

Cache Replacement Policies For P2P File Sharing Protocols

Adam Wierzbicki, PhD*
Polish-Japanese Institute of
Information Technology
adamw@icm.edu.pl
*ul. Orzycka 8 m. 37
02-695 Warsaw, Poland

Nathaniel Leibowitz
Tangium Networks
natan@expand.com

Matei Ripeanu
University of Chicago
matei@cs.uchicago.edu

Rafał Woźniak
Polish-Japanese Institute of
Information Technology

Abstract

Peer-to-peer (P2P) file-sharing applications generate a large part of today's Internet traffic. The large volume of this traffic (thus high potential caching benefits) and the large cache sizes required (thus nontrivial costs associated with caching) only underline that efficient cache replacement policies are important in this case. File popularity in P2P file-sharing networks does not follow Zipf's law and several additional characteristics set the generated traffic apart from well-studied Web traffic. All these lead us to conduct a focused study of efficient cache management policies for P2P file-sharing traffic. This paper uses real-world traces and trace driven simulations to compare traditional cache replacement policies and new policies that exploit characteristics of the P2P file-sharing traffic generated by applications using FastTrack protocol.

1. Introduction

In recent years, the growth of the Web traffic carried by protocols in the HTTP family has encouraged the development of caching. Research in this field resulted in cache replacement policies well suited for the characteristics of Web traffic [6,11,14,15,1]. However the relatively small size of Web objects and decreasing cost of disk and memory make today's Web caches able to store most cacheable content. Therefore, Web caches rarely need to perform cache replacement operations. The hit rates, and thus performance impact, of Web caches is limited to values below 40% [5,6] by Web traffic patterns and by the limited cacheability of Web objects. The increased popularity of dynamically-created, non-cacheable content decreases the potential benefits of caching.

Today, the traffic volume generated by the most popular peer-to-peer (P2P) file-sharing protocol: FastTrack (used by file sharing

applications like Kazaa and iMesh and serving more than 4.5M concurrent users [13]) has increased to the extent that it may dominate the Internet traffic [3,7,8]. This makes caching efforts concentrating on Web objects less effective since they target a small part of the Internet traffic volume, whose cacheability is further reduced by dynamically generated content. As a consequence, there is growing interest in using caching mechanisms for the large volume of FastTrack traffic. An additional incentive lies in the fact that objects transported by file-sharing protocols are generally immutable and therefore always cacheable.

This paper aims to answer the following question: Does the experience on caching Web objects research translate directly to P2P file-sharing traffic, in particular to FastTrack traffic? The salient features of this traffic, mainly large file sizes, file size variability, and ability to split a single file download into tens of download sessions over extended durations, suggest that a cache for this traffic may behave differently than a pure ‘Web’ cache. Additionally, P2P file-sharing object popularity does not follow Zipf’s law that had been used by researchers to model and explain Web cache performance [11,7]. Yet, to date, there has been only limited

work on caching mechanisms of for P2P traffic [9].

We use traces collected in the real world [2] and trace-driven simulations to compare cache replacement policies that were successful for Web traffic with new policies specialized for FastTrack traffic. We focus on technical aspects and ignore legal issues that could cause concerns for cache deployments. Note, however, that the same legal issues have generated concern for Web caching [16], although the issue of intellectual property rights was never as significant as in the case of file sharing.

This paper is organized as follows. The next two sections briefly present the FastTrack protocol and the characteristics of the traces used. Section 4 presents the main questions about cache operation that the paper attempts to answer and describes the cache replacement policies studied. The simulator design and simulation results are described in Sections 5 while Section 6 summarizes and concludes this paper.

2. FastTrack Protocol

The Kazaa network, the most popular application using the FastTrack protocol, consists of two entity types: a Kazaa *user agent* which downloads and shares files, and a Kazaa *supernode* which serves as a referral service to

where the requested files can be found. File identification is based on content: each file is assigned a unique identifier based on the actual content of the file. This enables a universal file identification scheme that is independent of advertised file names that may change from user to user (however different versions of the same content, e.g., music files with different quality or with slightly different duration, might still be treated as distinct). Kazaa user agents establish a channel with their local supernode over which they inform the supernode of the files they share and issue search requests. The purpose of this *control* channel is to enable actual file transfers carried out over *data* channels established directly between two Kazaa user agents. Since file transfers take place solely over the data channels, the control channel, while interesting in its own, has little relevance to the topic of this paper, hence we omit its details. As a summary, we outline the various steps of a typical file transfer sequence:

1. When Kazaa agent A is started, it establishes a persistent control channel with its supernode.
2. Assume user at agent A is interested in Mozart's 40th Symphony. Agent A will send a search request to its supernode over the control channel.

