J Braz Comput Soc (2013) 19:475–491
DOI 10.1007/s13173-013-0112-z

ORIGINAL PAPER

BitTorrent traffic from a caching perspective

Lesandro Ponciano · Nazareno Andrade · Francisco Brasileiro

Received: 13 July 2012 / Accepted: 15 May 2013 / Published online: 31 May 2013

Abstract BitTorrent currently contributes a significant amount of inter-ISP traffic. This has motivated research and development to explore caching and locality-aware neighbor selection mechanisms for costly traffic reduction. Recent researches have analyzed the possible effects of caching BitTorrent traffic and have provided preliminary results on its cacheability. However, little is known about the specifics of caching design that affect cache effectiveness and operation, such as replacement policy and cache size. This study addresses this gap with a comprehensive analysis of BitTorrent caching based on traces of user behavior in four popular BitTorrent sites. Our trace-driven simulation results show differences in BitTorrent traffic caching compared to that of the Web and other peer-to-peer applications. Differently from Web and other peer-to-peer caching, larger caches are necessary to achieve similar caching effectiveness in BitTorrent traffic. Furthermore, in BitTorrent caching, the LRU replacement policy that takes the temporal locality into account shows the best performance. We also use a locality-aware neighbor selection mechanism as a baseline to evaluate the LRU caching effectiveness. We find that LRU caching can provide greater traffic reduction than locality-aware neighbor selection in several scenarios of cache size and number of ISP clients.

L. Ponciano · N. Andrade · F. Brasileiro (✉)
Departamento de Sistemas e Computação, Universidade Federal de Campina Grande, Av. Aprígio Veloso, s/n-Bloco CO, Campina Grande, PB 58429-900, Brazil
e-mail: fubica@dsc.ufcg.edu.br

L. Ponciano
e-mail: lesandrop@lsd.ufcg.edu.br

N. Andrade
e-mail: nazareno@dsc.ufcg.edu.br

Keywords Caching · Network management · Network protocols · Peer-to-peer systems · Traffic analysis

1 Introduction

BitTorrent presently generates a substantial amount of the Internet traffic [22,34,48], and has therefore significant cost implications for infrastructure operators. Many thousands or even millions of users daily rely on BitTorrent to distribute high volumes of content, generating sizeable amounts of transit traffic for Internet Service Providers (ISPs) [19,29]. This observation has motivated research and development to reduce inter-ISP BitTorrent traffic [13,48,55].

The two main approaches to reduce this traffic are caching [2,37,38,48] and locality-aware neighbor selection mechanisms [13,26,30]. Through caching, an ISP stores local copies of objects[1] accessed by its users, and reduces transit traffic by serving future object requests using local copies. Locality-aware neighbor selection mechanisms affect how users of the peer-to-peer network choose peers for data exchange. These methods aim at concentrating traffic among users in a same ISP, keeping data transfers at the edges of the network.

We are not aware of any commercial system that has adopted locality-aware neighbor selection mechanisms. In contrast, there exist some commercial initiatives to provide caching [37,38]. We believe that the simplicity of implementing caching is what explains this state of affairs. Regarding the research community, several studies have addressed the effectiveness of locality-aware neighbor selection mechanisms in reducing BitTorrent inter-ISP traffic

[1] In the context of this paper, an object is a generic term that can represent a file or parts of a file.

[11,30,31,47,43], while little is known about the effectiveness of different designs for caching BitTorrent traffic in terms of number of cache clients, cache size, and replacement policies. Previous work point only at the potential of reducing BitTorrent transit traffic using caches [2,28], but has not addressed the factors affecting cache effectiveness. In particular, existing analyses do not provide evidence that can be used to dimension a cache, to select which object replacement policy should be used, or to account how caching effectiveness varies with the cache size. This study contributes to fill in this gap, analyzing BitTorrent caching design considering finite cache size, and analyzing a comprehensive set of object replacement policies which come from Web and P2P literature and leverage different traffic characteristics. To put cache performance in perspective, the study also uses a locality-aware neighbor selection mechanism as baseline to evaluate the caching effectiveness.

BitTorrent traffic has several peculiarities that can affect the effectiveness of cache design when compared to other traffic where cache has been used or thoroughly studied, such as Web traffic, and other peer-to-peer traffic (e.g., FastTrack [51], Ka-zaa [44], Gnutella [24]). The concentration of references in BitTorrent traffic is modeled by a log-normal distribution [5], while it is approximated by a Zipf distribution in Web [12], and by a Mandelbrot-Zipf distribution in Gnutella [24]. Object size in BitTorrent is typically larger than in the Web and other peer-to-peer traffic [9]. Finally, the temporal locality of access in the Web traffic is not strong [6,7,32,52], while in BitTorrent the popularity of files decrease rapidly with time [5,21,40]. In this study, we find that some of these characteristics, together with new dimensions that we investigate, lead to different behavior in BitTorrent caching when compared to Web and other peer-to-peer caching mechanisms.

Our method is based on traces of user behavior in four popular BitTorrent directories and trace-driven simulation. This simulation also considers a set of object replacement policies that includes referential policies that leverage traffic characteristics relevant to caching—least-frequently-used (LFU), size-based (SIZE), and least-recently-used (LRU)—, and two policies proposed for other peer-to-peer traffic: least sent bytes (LSB) [51], and proportional partial caching (P2P) [24]. Understanding the performance of such a set of policies at the same time unveils relevant properties of BitTorrent traffic for caching and provides a referential frame for the development of new policies for future work.

Our main results are:

- We find that LRU leads to the highest traffic reduction in BitTorrent caching. Both the best-performing policy for Web caching (LFU, for the performance criteria we evaluate [12]), and for other peer-to-peer traffic (P2P) are outperformed by LRU.
- We show that, similar to Web caching, the effectiveness of BitTorrent caching grows logarithmically with cache size. However, larger caches are necessary to achieve high effectiveness in BitTorrent traffic compared to Web and other peer-to-peer traffic.
- We analyze the point where caching provides greater reductions to transit traffic compared to locality-aware neighbor selection strategies for several scenarios, putting in perspective the requirements and performance of both approaches. Our results suggest that caching is a viable alternative for ISPs, specially while locality-aware mechanisms are not easy to adopt.
- We provide a comprehensive analysis of the characteristics in BitTorrent traffic that lead to the good performance of LRU. Our results show that a marked temporal locality in BitTorrent traffic chiefly influences cache effectiveness.

In the remainder of this paper, before presenting our results and analyses (Sects. 5 and 6), we review the relevant background and related work (Sect. 2), present a model that simplifies the simulation of BitTorrent caching (Sect. 3), and detail our experimental setup (Sect. 4).

2 Related work

This section contextualizes this study in relation to previous work. Because most studies on BitTorrent and peer-to-peer caching use object replacement policies first applied to Web traffic as baselines, we start with an overview work on Web traffic caching. Next, we review work on caching traffic from BitTorrent and other peer-to-peer systems. When examining both Web and peer-to-peer traffic caching, we focus on results related to the effect of cache size and object replacement policies on cache effectiveness. After reviewing work focusing on caching, this section discusses locality-aware mechanisms proposed in the literature.

2.1 Web caching

The relation between performance and cache size has been widely studied in Web caching. In general, small files are more popular in Web workloads, which leads to relatively small caches producing high hit rate [35]. Moreover, the cache hit rate increases logarithmically as a function of the cache size [12].

Regarding object replacement policies, three main types of policies initially used in file systems caches [35] are widely used in the context of Web caching [39]: *(i)* policies based on the object *size*, among which the most natural is *SIZE*, accord-

ing to which the largest objects in the cache are replaced first; *(ii)* policies based on the *frequency of requests* to the objects, among which the most natural policy is *least-frequently-used* (LFU), in which objects with lower access counts are replaced first; and *(iii)* policies based on the *time of the most recent reference* to each object, among which the most natural policy is *least-recently-used* (LRU), which determines that the object accessed the longest ago is replaced first.

Several studies analyze replacement policies' performance in Web caching [1,6,7,41,52]. These studies show that recency-based policies perform poorly for Web caching, because the temporal locality of access in Web traffic is not strong. In turn, frequency-based policies show good performance, outperforming recency-based and size-based policies [1,4,7]. The higher performance of frequency-based policies occurs because of an invariant in the concentration of references in Web traffic. The concentration of references in Web traffic is analogous to the object's popularity and can be modeled by a Zipf distribution [1,6,14,15]. This means that a small number of objects are extremely popular, while there is a long tail of unpopular objects, a characteristic that favors frequency-based policies in a workload [12].

2.2 Caching BitTorrent and other peer-to-peer traffic

Multiple designs are possible for caching BitTorrent and other peer-to-peer traffic[23,28,37,38]. Lehrieder et al. suggest a taxonomy of caches for peer-to-peer traffic with three categories [28]. *Transparent caches* use deep-packet inspection to intercept and answer requests made by its users to peers outside the ISP; these are currently exemplified by PeerApp's UltraBand technology [38]. *ISP-managed ultra-peers* are high-capacity BitTorrent peers operated by the ISP that discover peers through the regular BitTorrent protocol, and prioritize uploading to users inside the ISP network; these are presently illustrated by Oversi's OverCache product [37]. *ISP-managed caches* are the third category and are based on extensions to BitTorrent that enable users to discover and prioritize downloading from caches operated by the ISP.

Our work focuses on caching effectiveness considering transparent caches. Moreover, we consider that the transparent cache provides a download speed to its clients that is similar to what the client would achieve from the original sources. As pointed out by Lehrieder et al. this design is ISP-friendly, as it is less susceptible to increase the speed with which illegal content is potentially distributed [28]. Also, it is important to highlight that the ISP operates a cache to reduce the inter-ISP traffic, and not to increase the download speed of its customers.

Our focus on transparent caches is necessary to enable an accurate trace-driven analysis of cache behavior. Events recorded in the trace that potentially drive user behavior are not changed by a transparent cache, while the other cache designs result in different download times and, hence, in different trigger events. There are presently no clear models of how users would react to the differences introduced by such caches, and developing such a model is out of the scope of this work.

