling" multiple files into a larger file may increase the aggregate number of parallel downloaders, extend the lifetime of torrents, and reduce the download times (even for clients that only are interested in downloading part of the larger bundle). Peterson *et al.* [21] proposed a central coordinator that would allocate peers' upload capacities between different torrents in order to optimize download performance.

Our work is complimentary to the above works in several aspects. In contrast to [4], [20], [21], [22] we do not require peers to download more than they are interested in. Furthermore, our focus is on achieving good performance while maintaining the high availability of a replicated tracker based infrastructure pointed out in [1]. Rather than increasing seed capacity through cooperation among peers downloading different contents, our work focuses on how multiple swarms of the *same* content can be merged to improve the performance of small swarms. To the best of our knowledge, no other work (except our own [14]) has considered the performance benefits from adaptively merging peers for the same torrent using a tracker-based protocol. In this paper, we provide a more general and detailed description of the *DSM* protocol first described in [14], we describe modifications that significantly reduce the communication overhead between trackers, and provide a more detailed performance evaluation than was presented in our original paper [14].

IX. Conclusion

Based on an extensive measurement study we observed that there exist many moderately popular torrents with several small swarms that could significantly benefit from reallocating peers among trackers and/or modifying the peer behavior. We proposed an approach for reallocating peers among trackers based on splitting swarms on-demand and merging small swarms periodically. We formulated two versions of the problem of merging small swarms as optimization problems, and proposed a centralized and a decentralized protocol to solve the optimization problems. Both protocols have low overhead and computational complexity, furthermore the decentralized protocol can be introduced incrementally in existing BitTorrent systems.

Our measurement-based protocol evaluation suggests that dynamic swarm management could lead to a significant performance improvement in terms of peer throughput at a very low overhead; up to 40% for small torrents. While the proposed protocols may not be the only way to improve the throughput of small swarms, the potential benefits of other solutions would be similar to those identified in this paper.

Apart from improving peer throughput, swarm management could also facilitate locality-aware peer selection as it augments the set of peers to exchange data with. Tracker-based swarm management protocols could potentially also be used to dynamically bundle torrents, with the aim of increasing content availability. We leave these applications of swarm management to be subject of future work. We presented the protocols in the context of BitTorrent, and our work was motivated by the potential performance benefits in these systems. Nevertheless, the protocols are in general applicable to content delivery systems based on redundant trackers, and can be used to achieve high availability and good system performance with very little overhead.

References

[1] G. Neglia, G. Reina, H. Zhang, D. Towsley, A. Venkataramani, and J. Danaher, "Availability in BitTorrent Systems," in *Proc. IEEE INFOCOM*, May 2007.

[2] BitTorrent Extension Protocols (BEP-0011 and BEP-0012), Nov. 2010. [Online]. Available: http://www.bittorrent.org

[3] X. Yang and G. de Veciana, "Service Capacity of Peer-to-Peer Networks," in *Proc. IEEE INFOCOM*, Mar. 2004.

[4] D. Menasche, A. Rocha, B. Li, D. Towsley, and A. Venkataramani, "Content Availability and Bundling in Swarming Systems," in *Proc. ACM CoNEXT*, Dec. 2009.

[5] M. Arlitt and C. Williamson, "Internet Web Servers: Workload Characterization and Performance Implications," *IEEE/ACM Trans. on Networking*, vol. 5, no. 5, pp. 631–645, Oct. 1997.

[6] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web Caching and Zipf-like Distributions: Evidence and Implications," in *Proc. IEEE INFOCOM*, March 1999.

[7] M. Cha, H. Kwak, P. Rodriguez, Y. Ahn, and S. Moon, "I Tube, You Tube, Everybody Tubes: Analyzing the World's Largest User Generated Content Video System," in *Proc. ACM IMC*, Oct. 2007.

[8] P. Gill, M. Arlitt, Z. Li, and A. Mahanti, "YouTube Traffic Characterization: A View from the Edge," in *Proc. ACM IMC*, Oct. 2007.