3. The supernode uses its local database and collaborates with other supernodes to compile a list of other agents that store the file, and sends this list back to agent A.
4. Agent A establishes data channels to some of the agents specified in the reply, and requests different file-ranges from each. The ranges might overlap, and together span the whole file. It is common for an agent to prematurely abort a connection when it is able to receive an equivalent range from a better source.
5. Once user agent A obtains the complete file, it may announce its supernode that it shares a new file.

Splitting a single file download into multiple, independent file-range downloads is a central feature of the FastTrack protocol, and requires a few new terms. As in [2], we use *download session* or simply *session* to describe a single TCP session between two agents, over which a range of a file (none, part, or all of the file) is transferred. We use *download cycle* for the logical transfer of a whole file, which might consist of tens of sessions and might extend over hours or even days.

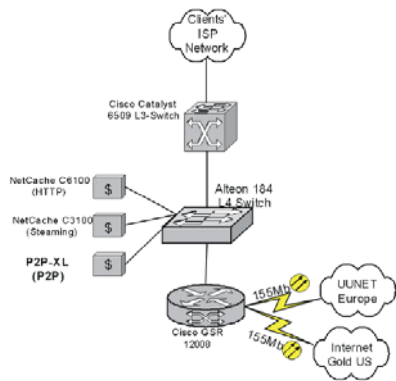


Figure 1. FastTrack cache installation used for trace gathering.

3. Trace Collection and Traffic Statistics

The FastTrack traces we employ have been obtained from a P2P proxy cache installed at a large Israeli ISP. This installation has been active for about a year, and handles on average 2000 concurrent download sessions generating about 80 Mbs of traffic. A server is installed at the border between the local user base of the ISP and the Internet cloud (Figure 1). Based on the HTTP headers used to initiate each download, a Layer-4 switch transparently redirects all Kazaa traffic to this server. Thus, the server is able to intercept all downloads performed by local users from the external Internet. We note that in the data we analyze we focus on downloads performed by local users and completely ignore downloads performed by outside users from local file providers (in other words we are only interested in incoming traffic).

The cache used in the installation had a size of 200GB and 1GB of main memory. The traces we employ cover a 26-day period from 1/25/2003 to 2/20/2003. They consist of about 4.2 million download sessions over which 12.2 TB of data were transferred.

A previous analysis [2] of other traces from the same source has shown a high reference locality (the ideal byte hit rate of a cache was estimated at 67%) and has estimated that a cache size of about 200GB should be sufficient to achieve a byte hit rate of about 60%. Further characteristics of the traffic observed in this specific installation are detailed in [3]. Subsequent analysis of the behavior of similar P2P proxy caches installed at two other ISPs revealed identical behavior, indicating that the traffic we are analyzing is a representative sample of FastTrack traffic.

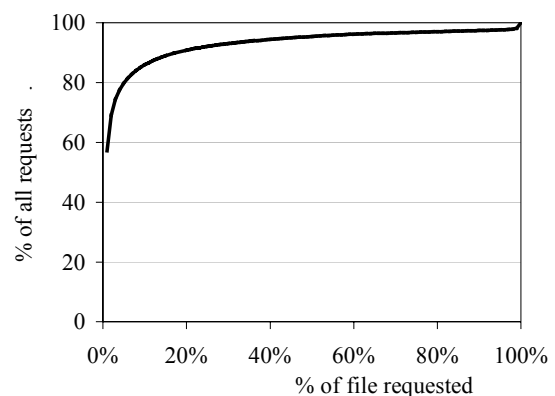


Figure 2. Cumulative distribution function (CDF) for the percentage of the file requested in each request.

Download sessions are generally short when compared with the size of the entire file. Figure 2 presents the cumulative distribution function for the percentage of the file requested in each download session: 80% of all requests ask for 10% of the file or less. Additionally, the start of requested ranges is uniformly distributed over the entire file.