Previous work provide evidences that it is possible to achieve high byte hit rate when caching BitTorrent traffic [2,3,26]. Ager et al. examined the effect of a cache with infinite size on a trace of user behavior in an ISP and suggest that caching BitTorrent traffic can yield high byte hit rates [2,3]. They also find that the performance of a cache improves with the number of clients using it. Likewise, Karagiannis et al. employ a cache with infinite storage on a different trace, and comment on the cacheability of this traffic, reporting that such a cache can lead to a byte hit rate higher than 0.9 [26]. Our work complements these efforts by (i) going one step further and examining how cache size and object replacement policies affect cache performance, and (ii) performing our analyses on multiple traces of BitTorrent usage that include different types of content and user bases. To the best of our knowledge, this is the first study that focuses on analyzing BitTorrent caching design considering finite cache size and object replacement policies that leverages a wide range of traffic characteristics.

Another thread of related work for the behavior of caches in BitTorrent traffic is the experience with other peer-to-peer traffic. Regarding the relation between performance and cache size in peer-to-peer traffic, Wierzbicki et al. show that in FastTrack traffic there are required larger cache sizes than Web traffic to attain a given byte hit rate, because files requested in Kazaa are typically much larger than in the Web [50,51]. Furthermore, other studies showed that in Web, Kazaa and Gnutella traffic the sizes of accessed files vary substantially, and there is a strong preference for small files [9,15,20,44], which may increase the performance of a small cache.

Some studies analyze the effectiveness of replacement policies for peer-to-peer caching [24,50]. The observation that file popularity in peer-to-peer traffic does not fit well to a Zipf distribution has motivated the development of new replacement policies for caching peer-to-peer traffic [5,20,50]. Wierzbicki et al. propose the policy *least-sent-bytes* (LSB). This policy works on parts of requested objects, and evicts from the cache first the parts of objects that have served the least number of bytes. LSB performs equivalently or better than LRU and LFU in Wierzbicki et al.'s experiments. Hefeeda and Saleh propose a proportional partial caching algorithm for Gnutella traffic named P2P [24]. This policy segments files and admits new segments according to the number of references to the stored segments. Hefeeda and Saleh show that LSB performs better than LRU and LFU, but worse than the P2P policy for Gnutella traffic.

In this study, we evaluate both LSB and the P2P policies in the context of BitTorrent traffic. Although there are similarities between BitTorrent and other peer-to-peer traffic, there are also notable differences. In particular, previous works have pointed out differences between BitTorrent and other traffic with respect to locality of references [5,21,40]. We extend these results by examining other peculiarities in BitTorrent traffic that are relevant for caching, and by investigating how LSB and P2P (together with other policies) are affected by these peculiarities.

2.3 Locality-aware mechanisms

Multiple designs are possible for locality-aware mechanisms that decrease BitTorrent inter-ISP traffic. Several studies have shown that locality-aware mechanisms reduce inter-ISP traffic and can be exploited in several ways: shifting inter-ISP traffic to local links of content distribution networks [47], biasing BitTorrent neighbor selection [11,30,31,47] or uploading policies [31], and through proxy-trackers, which intercept requests from peers in an ISP and redirects them to active peers inside the boundaries of the same ISP's network [26].

Some studies relate the locality-aware and caching mechanisms [11,26]. Relevant to the present work, Karagiannis et al. find that proxy-trackers deployed at the edges of the network reduce the peak demand for download bandwidth in an ISP, but overall keep the demand for download bandwidth multiple times higher than an infinite cache [26]. Bindal et al. using a simulation model, report that when the cache is combined to a locality-aware mechanism, there is a reduction on the peak and average bandwidth needed by the cache to serve its clients [11].

Our work evaluates the effectiveness of different caching strategies under realistic scenarios, including limits on the cache size and analysis of multiple traces, and compares their performance with a locality-aware neighbor selection mechanism. We note that the comparison of caching performance against that of a locality-aware mechanism aims simply to understand the impact that these two techniques may have on the reduction of inter-ISP traffic, and not to promote caching as an alternative to locality-aware mechanisms, or vice-versa. It is possible, as indicated by the preliminary studies reviewed above, that the best performance will be attained by a suitable combination of the two mechanisms. A study of such combinations, however, is beyond the scope of this paper.

3 Modeling caching and locality-aware mechanisms

Our method to investigate the effect of multiple factors on BitTorrent cache effectiveness is to simulate how a cache would perform if submitted to different workloads derived from real traces of user behavior. The use of simulations is justified because, on the one hand, obtaining access to the traffic of an ISP and submitting it to the effect of a cache is not feasible, and, on the other hand, resorting to simulation allows us to control experimental factors and study a wide range of parameters and configurations.

This section details the simulation model for evaluating the effects of caching and a locality-aware mechanism in the traffic described by a BitTorrent trace. Before that, we describe BitTorrent's functioning and a model for deriving, given a trace of user behavior, the transit traffic BitTorrent generates.

3.1 BitTorrent overview

A BitTorrent system is composed of a set of users that participate in *torrents* over time. A torrent is a group of users collectively distributing and/or downloading a file.[2] At a given moment, a user may be online as a node in none, one, or several torrents in the system. In each torrent, a node discovers peers by periodically contacting a centralized discovery component named *tracker*. In each contact, the node updates the tracker on current download and upload progress, and requests a number of randomly selected peers. The peers with whom a node is connected at an instant in time form its neighborhood set, and each node uses the tracker to maintain a neighborhood set of at least 50 peers, a default value used by several BitTorrent clients [25,46].

To be distributed in a torrent, a file is divided into a number of pieces. When a peer has all the pieces of the file in a torrent, it is called a *seeder*; otherwise it is named a *leecher*. A *session* is a contiguous period during which a peer is online as seeder or leecher in a torrent, and peers may have multiple sessions throughout their participation in each torrent. During each session where a peer is a leecher, it continuously requests pieces from the peers in its neighborhood, part of which may be in another ISP.

3.2 Estimating intra- and inter-ISP traffic

Our goal is to estimate the effect of traffic reduction mechanisms—a cache and a proxy-tracker—on the traffic requested for peers outside an ISP of interest. This traffic is derived from traces of user behavior (described in detail in Sect. 4.1) that depict the behavior of dozens of thousands of users. In summary, the trace stores download sessions, a continuous period that the peer remain online in a swarm. For each user session, it has available information of its start time,

[2] A torrent may distribute multiple files. Nonetheless, for simplicity of presentation, throughout the paper we refer to the single file case, and call the content distributed in a torrent a file.

end time, the amount of data downloaded and whether the peer is a seeder or a leecher. The first step in this process is, given a set of users inside an ISP, to determine the traffic that passes through the traffic reduction mechanism. Because the mechanisms we consider target only downstream traffic, our task is to determine which pieces of the file users download from outside the ISP over time.

There is a sizable issue with the complexity of precisely simulating this in a large-scale multi-torrent system. Files are often in the order of hundreds of megabytes [56], and, hence, have thousands of pieces, and the simulation may have thousands of clients downloading hundreds of different files simultaneously. The processing required to compute the effects caused by every piece downloaded in the system are prohibitive.

Our approach to circumvent this difficulty is to simulate the system with a coarser granularity, considering a file is composed of *segments* that are larger than pieces. The fewer the parts that comprise a file, the less costly it is to simulate the system, and we show in Sect. 4.2 that if the segment size is chosen appropriately, it is possible to reduce the cost of processing and indexing data in the simulator while maintaining accuracy.

Besides coping with the number of pieces to be indexed, the simulator must also reflect the effect of piece selection policies. In BitTorrent, peers choose which piece to download next based on the rarity of pieces in their neighborhood. Again, recreating the precise order in which pieces are downloaded, incurs an impractical complexity when simulating thousands of peers participating in thousands of torrents. To avoid this complexity, we model the effects that any piece selection policy can have on a traffic reduction mechanism—both a cache and a proxy-tracker. The key insight in this model is that any piece selection policy will only determine, for a given number of pieces that a user downloads in a period, (i) how many of the pieces are downloaded from peers in the same ISP, and (ii) how the traffic requested from outside the ISP affects and is affected by the traffic reduction mechanism. We now describe how to model the first aspect. The second one is different for a cache and a proxy-tracker, and is described in Sects. 3.3 and 3.4.

Deciding which proportion of the download that a user performs in a session is from peers in the same ISP is done as follows. The traces provide us with the amount of segments r requested during each session of each leecher P, and we assume segments are downloaded at constant throughput during the session. Following the trace, the simulator details tracks peer discovery dynamics over time (tracker contacting period is set to 20 min [36]). For each new segment node P downloads, the chance that P downloads it from a peer inside P's ISP is the proportion of P's neighborhood set that sits inside this ISP at the moment the download starts. If P's request should be directed to nodes inside its ISP but none of these nodes has the requested segment, the request is directed to nodes outside the ISP.

Effectively, this approach implies that if a fraction f of P's neighbors are in its ISP, P will download a proportion f of traffic from them. This model does not consider the effects of the heterogeneity of peer's connections on data provisioning. Nevertheless, note that this approach becomes accurate as the number of peers online grows and approximates reality for a large enough number of peers. This happens because, for a large number of peers, the average capacity of peers inside and outside the ISP becomes equivalent and so does the chance of selecting peers inside and outside the ISP for some download.

Given the traffic directed to peers outside the ISP of interest, we must next estimate the traffic reduction provided by the caching and proxy-tracker mechanisms.

3.3 A model for BitTorrent caching

BitTorrent transparent caching works intercepting requests for segments that peers inside the ISP sent to peers outside the ISP. The cache intercepts this type of request, and checks if the segment to be transmitted is already stored locally. If it is, the segment is not downloaded again but transferred from the cache to the requesting peer. If it is not stored locally at this time, the request is relayed to the peer outside the ISP, and the caching mechanism waits for an answer. As the peer outside the ISP answers the request, the segment is transmitted to the requesting peer, and stored in the local cache. If the cache is full, the mechanism uses an object replacement policy to free up space, before attempting to store the segment in the cache.

From the model described so far, it is possible to estimate the segments a node downloads from outside its ISP during a period. To compute the precise byte hit or byte miss associated with this download, it is necessary to determine which segments were downloaded by the user and in which order. This is necessary because each segment may coincide with a segment the cache currently stores or not. On the other hand, as discussed before, simulating the dynamics of piece selection is computationally prohibitive.