[9] G. Dán and N. Carlsson, "Power-law revisited: A large scale measurement study of P2P content popularity," in *Proc. International Workshop on Peer-to-peer Systems (IPTPS)*, Apr. 2010.

[10] J. S. Otto, M. A. Sanchez, D. R. Choffnes, F. E. Bustamante, and G. Siganos, "On blind mice and the elephant: Understanding the network impact of a large distributed system," in *Proc. of ACM SIGCOMM 2011*, Aug. 2011.

[11] A. Barabási and Z. Oltvai, "Network biology: understanding the cell's functional organization," *Nat. Rev. Genet.*, vol. 5, pp. 101–113, 2004.

[12] B. Bollobás and O. Riordan, "The diameter of a scale-free random graph," *Combinatorica*, vol. 24, no. 1, pp. 5–34, 2004.

[13] S. Assmann, D. Johnson, D. Kleitman, and J. Leung, "On a dual version of the one-dimensional bin packing problem," *J. of Algorithms*, vol. 5, pp. 502–525, 1984.

[14] G. Dán and N. Carlsson, "Dynamic Swarm Management for Improved BitTorrent Performance," in *Proc. International Workshop on Peer-to-peer Systems (IPTPS)*, Apr. 2009.

[15] A. R. Bharambe, C. Herley, and V. N. Padmanabhan, "Analyzing and Improving a BitTorrent Network's Performance Mechanisms," in *Proc. IEEE INFOCOM*, Apr. 2006.

[16] A. Legout, G. Urvoy-Keller, and P. Michiardi, "Rarest First and Choke Algorithms Are Enough," in *Proc. ACM IMC*, Oct. 2006.

[17] C. Gkantsidis and P. R. Rodriguez, "Network Coding for Large Scale Content Distribution," in *Proc. IEEE INFOCOM*, Mar. 2005.

[18] X. Zhang, J. Liu, B. Li, and T.-S. P. Yum, "CoolStreaming/DONet: A Datadriven Overlay Network for Peer-to-Peer Live Media Streaming," in *Proc. IEEE INFOCOM*, Mar. 2005.

[19] L. Guo, S. Chen, Z. Xiao, E. Tan, X. Ding, and X. Zhang, "Measurement, Analysis, and Modeling of BitTorrent-like Systems," in *Proc. ACM IMC*, Oct. 2005.

[20] Y. Yang, A. L. H. Chow, and L. Golubchik, "Multi-Torrent: A Performance Study," in *Proc. IEEE MASCOTS*, Sept. 2008.

[21] R. S. Peterson and E. G. Sirer, "AntFarm: Efficient Content Distribution with Managed Swarms," in *Proc. NSDI*, May 2009.

[22] N. Carlsson, D. L. Eager, and A. Mahanti, "Using Torrent Inflation to Efficiently Serve the Long Tail in Peer-assisted Content Delivery Systems," in *Proc. IFIP/TC6 Networking*, May 2010.

György Dán received the M.Sc. degree in computer engineering from the Budapest University of Technology and Economics, Hungary in 1999 and the M.Sc. degree in business administration from the Corvinus University of Budapest, Hungary in 2003. He worked as a consultant in the field of access networks, streaming media and videoconferencing 1999-2001. He received his Ph.D. in Telecommunications in 2006 from KTH, Royal Institute of Technology, Stockholm, Sweden, where he currently works as an assistant professor. He was a visiting researcher at the Swedish Institute of Computer Science in 2008. His research interests include the design and analysis of distributed and peer-to-peer systems.

Niklas Carlsson is an Assistant Professor at Linköping University, Sweden. He received his M.Sc. degree in engineering physics from Umeå University, Sweden, and his Ph.D. in computer science from the University of Saskatchewan, Canada. He has also worked as a Postdoctoral Fellow at the University of Saskatchewan, Canada, and as a Research Associate at the University of Calgary, Canada. His research interests are in the areas of design, modeling, characterization, and performance evaluation of distributed systems and networks.