Sessions in each trace are ordered by their termination time. For each download session we use: the unique ID for the file downloaded, the range requested in the session, the size of the entire file, and the actual number of bytes that were transferred during the session.

3.1. Does Zipf's law apply to P2P traffic?

Zipf's law is a frequently cited and studied characteristic of Web traffic. This law describes the frequency of occurrence of objects in a larger set (object *popularity*). The original observation, made by Zipf in 1965, concerned the frequency of the use of words in natural language [10]. For Web traffic, it has been claimed by many researchers that the popularity of files requested by clients follows Zipf's law [11]. From this observation, conclusions about other observable Web traffic characteristics, the performance of Web caching systems and replacement algorithms have been drawn.

Zipf's law relates the popularity of an object to its rank in a ranking of popularity as follows:

$$f(r) = c r^{-\alpha}$$

where r is the object rank, when the objects are sorted in decreasing order of their popularity $f(r)$, and c and α are constants.

In order to illustrate Zipf's law, the results of a ranking can be plotted with the rank on the x-axis, and the popularity on the y-axis. By plotting the observed values on a log-log scale, the result should be a straight line if the observations concur with Zipf's law. In order to investigate whether FastTrack traffic conforms with Zipf's law, we have made an observation of the popularity of all distinct files in our traces.

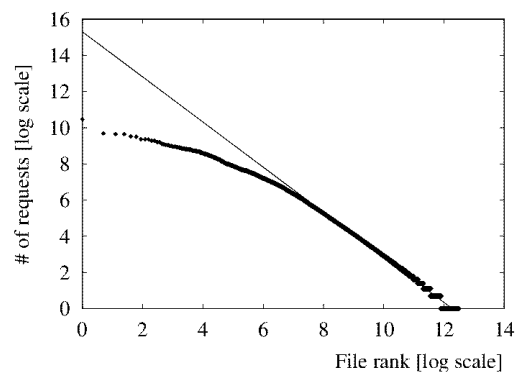


Figure 3. Most popular files ranked by number of requests

Figure 3 shows the result of a ranking using the number of requests. While we have not performed a rigorous statistical test, the result of fitting a linear trend on the data indicates that the observations do not come close to a straight line. From this we conclude that FastTrack object

popularity may not follow Zipf's law. This finding is supported by the results obtained in [7]. The observed traffic is less concentrated on popular files. For a ranking that used the number of downloaded bytes, the deviation from Zipf's law was even more pronounced.

3.2. Rate of Change

Understanding the dynamics of the set of most popular files is important both from a caching perspective as well as for understanding general usage patterns of a file-sharing system. Consider, for instance, compiling the list of 100 most popular files every day. How would this list change over time? Would it be possible to identify files that are always on this list (all time favorites), or would the list change frequently (the equivalent of one-day stars)?

To investigate this question, we determine from our traces the N most popular files during consecutive observation periods, where $N \in \{4, 50, 400\}$. The observation periods are approximately 24-hour intervals. The popularity of a file is measured by the number of download cycles of the file.

The first part of our analysis investigates how much the list of N most popular files changes from one observation period to another. Let x_t be the set of files that were on both N -most-popular-lists at time $t-1$ and t . We compute the percentage

of the popular files that have persisted between the two observations as: $100 * x_t / N$ and we plot this value in Figure 4 for different values of N .

For $N=4$, the percentage of recurrently popular files is almost always 50%, which means that the same two files constantly ranked in the 'Top 4' lists. Based on accumulated experience with Kazaa, we assume these files are most likely Kazaa software installation packages, which circulate frequently in the network. For higher values of N , the situation changes. The percentage of recurrently popular files is stable at about 30%, slightly decreasing for large N . This suggests that caching can be effective for Kazaa traffic.

We now look at the set of files with long-term popularity: for each new observation period, we intersect the list corresponding to that period with the intersection of the lists from all previous observation periods. In Figure 5 we plot the percentage of the files in the first list that remained in this intersection after t observation periods. The percentage of files that are popular in all observation periods stabilizes at about 15%. This suggests that there are indeed a number of "all-time favorites" during our observation.