Our approach to this issue is to, instead of calculating the precise byte hit resulting from a user download, compute the limits for the best and worse effects that piece choices can have on the cache performance. This computation is at the same time simpler and more general than simulating a particular piece selection policy, and our results (in Sect. 5) show it is still possible to differentiate the performance of alternative cache designs considering these limits.

The upper limit for the effect of piece choice on the cache is an *optimistic* case: when a peer requests a segment to the cache, we consider that, if the cache stores segments that the peer does not own, the segment requested by the peer will be

one present in the cache. This results in the maximum byte hit for each request. In the *pessimistic* limit, conversely, we consider that a peer requests a segment stored in the cache only if there are no other segments that the peer may ask from the cache. This results in the minimum byte hit rate each request can produce.

Formally, the simulator computes the amount of requested segments intercepted by the cache r_c, and the amount of segments that resulted in hit h_c, i.e, the amount of segments that were served by the cache over a simulation time. Thus, given the segment size s, the traffic reduction provided by caching is $t_c = h_c \times s$, and the cache byte hit rate is computed as $b_c = h_c/r_c$. For each simulation configuration, there is a traffic reduction and a cache byte hit rate for optimistic and pessimistic scenarios.

We use optimistic and pessimistic limits to simulate what would be the best- and worst-case in a given configuration. Although these boundaries do not give us the exact effectiveness of a given cache configuration, they allow us to compare different cache configurations. If the best simulation scenarios for a configuration A is worse than the worst simulation scenario of configuration B, we can affirm that caching in B performs better than in A. If the interval between the best and worst estimates of two configurations overlaps, however, we cannot affirm that their effectiveness is different.

3.4 A model for a BitTorrent proxy-tracker

We simulate a locality-aware neighbor selection mechanism as implemented by a proxy-tracker at the edges of an ISP network [26]. Although multiple designs have been proposed for increasing traffic locality, the proxy-tracker provides a reference of the highest locality achievable by locality-aware mechanisms. Our goal is to use the results from this reference as a baseline for the traffic reduction provided by a caching mechanism.

The proxy-tracker works by intercepting segment requests from local peers and redirecting them to active peers inside the ISP network that can fulfill such requests. To compute the effect of this behavior on BitTorrent traffic, the simulator computes the byte hit similarly to the optimistic limit in the cache model. For each segment requested by a node P to a peer outside its ISP, a hit occurs if there is some active peer in the ISP that has a file segment that P does not have. If peers inside P's ISP only have segments that P already owns, P's request is directed to peers outside the ISP. Note that our simulation differentiates seeders and leechers. When there are seeders peers within the ISP, the request is always satisfied. Also, it is important to remark that considering the optimistic limit for the byte hit of the proxy-tracker, our baseline represents an unfavorable scenario for the caching mechanism in any comparison we perform.

More formally, the simulator computes the amount of segments intercepted by the proxy-tracker, and redirected to peers inside the ISP (r_p), and the amount of segments that resulted in hit (h_p), i.e, those segments that were served by peers inside the ISP over the simulation. These values allows us to compute the traffic reduction provided by the proxy-tracker mechanism as $t_p = h_p \times s$, and its byte hit rate as $b_p = h_p/r_p$.

3.5 Trace-driven simulation dynamics

For our experiments, our models described in this section are implemented in a trace-driven simulator, which we now detail further. For all experiments, each simulation is driven by a workload derived from traces that describes download and upload sessions of a set of users (Sect. 4.1 details the real traces used). The workload is composed by a series of user sessions ordered by their start time. For each session, the workload contains: a user identifier, start time, end time, number of bytes requested, and the size of the file accessed.

The simulator processes each session in the workload as follows. At the session start time, the peer starts the session in the tracker and requests a list of up to 50 other peers currently online in the same torrent. The tracker returns a list of peers randomly chosen among all peers online in the torrent. The peer performs new requests to the tracker every 20 min (usual period between requests to the tracker [36]) and tries to keep a neighborhood set of at least 50 peers in case there are less than 50 peers in the torrent, neighborhood set sizes are limited to the number of available peers.

The content downloaded during the session is divided in segments. All segments are downloaded at a constant throughput, calculated as the size of all segments requested during the section divided by the duration of the session. When a peer P will download a segment, it randomly selects a peer among the set of peers to which it is connected. If the chosen peer is in P's ISP, the segment is downloaded with no interference from other parties. If the chosen peer is outside P's ISP, the simulation differs depending on the use of the proxy-tracker or caching. The simulation is performed separately for each mechanism. Using a proxy-tracker, the peer request is intercepted by the proxy-tracker, and it processes the download as described in Sect. 3.4. Using caching, the request is first directed to the cache, and can result in a cache miss or hit as described in Sect. 3.3. All caching simulations exclude the cache warm-up phase, i.e, the computation of cache hits and misses start only after the cache is full.

At the end of a session, the peer closes all connections with other peers and with the tracker. The simulation ends when all sessions in the workload end. As the workload is processed, the simulator calculates the traffic reduction achieved by each mechanism.

Table 1 Characteristics of the traces used

Trace	Start	Duration (days)	#Users	#Torrents	#Session	Download traffic (TB)
Alluvion	2003–10–12	30	39,572	1,474	359,380	110.83
Bitsoup	2007–04–28	7	44,678	6,571	969,024	160.65
Etree	2005–03–11	10	46,801	1,589	141,945	39.47
Filelist	2005–12–11	7	46,705	3,853	7.765.819	66.15

4 Experimental setup

After presenting our approach to simulate traffic reduction mechanisms for the BitTorrent system, we focus on our method for analyzing the effectiveness of such mechanisms. This section details the data used in our trace-driven simulation, and the scenarios evaluated.

4.1 Datasets

We use traces of user behavior in four BitTorrent sites[3] for our analyses: Alluvion, Bitsoup, Etree, and Filelist. The four traces were collected by three different research groups and are available in the Peer-to-Peer Trace Archive.[4] Although these traces have been analyzed for different purposes in the past (eg., [5,10,21]), they have not been analyzed with regard to the cacheability of their traffic, or focusing on how traffic invariants affect cache performance. Table 1 depicts the characteristics of these data sets.

Each of the four BitTorrent sites in the traces revolves around a directory of torrents distributing content published by its users. Bitsoup and Filelist allow their users to publish all types of content and are particularly popular for the distribution of movies. Etree is restricted to the sharing of lossless audio recordings of live performances. Alluvion is a companion site to a popular discussion forum and, at the time of data collection, distributed miscellaneous content produced by the users of this forum.

The sample constituted by these four sites is neither a comprehensive nor a random sample of BitTorrent sites. Instead, the use of these sites is dictated primarily by feasibility in data collection: these sites made it possible to obtain detailed information about user behavior over time, a rare feature among BitTorrent sites. Nevertheless, in spite of not being random, this sample contains two sites of a similar type—Bitsoup and Filelist—and two sites of considerably different audience—Etree and Alluvion. Regularities found across these four sites are evidence of generality for our results.

Besides content directories, all four sites had, at data collection time, additional Web pages that tracked the upload and download progress together with the uptime of users in all torrents. The traces were collected by periodically scraping these Web pages. Scrapes were hourly for Bitsoup, Etree and Alluvion, and happened every 6 min for Filelist.

The IP addresses of users are not available in any of the traces. With the identification schemes in the four traces, it is neither possible to map users to ISPs nor to map names in different traces to the same user. A second information that is not present in the traces is the capacity of users' Internet connections. Although it is possible to deduce their experienced upload and download speeds from the traces, it is not clear if these correspond to the capacities of their connections.

Given the series of observations that constitutes each trace, we reconstruct users' sessions similarly to Andrade et al. [5]. Following this approach, a user is online during a period if the user was online in the first and last observations of a torrent in that period. A contiguous period online is termed a session, and the amount downloaded in a session is that reported in the last snapshot minus that reported in the first snapshot defining the session. For each user session, there is available information of its start time, end time, the amount of data downloaded and whether the peer is a seeder or a leecher.

It is worthwhile mentioning that there exist sites much larger than those in our sample, such as, for example, The Pirate Bay. Although using data from one of such sites is desirable, obtaining such data with the level of detail in our traces is arguably infeasible: there are millions of users and no accessible centralized registry for their state. Furthermore, one should note that the vast majority of BitTorrent sites with large user bases are not nearly as large as The Pirate Bay [57]. In this perspective, our traces represent the typical large BitTorrent sites [57].

Besides the issue of considering average-sized sites, we also consider a limited number of them. Although a larger number of sites is certainly desirable for further analyses, it is important to stress that the available studies of BitTorrent caching and most of the literature in Web caching (discussed in Sect. 2) has relied on experiments using one trace only. In this respect, this work is also novel in comparing multiple BitTorrent traces.

[3] Sometimes also referred to as BitTorrent communities.

[4] http://p2pta.ewi.tudelft.nl.

4.2 Evaluation scenarios

This subsection presents the settings of the simulations scenarios in which an ISP implements either a *caching* or a *proxy-tracker* mechanism. The simulation is driven by a workload derived from the traces that describes the download sessions of a set of users. The content requested during a session is divided in segments of 10 MB. We find that a segment size lager than 10 MB would generate a significant change in the amount of download users perform in the trace. The cumulative distribution function (CDF) in Fig. 1 shows that the use of this segment size does not bias the amount of download done by users during a session when compared with a simulation that considers piece granularity. The effect of segment size in all other workloads is similar to the one shown in this figure.

Users in the workload are divided between those inside and outside the ISP we simulate. For every simulation, a proportion of all peers is randomly allocated for each position. Because previous studies have shown that the performance of both proxy-tracker [26,30] and cache with unlimited size [2] vary with the number of users inside the ISP, we consider two cases for the proportion of peers inside the ISP of interest: 3 and 0.3 % of all users in the workload. Considering 3 % of the users in each workload as belonging to the ISP, our simulation fulfills two requirements in our evaluation: (i) the population of users in the ISP is larger than 1,000 users for all workloads, and (ii) the number of users inside the ISP is never larger than the number of files in the workload, guaranteeing that our sampling does not artificially increase the locality of reference in the workload. To contrast with the situation where 3 % of the users in each workload as belonging to the ISP we simulate, we also examine the scenario where the proportion of users in the ISP is 10 times smaller (0.3 %).

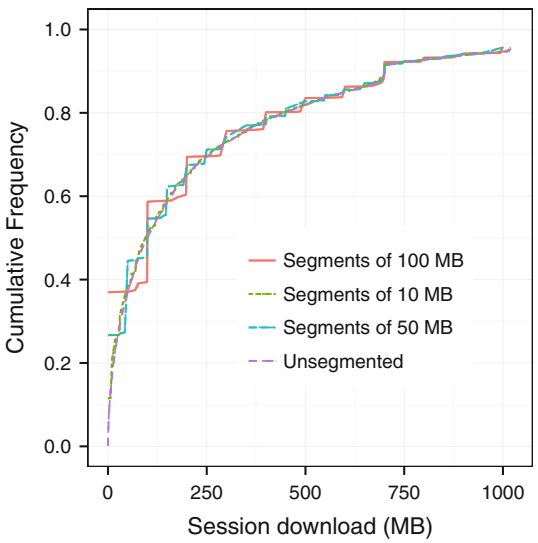

Fig. 1 Cumulative distribution function of the download happening in sessions for different segment sizes in the workload alluvion

Table 2 Characteristics of an ISP formed by 3 % of the workload users

Trace	Download traffic (GB)	Ideal cache size (GB)
Alluvion	2,025.472 ± 88.063	344.064 ± 6.144
Bitsoup	3,177.472 ± 77.823	1,708.031 ± 39.936
Etree	581.632 ± 16.384	297.983 ± 7.168
Filelist	1,449.983 ± 36.863	376.831 ± 7.170

When simulating caching, each simulation considers, besides the number of ISP clients, a cache size and an object replacement policy. We simulate cache sizes up to 50 % of the size necessary to accommodate all files requested by the users in the ISP we simulate. This cache size is henceforth dubbed *ideal cache size*. Our limit on the maximum cache size simulated is chosen so as not to spend a larger proportion of the trace in the warm-up phase. In all of 30 simulation for each BitTorrent workload, no more than 43 % of the requests in the trace was spent in the warm-up phase with cache size of 50 % of the the ideal cache size. Table 2 presents the ideal cache size for each workload when the simulated ISP is composed by 3 % of the users in the workload. The variation in cache sizes we analyze also allows us to model how the cache effectiveness varies with its size.

Caching simulations consider the five object replacement policies presented in Sect. 2: LFU, LRU, LSB, P2P, and SIZE. This set of policies allows the comparison of three reference policies that explore basic traffic characteristic relevant for caching—LRU, LFU and SIZE—and two policies proposed for caching peer-to-peer traffic: LSB, proposed for caching FastTrack traffic [51], and P2P, proposed for Gnutella traffic [24]. No object replacement policies are necessary when simulating a proxy-tracker.

Comparisons between the performance of two scenarios are done based on the interval defined by the *byte hit rate* of the scenarios for the pessimistic and optimistic estimations in our cache model. If two intervals do not overlap, we can claim the cache in one of them is necessarily more efficient than the other. In all scenarios evaluated, we focus on evaluating the incoming traffic at the ISP, and use the byte hit rate as a metric to measure the reduction of traffic yielded by the strategies. Henceforth, all our results are averages of 30 simulation runs shown with 95 % confidence intervals.

5 BitTorrent cache effectiveness

This section presents results of our simulations to analyze the performance of BitTorrent caching under various circumstances. First, this analysis focuses on the effect of number of

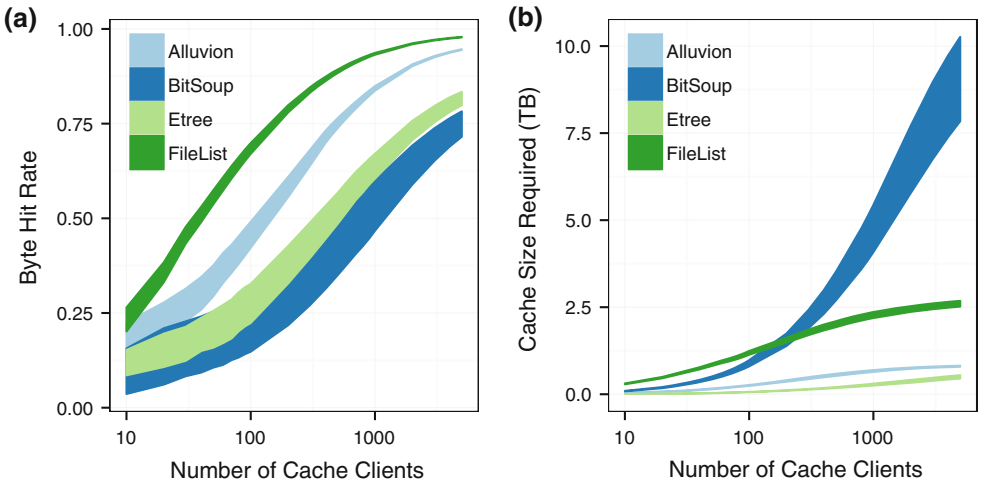

Fig. 2 Theoretical caching potential with different number of clients. *Each colored area* represents the space between the pessimistic and optimistic cases in one scenario. The *solid line* on *top* of an area is the optimistic result and the *line* under it, the pessimistic one. The relative error is at most ± 0.05 for a confidence level of 95 % **a** Byte hit rate **b** Cache size

cache clients on BitTorrent traffic cacheability. Then, it analyzes how inter-ISP traffic reduction is affected by the object replacement policy, and the cache size. Finally, it compares the traffic reduction achieved by caching with that achieved by a proxy-tracker.

5.1 Effect of number of cache clients on BitTorrent traffic cacheability

We first examine how the efficiency limit of the cache varies when the number of cache clients increases. This limit, termed cacheability, quantifies the caching potential of BitTorrent traffic and how the scale of the user population affects this potential. The natural measure for the cacheability is the byte hit rate. We isolate the effect of user population from cache size and object replacement policy by simulating a cache of infinite size.

The population of cache users in this experiment is a randomly selected and ordered sample of 5,000 clients from each trace. We simulated ISP with up to 5,000 clients in order to avoid artificial cache hits. Artificial cache hits are inevitably generated when the number of ISP clients becomes larger than the number of files in the trace. With these populations, we simulate BitTorrent caches serving the first client, the first two clients, the first three clients, and so on, all the way up to 5,000. Each simulation computes the byte hit rate for all requests simulated. There is no period of cache warm up in this experiment, as the cache has unlimited size.

Figure 2 shows both the byte hit rate as a function of the number of cache clients in this experiment (Fig. 2a) and the required cache size to achieve this byte hit rate (Fig. 2b). Our results show byte hit rates between 0.75 and 0.98 with 5,000 clients. The traces considered have a different number of peers, but 5,000 is less than 11 % of the peers in the trace with the smallest number of peers. These results show that (i) for BitTorrent traffic, caching achieves high byte hit rates even with few cache clients, and (ii) the benefits from the shared cache increases with the number of clients. However, larger cache size can be required to achieve such performance. The required cache size is related to the amount of context in the traffic. For example, in the BitSoup trace, the largest trace in terms of number of files, the cache size required to store the downloaded content in 7 days can reach up to 10 TB. For longer periods, new content will appear in traffic which will further increase the cache size required. Although the storage capacity is not a big problem nowadays, the efficiency of the indexing algorithm cache tends to decrease when size increase [8]. This puts into perspective the demand for replacement policies that are able to expore the cacheability of traffic when the cache size is limited.

5.2 Impact of object replacement policy and cache size

Our first experiment investigates the effect of different object replacement policies and cache sizes on the byte hit rate of the caching mechanism. The first questions we address are: *which replacement policy, among well-known Web replacement policies (LFU, LRU, and SIZE) and peer-to-peer replacement policies (LSB, and P2P), achieves the highest byte hit rate? How effective they are when compared to a Random replacement policy?* The answer to this question is important to inform designers and operators of caching mechanisms that must choose which policy to implement or use. The results of simulations for Web replacement policies and the Random replacement policy in all traces are shown in Fig. 3.

We find that LRU attains a higher byte hit rate than LFU, and SIZE in nearly all scenarios. The single exception is FileList with 50 % of the ideal cache size, where we cannot distinguish the byte hit rates of LRU and LFU. Overall, the maximum byte hit rate in the experiment lies between 50–80 % for all workloads with LRU and 50 % of the ideal