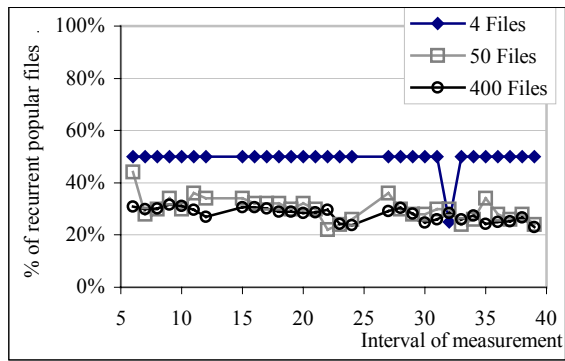


Figure 4: Ratio of the popular files that remain popular during consecutive time periods.

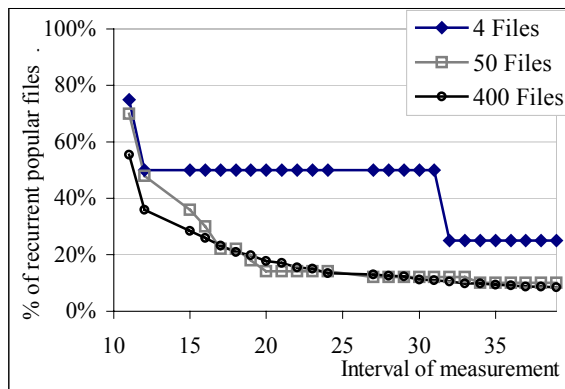


Figure 5: Ratio of the popular files set that remains stable when compared with a base period.

The number of files that remain popular in the next observation period is larger than the number of files that are popular in all observation periods. This suggests that the set of popular files changes slowly over time, since only about half of these

files are present in all observation periods. A longer experimentation period is required to determine how persistent is this group and further quantify their rate of change over months.

In summary, the two previous experiments show that 15% of the highly popular files remain popular throughout the experiment, while the rest are popular shorter time intervals. This indicates that the set of popular files is composed of two subsets: a set of persistently popular files and a set of transiently popular files whose popularity is relatively short lived.

4. Peer-to-Peer Cache Operation and Replacement Policies

This section presents the main aspects of P2P cache operation that impact on performance as well as the cache replacement policies we investigate. Apart from the question: ‘What is the best replacement policy?’ we study three different issues brought by P2P caching that were not relevant for Web caching. We present these issues first then we study the effectiveness of cache replacement policies.

4.1. When Does a Hit Occur?

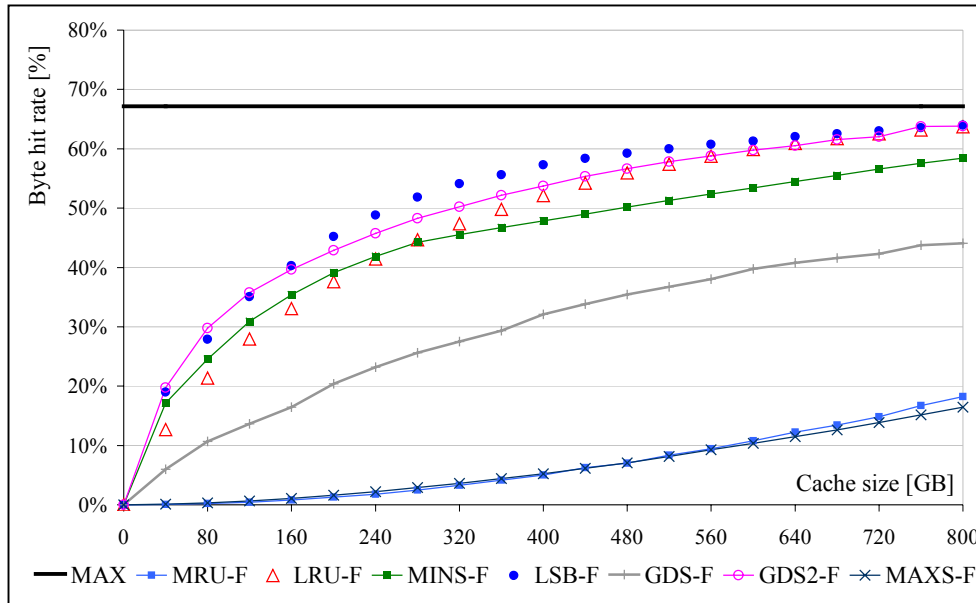


Figure 6: Comparison of file replacement strategies for full caching

In the case of FastTrack traffic, deciding when a cache-hit occurred is no longer as obvious as when dealing with regular Web objects: the request is made for a range of a file and the cache may contain ranges that overlap with the requested range.