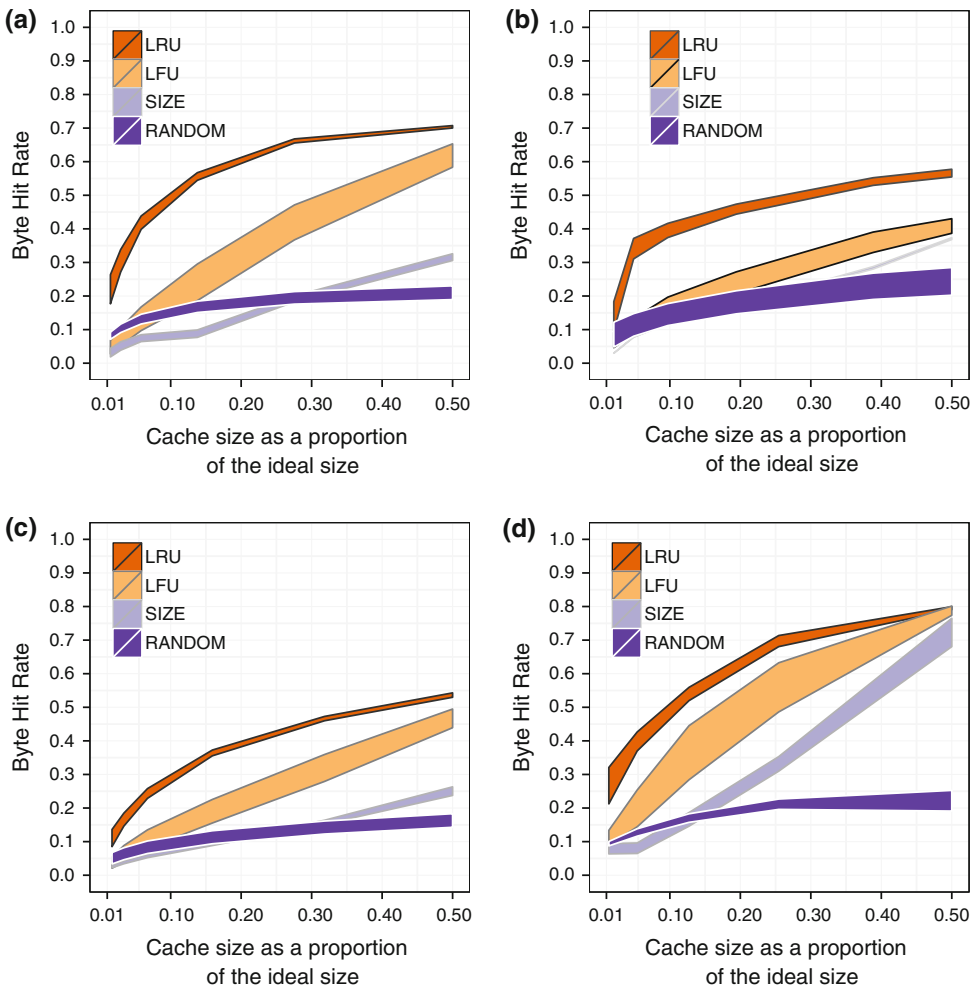

Fig. 3 Comparison of byte hit rate for LRU, LFU, SIZE, and Random policies varying the workload, and cache size. The relative error is at most ±0.05 for a confidence level of 95 %. All simulations consider ISPs with 3 % of the users in the workload **a** Alluvion **b** BitSoup **c** Etree **d** FileList

cache size. For most cache sizes, the Random policy show to be less effective.

Figure 4 compares the byte hit rate of LRU with the peer-to-peer object replacement policies LSB and P2P. P2P and LSB are undistinguishable in our experiment, while LRU clearly achieves a higher byte hit rate than both. Our results provide evidence that although LSB and P2P have been shown to perform well in other peer-to-peer traffic, LRU is more adequate for BitTorrent traffic.

The adequacy of LRU to BitTorrent traffic has significant implications for cache design. This policy is trivial to implement, has low computational complexity [42,49], and requires little state to be kept—only the order of arrival of objects. These requirements are significantly simpler than those posed by most of the other policies we consider, including those developed for peer-to-peer traffic. Our simulation results indicate that LRU allows for the development of effective caches for BitTorrent traffic.

We now turn to the question of *how the byte hit rate varies with the cache size.* This information is important to understand how to dimension a cache for BitTorrent traffic. To address this question, we model how the byte hit rate of LRU grows as a function of the cache size in our experiments. The model considers only LRU's byte hit rate, as it achieved the best performance in our previous experiments.

A linear regression of the byte hit rate as a function of the logarithm of the cache size results in an adjusted R^2 between 0.91 and 0.95 with errors statistically independent and normally distributed in all workloads, both for optimistic and pessimistic scenarios. This indicates that the byte hit rate grows logarithmically with the cache size, showing a sharp increase when the cache size increases at first, but with a decreasing additional gain with further increases in the cache size.

5.3 Comparison with the proxy-tracker

To put the cache performance in perspective, we now compare its traffic reduction with that of a proxy-tracker in the same scenario. The proxy-tracker, as we simulate, represents

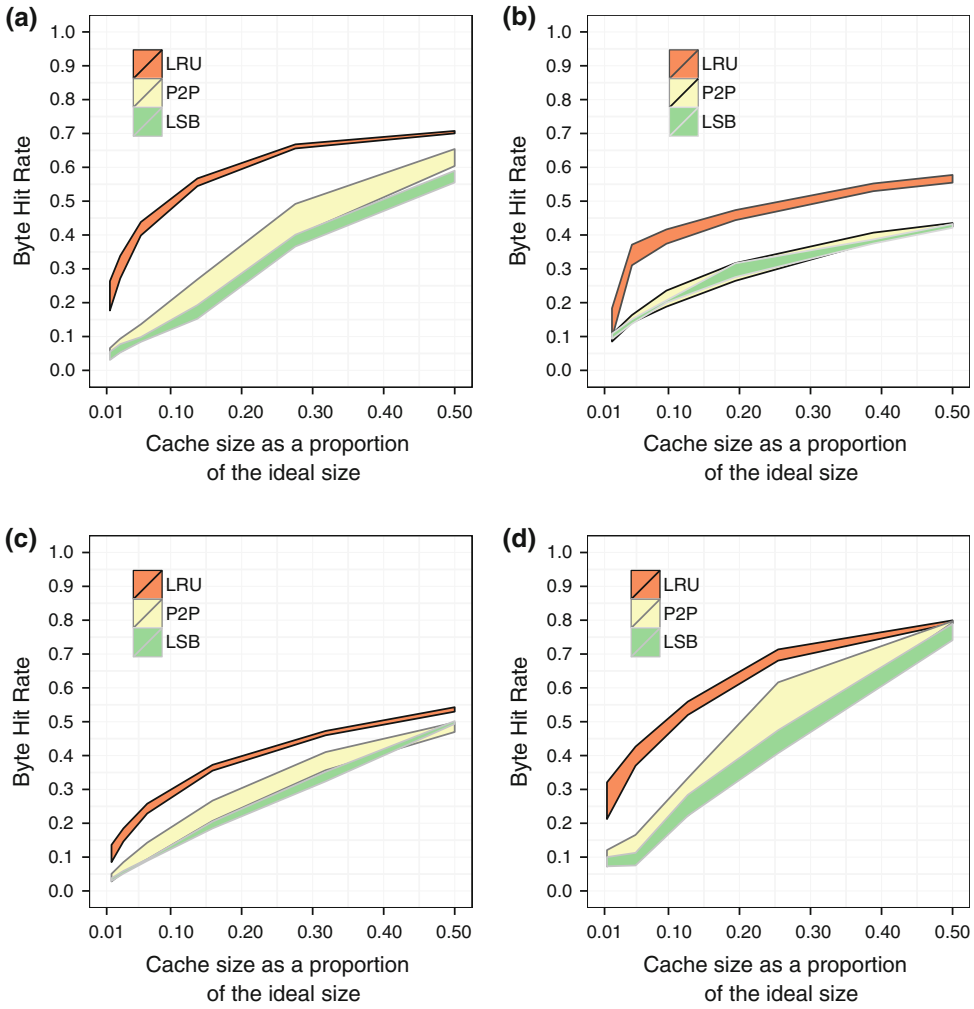

Fig. 4 Comparison of byte hit rate of LRU, P2P, and LSB policies for the different workloads and varying cache size. The relative error is at most ±0.05 for a confidence level of 95 %. All simulations consider ISPs with 3 % of the users in the workload **a** Alluvion **b** BitSoup **c** Etree **d** FileList

the best performance of a locality-aware neighbor selection mechanism.

Given that the performance of both mechanisms vary with the number of ISP clients [2,26,30], this experiment compares these mechanisms considering two possibilities for the number of ISP clients that differ from each other 10 times. This analysis aims at identifying whether the difference of performance between these polices varies with the number of ISP clients.

Figure 5 shows the traffic reduction achieved by proxy-tracker and LRU caching when 3 % of peers of the workload are inside the ISP and considering different cache sizes. This figure shows the results for FileList and Etree, which show, respectively, the best and worst cases for the proxy-tracker and for the LRU traffic reduction as a function of the cache size. The results show that LRU achieves higher traffic reduction than the proxy tracker when the cache size is higher than 30 % of the ideal cache size in Etree, and 20 % of the ideal cache size in FileList. These cache sizes are equivalent to 89.39 and 75.36 GB, respectively.

In turn, Fig. 6 shows the traffic reduction achieved by LRU caching and by proxy-tracker when 0.3 % of peers of the workload are in the ISP (10 times less than that present in Fig. 5). In this figure LRU caching achieves higher traffic reduction than the proxy-tracker when the cache size is higher than 10 % of the ideal cache size (equivalent to 44 GB). Furthermore, for all workloads, the traffic reduction achieved by caching LRU was less sensitive to variation in the number of peers inside the ISP than that achieved by the proxy-tracker. For example, in the LRU cache with a size of 50 % of the optimal cache size, the reduction of 10 times in the number of peers inside the ISP has generated on average a reduction of 63.63 % in the efficiency of caching and a reduction of 88.89 % in the the efficiency of proxy-tracker. In Filelist, the reduction is of 50 % in the efficiency of caching and of 84.61 % in the proxy-tracker efficiency.

In summary, the results in this section show that the traffic savings achieved by LRU caching increases as the cache capacity increases, outperforming the traffic reduction

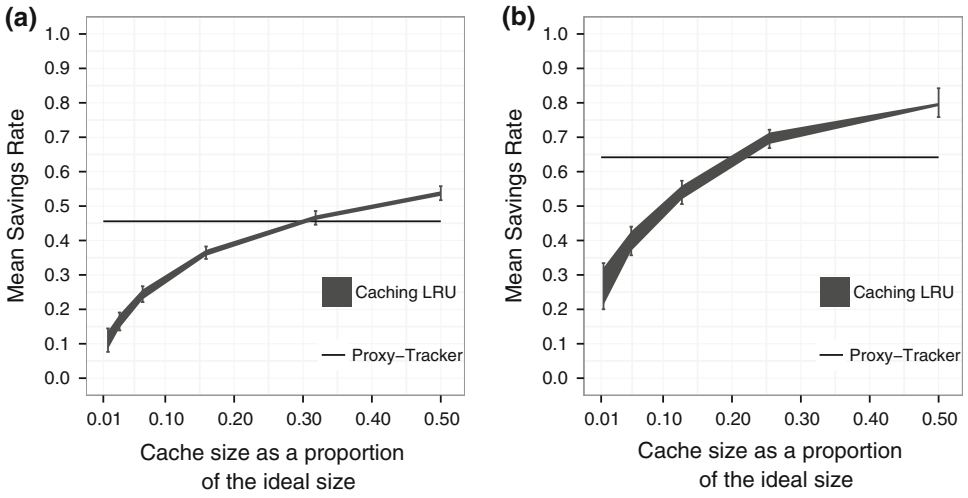

Fig. 5 Comparison of traffic reduction of LRU caching and proxy-tracker. The evaluated ISP consists of a sample of 3 % of peers in the workload, the other peers are in other ISPs in the network. In the result of LRU caching, we plot the *upper limit* of the confidence interval of the optimistic scenario and the *lower limit* of the confidence interval of the pessimistic scenario **a** Etree: 1403 peers inside the ISP **b** FileList: 1459 peers inside the ISP

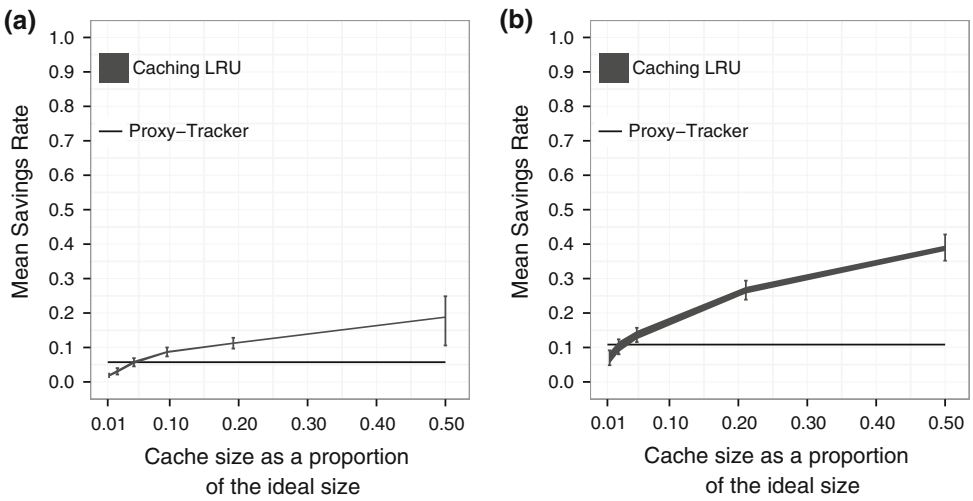

Fig. 6 Comparison of traffic reduction for a cache using LRU and a proxy-tracker. The simulated ISP consists of a sample of 0.3 % of peers in the workload; other peers are outside this ISP. In the result of LRU caching, we plot the *upper limit* of the confidence interval of the optimistic scenario and the *lower limit* of the confidence interval of the pessimistic scenario **a** Etree: 140 peers inside the ISP **b** FileList: 146 peers inside the ISP

achieved by the Proxy-Tracker mechanism. Naturally, cache is a limited storage and its effectiveness is impacted by its capacity. In all our analyzes, even a small cache (near 100GB) provided a higher traffic reduction than Proxy-Tracker. A larger cache size outperforms Proxy-Tracker because of its availability. The cache is always in place, while Proxy-Tracker needs that other peers are online and have requested the content in the same ISP of the requester peer.

6 BitTorrent traffic invariants

From the results presented so far, it is clear that there are differences in BitTorrent caching effectiveness in terms of replacement policies and cache size when compared to the Web and other peer-to-peer systems. These differences may happen because of different traffic characteristics. This section investigates which BitTorrent traffic characteristics impact on caching effectiveness, and how these characteristics differ from the Web and other peer-to-peer traffic.

Our analysis is based on more workloads than earlier studies of BitTorrent traffic [5,2,21,40], and focuses on three traffic characteristics relevant to caching: object temporal locality, concentration of references, and file size distribution. We model these traffic characteristics in our four BitTorrent workloads, and compare our results with those reported by studies on Web, FastTrack, and Gnutella traffic.

6.1 Object temporal locality

Temporal locality indicates how likely it is that an object referenced in the recent past is referenced in the near future [7,33]. A standard to measure the temporal locality in a workload is the Least Recently Used Stack Model (LRUSM),

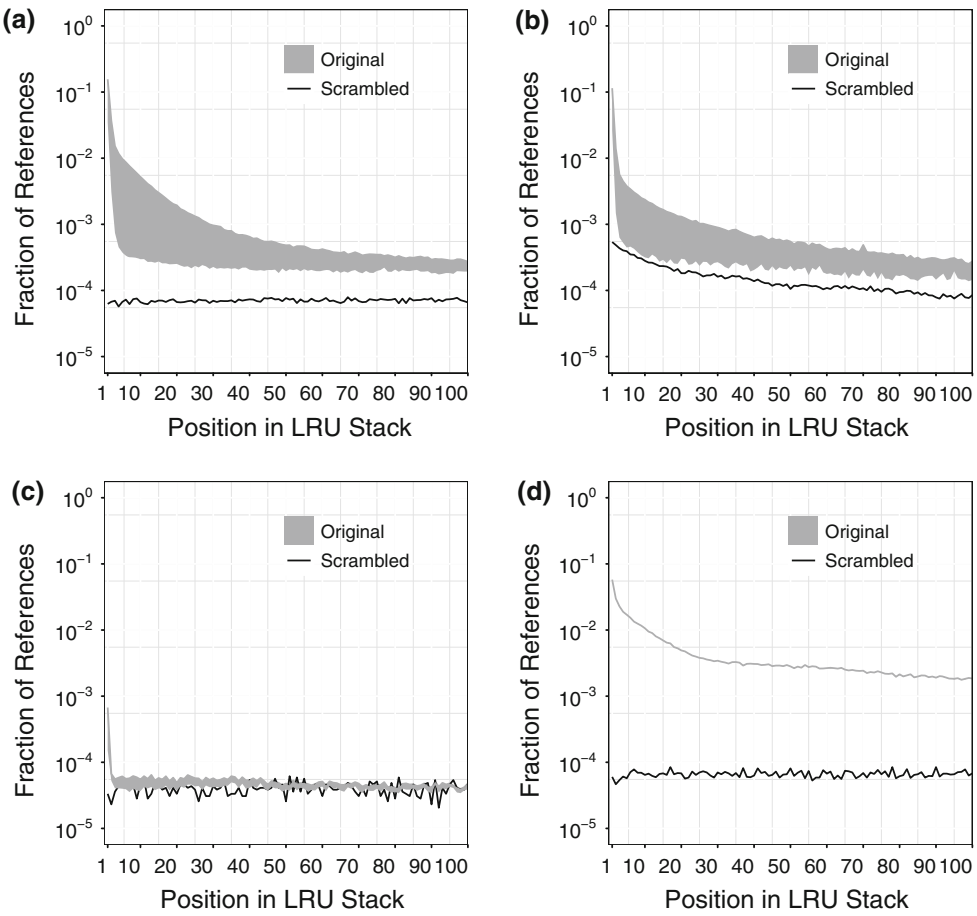

Fig. 7 Temporal locality for each BitTorrent site as measured by the Least Recently User Stack Model (LRUSM). The normal workload reflects the temporal locality in BitTorrent traffic. Scrambled data indicates only the locality resulting from file popularity **a** Alluvion **b** Etree **c** BitSoup **d** FileList

which works as follows: when an object is referenced in the workload, it is placed on top of a stack. All other objects already in the stack are pushed down by one position. When the object is subsequently referenced, its current position in the stack is accounted for in the experiment, and the object is moved back to the top of the stack. Other objects are pushed down, as necessary.

In LRUSM, temporal locality manifests through a high probability of references to positions at, or near, the top of the stack. At the same time, a high probability of references to the top-positioned objects in the LRUSM can be a result of the sheer popularity of files. To isolate the effect of temporal locality, we compare the behavior of LRUSM in our original workloads with that of a modified version of the workload where the order of requests is scrambled. The scrambled traces neutralize the effect of temporal locality in the workload, isolating the effect of file popularity in the LRUSM [18].

Figure 7 shows the comparison between the normal and scrambled workloads. In this figure the original access are represented by an area formed by the optimistic and the pessimistic scenarios. The distance between the optimistic and pessimistic scenarios is impacted by the temporal locality. For example, in FileList (Fig. 7d) the distance between these scenarios is small and tends to a line because of the strong temporal locality found both in segments and in file accesses.

Filelist (Fig. 7d) and Etree (Fig. 7b) have a considerably higher number of requests made to the few first positions in the stack when compared to the other two workloads. We conjecture that this happens because of the widespread use of automated download mechanisms by users of these sites. In fact, we observed that the proportion of requests for a file happening on the first hours after the file publication is typically higher in these two workloads than in the other two.

We conjecture that the typical life cycle of files in BitTorrent traffic [21] is the main characteristic that contributes to high temporal locality (and the good performance of LRU). In BitTorrent, when a file is released, it usually receives many downloads in the first hours and days of its distribution. This number of downloads achieves a peak and, after a certain time, it quickly reduces and tends to zero. The larger number of downloads that the file experiences when it released contributes to the temporal locality.

In spite of this peculiarity, however, the effect of temporal locality is noticeably higher in our BitTorrent traffic workloads than that reported in Web traffic studies [7,52].

This observation justifies the higher byte hit rate achieved by LRU recency-based replacement policy in our BitTorrent caching simulation.

6.2 Object popularity

After considering temporal locality, we analyze the object popularity in isolation. Figure 8 depicts the popularity of objects in our BitTorrent workloads. We find that in BitTorrent traffic the object popularity is well modeled by a Weibull or Log-Normal distribution, which implies in the absence of a long-tail and in a less pronounced concentration of references in the most popular files, as compared to typical Web traffic.

As a result, this concentration of references hampers the performance of the LFU object replacement policy in BitTorrent caching, which does not show the best performance as in Web caching. In Web traffic, few highly popular files are never replaced in the cache and are responsible for increasing the hit and the byte hit rate.

6.3 File size distribution

The third traffic characteristic that typically affects cache performance is the correlation between file reference and file size. In Web, FastTrack, Gnutella and Kazaa traffic, there is a wide variation in the size of objects, and smaller files are often more popular [9,15,17,20,44].

Figure 9 shows the cumulative distribution of file sizes in the BitTorrent sites used in this study. These results show that the size of files shared through BitTorrent is one order of magnitude larger than those accessed through Web and other peer-to-peer file-sharing systems [35,45,52].

Regarding the correlation between file reference and file size, we observe that in the workloads we have studied there is no strong correlation between the size of a file and the number of requests it receives. The low correlation can be seen in Fig. 10. To quantify it, we use the non-parametric statistic Kendall's τ coefficient [27], and measure the association or statistical dependence between file sizes and their popularity. There is only a weak correlation: in all workloads τ is between 0.09 and 0.25 (Alluvion = 0.11, BitSoup = 0.25, Etree = 0.09, FileList = 0.15), with p value <0.001. It is likely that the low tendency for users to request smaller files more frequently accounts for the poor performance of the SIZE object replacement policy in BitTorrent traffic, when compared to its performance in Web traffic.

In summary, our results indicate that LRU is the best performing policy for BitTorrent traffic mainly due to a pronounced temporal locality in file access. The adequacy of LRU for this traffic also benefits from a combination of a less concentrated distribution of popularity among files and low correlation between file sizes and popularity, which are the characteristics leveraged by other object replacement policies. These results have implications both for cache designers and operators. In particular, for cache designers, the understanding that LRU is an effective object replacement policy significantly reduces the complexity of implementing the cache. For operators, our results provide a model for dimensioning BitTorrent caches, and a thorough evaluation of cache performance with multiple workloads.

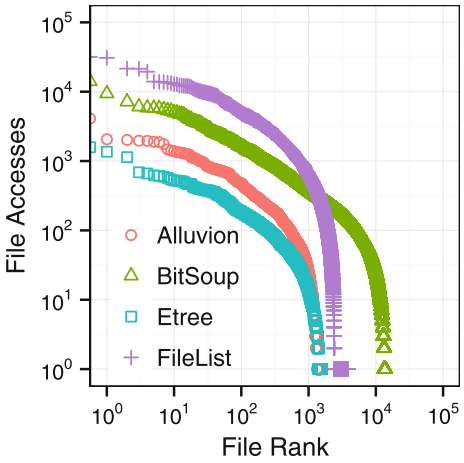

Fig. 8 Distribution of BitTorrent files popularity, frequency of file accesses versus file ranking

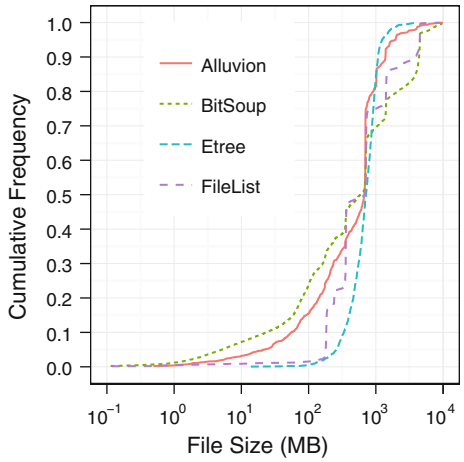

Fig. 9 Cumulative distribution function (CDF) of files size in the BitTorrent sites Alluvion, BitSoup, Etree, and FileList

6.4 Limitations

Our study, albeit aiming at capturing the central feature of the relation between BitTorrent traffic and caching, relies, as any simulation study, in a model that has a number of assumptions and simplifications. In the following, we discuss potential limitations in our approach stemming from such assumptions that should be considered by future work.

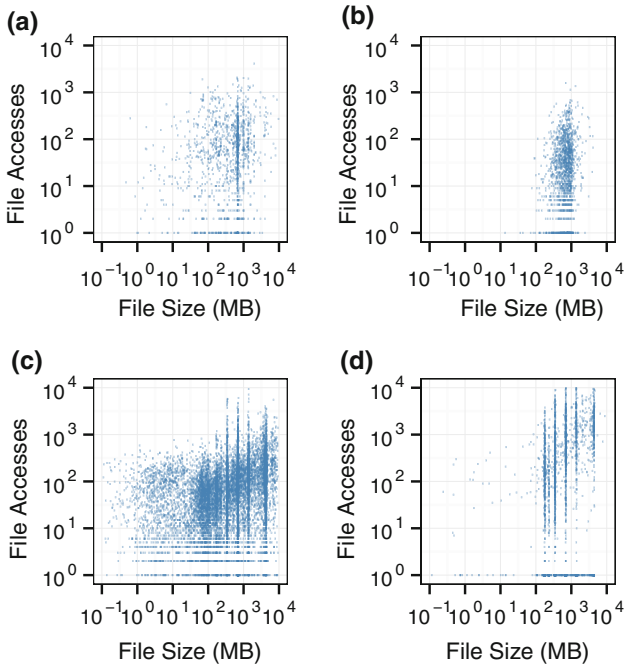

Fig. 10 Correlation between file reference and file size. Kendall's τ correlation coefficient shows weak correlation in all BitTorrent sites: **a** Alluvion $\tau = 0.11$, **b** Etree $\tau = 0.09$, **c** BitSoup $\tau = 0.25$, **d** FileList $\tau = 0.15$

- In this work, we assume that peers' request for segments are chosen in proportion to the number of peers in-side/out-si-de the ISP. Indirectly, this implies that, in our model, an equal number of peers inside and outside the ISP are able to provide the same aggregated throughput to the downloader. It is possible that in some situations the peers inside the same ISP as the downloader are able to provide a higher aggregate throughput. However, it is not clear that this higher throughput indeed manifests often, perhaps due to frequent last-mile congestion. Wojciechowski and Pouwelse overview this phenomenon and present an experiment that illustrates this argument [54].
- In a real situation, over a download session, BitTorrent peers experiment fluctuation in their throughput, for example, due to connection instability and/or to connection sharing with other protocols. In this work we compute the average throughput in the session (the ratio between the amount of data downloaded in the session and duration of the session) and we assume that segments are downloaded at this constant throughput. When observed in the perspective of each peer, this may impact the time a content is requested, and, therefore, the time the content enters and the time in which it is removed from the cache. When observed in a cache perspective, we believe our simplification has not a significant impact on the results because the cache that has thousands of clients with different fluctuations in throughput; thus, fluctuations in single peer is diluted when viewed from the perspective of the cache.
- We analyze BitTorrent traffic data collected in specific periods in time: 2003, 2005 and 2007. On the one hand, although collected at different periods, these data show similarities in the three main characteristics relevant to replacement policy evaluation: file size distribution, temporal locality, and files popularity. Furthermore, previous studies show that these features have not changed much in recent years[16,53,57]. On the other hand, it is important to highlight that our results are sensitive to changes in these traffic characteristics.

7 Concluding remarks

This study contributes to the understanding of how caching reduces the inter-ISP traffic associated to BitTorrent workloads. We analyzed the impact of cache sizes and well-known object replacement policies on BitTorrent caching effectiveness. To put cache performance in perspective, we compared its traffic reduction with that achieved by a proxy-tracker, which represents the best performance of a locality-aware neighbor selection mechanism. Furthermore, we investigated which BitTorrent traffic characteristics impact on caching effectiveness, and how these characteristics differed from the Web and other peer-to-peer traffic.

Regarding object replacement policies, LRU shows the best performance in all of our experiments. This result contrasts with that of experiments with Web and other peer-to-peer traffic in the literature, and points out that peculiarities in BitTorrent traffic must be accounted for when designing caches for this system. Examining further such peculiarities, we could identify the characteristics of BitTorrent traffic that lead to the good performance of LRU: a significant temporal locality, concentration of references that fits a Log-Normal distribution, and a weak correlation between file sizes and their popularity.

The adequacy of LRU to BitTorrent traffic has significant implications for cache design. This policy can be trivially implemented, has low computational complexity, and requires a small amount of state to be kept. These requirements are significantly simpler than those posed by most of the other policies that we considered in our study, including those developed for peer-to-peer traffic. In summary, our results indicate that LRU allows for the development of effective and scalable caches for BitTorrent traffic.

Analyzing the effect of cache size on its effectiveness, we found out that the byte hit rate of a BitTorrent cache grows logarithmically with its size. This result provides a model for cache provisioning. Putting the traffic reduction obtained with a certain cache size in perspective, our comparison of caching and the use of a proxy-tracker shows

that for the workloads we consider, reasonable cache sizes can outperform locality-aware mechanisms. Naturally, both mechanisms can also be used in conjunction. Nevertheless, the deployment of locality-aware mechanisms depends on its adoption by a large user base. Our results suggest that caching is a viable alternative for ISPs while locality-aware mechanisms are not adopted.

Overall, our study presents a comprehensive analysis of BitTorrent traffic relevant to cache design and it performs a wide evaluation of the effectiveness of several different designs for caching BitTorrent traffic. Nevertheless, our findings may be extended in various directions which we left out of our scope. First, given that our results show that a marked temporal locality in BitTorrent traffic chiefly influences cache effectiveness, future research should extend our evaluation exploring optimizations in a LRU-based cache. Such optimizations can focus on the development of an efficient cache implementation, analyzing, for example, how variations in the number of cache users impacts on replacement policy effectiveness. Furthermore, future research should explore a hybrid approach that combines both caching and locality-aware mechanisms. Our results show that the traffic reduction achieved by each one of these strategies leaves a margin for applying further reduction techniques. In this direction, future work should analyze if the traffic reductions achieved by each mechanisms are indeed complementary, and how each mechanism impacts on the effectiveness of the other.

References

1. Abrams M, Standridge CR, Abdulla G, Fox EA, Williams S (1996) Removal policies in network caches for world-wide web documents. In: SIGCOMM '96: Conference proceedings on Applications, technologies, architectures, and protocols for computer communications. ACM, New York, pp 293–305
2. Ager B, Kim J, Schneider F, Feldmann A (2010) Revisiting cacheability in times of user generated content. In: Proceedings of the 13th IEEE global internet, symposium
3. Ager B, Schneider F, Feldmann A (2009) Cacheability of bulk content for isps. In: SIGCOMM '09: poster session of the 2009 conference on applications, technologies, architectures, and protocols for, computer communications
4. Aggarwal C, Wolf JL, Yu PS (1999) Caching on the world wide web. IEEE Trans Knowl Data Eng 11(1):94–107
5. Andrade N, Santos-Neto E, Brasileiro F, Ripeanu M (2009) Resource demand and supply in bittorrent content-sharing communities. Comput Netw 53(4):515–527
6. Arlitt MF, Williamson CL (1996) Web server workload characterization: the search for invariants. Performance evaluation review of the ACM special interest group on performance evaluation (ACM-SIGMETRICS) 24(1):126–137
7. Arlitt MF, Williamson CL (1997) Internet web servers: workload characterization and performance implications. IEEE/ACM Trans Netw 5(5):631–645
8. Badam A, Park K, Pai VS, Peterson LL (2009) Hashcache: cache storage for the next billion. In: Proceedings of the 6th USENIX symposium on networked systems design and implementation, NSDI'09. USENIX Association, Berkeley, CA, USA, pp 123–136
9. Basher N, Mahanti A, Mahanti A, Williamson C, Arlitt M (2008) A comparative analysis of web and peer-to-peer traffic. In: Proceeding of the 17th international conference on world wide web. ACM, New York, pp 287–296
10. Bellissimo A, Levine BN, Shenoy P (2004) Exploring the use of bit torrent as the basis for a large trace repository. Technical Report 04–41, University of Massachussets
11. Bindal R, Cao P, Chan W, Medved J, Suwala G, Bates T, Zhang A (2006) Improving traffic locality in bittorrent via biased neighbor selection. In: Proceedings of the 26th IEEE international conference on distributed computing systems, ICDCS '06. IEEE Computer Society, Washington, pp 66–66
12. Breslau L, Cao P, Fan L, Phillips G, Shenker S (1999) Web caching and zipf-like distributions: evidence and implications. In: Proceedings of the 1999 conference on computer communications. New York, NT, USA, pp 126–134
13. Choffnes DR, Bustamante FE (2008) Taming the torrent: a practical approach to reducing cross-isp traffic in peer-to-peer systems. SIGCOMM Comput Commun Rev 38:363–374
14. Clevenot-Perronnin F, Nain P (2005) Stochastic fluid model for p2p caching evaluation. In: Proceedings of the 10th international workshop on web content caching and distribution. IEEE Computer Society, Washington, pp 104–111
15. Cunha C, Bestavros A, Crovella M (1995) Characteristics of www client-based traces. Tech. Rep. TR-95-010, Boston University, Boston
16. Dán G, Carlsson N (2010) Power-law revisited: large scale measurement study of p2p content popularity. In: Proceedings of the 9th international conference on peer-to-peer systems, IPTPS'10. USENIX Association, Berkeley, p 12
17. Faber AM, Gupta M, Viecco CH (2006) Revisiting web server workload invariants in the context of scientific web sites. In: SC '06: Proceedings of the 2006 ACM/IEEE conference on supercomputing. ACM Press, New York, p 110
18. Feitelson D (2011) Workload modeling for computer systems performance evaluation. Tech. rep.
19. Garetto M, Figueiredo D, Gaeta R, Sereno M (2007) A modeling framework to understand the tussle between isps and peer-to-peer file-sharing users. Perform Eval 64(9–12):819–837
20. Gummadi KP, Dunn RJ, Saroiu S, Gribble SD, Levy HM, Zahorjan J (2003) Measurement, modeling, and analysis of a peer-to-peer file-sharing workload. Operating systems review of the ACM special interest group on operating systems (ACM-SIGOPS) 37(5):314–329
21. Guo L, Chen S, Xiao Z, Tan E, Ding X, Zhang X (2005) Measurements, analysis, and modeling of bittorrent-like systems. In: IMC '05: Proceedings of the 5th ACM SIGCOMM conference on internet measurement. USENIX Association, Berkeley, CA, USA, pp 35–48
22. Halinger G, Hartleb F (2011) Content delivery and caching from a network providers perspective. Comput Netw 55(18):3991–4006
23. Hefeeda M, Hsu C, Mokhtarian K (2011) Design and evaluation of a proxy cache for peer to peer traffic. IEEE Trans Comput (99)1
24. Hefeeda M, Saleh O (2008) Traffic modeling and proportional partial caching for peer-to-peer systems. IEEE/ACM Trans Netw 16(6):1447–1460
25. Iliofotou M, Siganos G, Yang X, Rodriguez P (2010) Comparing bittorrent clients in the wild: the case of download speed. In: Proceedings of the 9th international conference on peer-to-peer systems, IPTPS'10. USENIX association, Berkeley, p 11
26. Karagiannis T, Rodriguez P, Papagiannaki K (2005) Should internet service providers fear peer-assisted content distribution? In: Proceedings of the 5th ACM SIGCOMM conference on internet measurement. USENIX Association, Berkeley, p 6

27. Kendall MG (1990) Rank correlation methods, vol. 3. Griffin
28. Lehrieder F, Dan G, Hobfeld T, Oechsner S, Singeorzan V (2010) The impact of caching on bittorrent-like peer-to-peer systems. In: Proceedings of the IEEE international conference on peer-to-peer computing
29. Liao WC, Papadopoulos F, Psounis K (2007) Performance analysis of bittorrent-like systems with heterogeneous users. Perform Eval 64(9–12):876–891
30. Lin M, Lui J, Chiu DM (2010) An isp-friendly file distribution protocol: analysis, design, and implementation. IEEE Trans Parallel Distrib Syst 21(9):1317–1329
31. Liu B, Cui Y, Lu Y, Xue Y (2009) Locality-awareness in bittorrent-like p2p applications. IEEE Trans Multimed 11(3):361–371
32. Mahanti A, Eager D, Williamson C (2000) Temporal locality and its impact on web proxy cache performance. Perform Eval 42(2–3):187–203
33. Mahjur A, Jahangir A, Gholamipour A (2005) On the performance of trace locality of reference. Perform Eval 60(1–4):51–72
34. Maier G, Feldmann A, Paxson V, Allman M (2009) On dominant characteristics of residential broadband internet traffic. In: IMC '09: Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference, pp 90–102
35. Markatos EP (1996) Main memory caching of web documents. Comput Netw 28(7–11):893–905
36. Neglia G, Reina G, Zhang H, Towsley D, Venkataramani A, Danaher J (2007) Availability in bittorrent systems. In: INFOCOM 2007. 26th IEEE international conference on computer communications. IEEE, pp 2216–2224
37. OverCache P2P: http://www.oversi.com. Online December 2011
38. PeerApp UltraBand: http://www.peerapp.com. Online December 2011
39. Podlipnig S, Böszörmenyi L (2003) A survey of web cache replacement strategies. ACM Comput Surv 4:374–398
40. Pouwelse JA, Garbacki P, Epema DHJ, Sips HJ (2005) The bittorrent p2p file-sharing system: measurements and analysis. In: IPTPS'05: 4th international workshop on peer-to-peer systems
41. Psounis K, Zhu A, Prabhakar B, Motwani R (2004) Modeling correlations in web traces and implications for designing replacement policies. Comput Netw 45(4):379–398
42. Rabinovich M, Spatschek O (2002) Web caching and replication. Addison-Wesley Longman Publishing Co., Inc., Boston
43. Ruan B, Xiong W, Chen H, Ye D (2009) Improving locality of bittorrent with isp cooperation. In: Proceedings of the 2009 international conference on electronic computer technology. IEEE Computer Society, Washington, pp 443–447
44. Saroiu S, Gummadi KP, Dunn RJ, Gribble SD, Levy HM (2002) An analysis of internet content delivery systems. In: OSDI '02: proceedings of the 5th symposium on operating systems design and implementation. ACM, New York, pp 315–327
45. Saroiu S, Gummadi KP, Dunn RJ, Gribble SD, Levy HM (2002) An analysis of internet content delivery systems. Operating systems review of the ACM special interest group on operating systems (ACM-SIGOPS) 36(SI):315–327
46. Sherman A, Nieh J, Stein C (2009) Fairtorrent: bringing fairness to peer-to-peer systems. In: Proceedings of the 5th international conference on emerging networking experiments and technologies, CoNEXT '09. ACM, New York, pp 133–144
47. Tian C, Liu X, Jiang H, Liu W, Wang Y (2008) Improving bittorrent traffic performance by exploiting geographic locality. In: Proceedings of the global communications conference GLOBECOM, IEEE, pp 2489–2493
48. Wang J, Wang C, Yang J, An C (2011) A study on key strategies in p2p file sharing systems and isps p2p traffic management. Peer-to-Peer Networking and Applications, pp 1–10
49. Wessels D (2001) Web caching. O'Reilly & Associates, Inc., Sebastopol
50. Wierzbicki A, Leibowitz N, Ripeanu M, Wozniak R (2004) Cache replacement policies for p2p file sharing protocols. Eur Trans Telecommun 15(6):559–569
51. Wierzbicki A, Leibowitz N, Ripeanu M, Wozniak R (2004) Cache replacement policies revisited: the case of p2p traffic. In: CCGRID '04: proceedings of the 2004 IEEE international symposium on cluster computing and the grid. IEEE Computer Society, Washington, pp 182–189
52. Williams A, Arlitt M, Williamson C, Barker K (2005) Web workload characterization: ten years later. In: Tang X, Xu J, Chanson ST (eds) Web content delivery, chap. 1, pp 3–21. Springer, New York
53. Wojciechowski M, Capotă M, Pouwelse J, Iosup A (2010) Btworld: towards observing the global bittorrent file-sharing network. In: Proceedings of the 19th ACM international symposium on high performance distributed computing, HPDC '10. ACM, New York, pp 581–588
54. Wojciechowski M, Pouwelse J (2011) It's not that simple: an empirical study on the influence of locality on download speed in bittorrent. Tech. Rep. PDS-2011-008, Delft University of Technology
55. Xie H, Yang YR, Krishnamurthy A, Liu YG, Silberschatz A (2008) P4p: provider portal for applications. SIGCOMM Comput Commun Rev 38:351–362
56. Zhang B, Iosup A, Pouwelse J, Epema D (2010) The peer-to-peer trace archive: design and comparative trace analysis. In: Proceedings of the ACM CoNEXT student workshop, CoNEXT '10 student workshop, pp 21:1–21:2. ACM, New York
57. Zhang C, Dhungel P, Wu D, Ross K (2011) Unraveling the bittorrent ecosystem. IEEE Trans Parallel distrib Syst (99)1