To satisfy the request completely, the cache should contain the entire requested range. We shall refer to this scenario as ‘full P2P caching’. In this case the cache is both *transparent* (no changes are required to the download protocol) and *passive* (the cache does not originate download requests itself). In this case however, requests that are only partially cached will not be served. To address this inefficiency two alternatives are possible. Firstly, in a scenario we refer as ‘partial P2P caching’, the cache can remain passive but give up transparency: it would

modify the current protocol and negotiate with the client the download of sub-ranges of the requested range. Alternatively the cache can become *active* and issue a download request itself for the missing sub-ranges. In this case the cache acts as a FastTrack client itself. For brevity, in the rest of this paper, we use “partial/full P2P caching” as shortcuts for “caching that serves partial/full hits”.

4.2. Should the Cache Ignore User Aborts?

A second question is whether a cache should ignore user aborts in the case of a cache miss. A user abort is issued when a user agent has found a better download source or when the user simply cancels the download (possibly after evaluating the content based on ranges already downloaded).

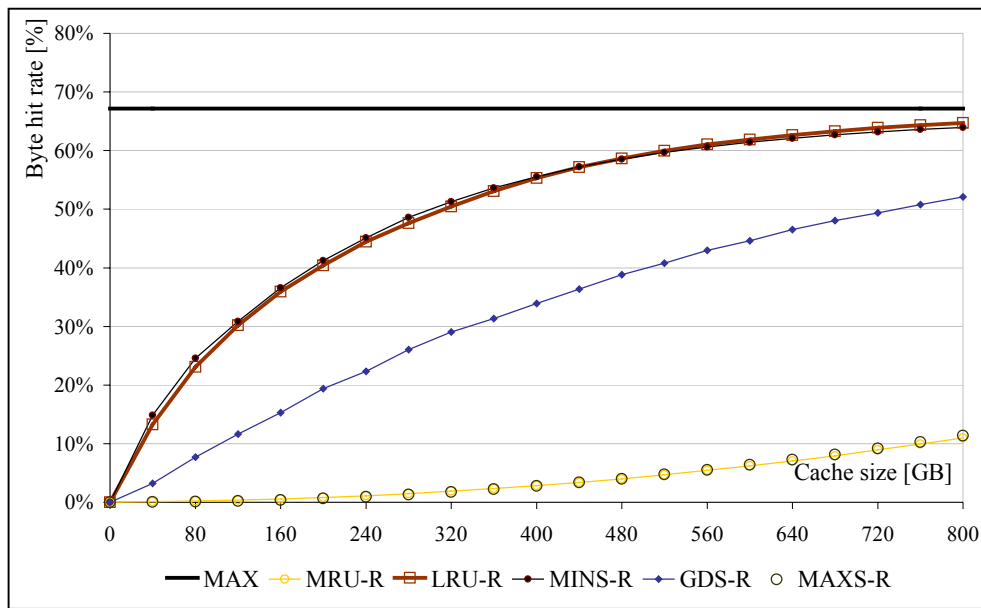


Figure 7: Comparison of range replacement strategies for full caching

In this case a cache that is serving a miss will stop receiving the information, since it is clear that it will not be needed. On the other hand, the cache could keep downloading to anticipate future user requests. This behavior is similar to prefetching. Since range requests of FastTrack user agents frequently overlap, a cache ignoring aborts could obtain a better byte hit rate.

Since the main goal of caching is reducing generated network traffic, deciding on how to handle user aborts should depend on the tradeoff between the potential increase in the byte hit rate resulting from more caching and the increased download traffic to fill the cache.

4.3. Should a Cache Replace File Ranges?

A cache replacement policy can be viewed as a specialized instance of the well-known

knapsack problem. The set of files cached has to maximize a certain utility function while satisfying a size constraint. In the knapsack problem, it is often easier to store many objects if the sizes of all objects are small relative to the knapsack size. A P2P cache that stores file ranges might therefore benefit from the replacement of individual ranges instead of whole files, because ranges are smaller and offer more flexibility to the replacement policy.

This assumption would fail if the reference locality of FastTrack requests would always focus on entire files instead of a range of the file. If FastTrack users always (or only frequently) download entire files or large portions of a file, then it would not make sense for the cache to replace individual file ranges. To verify this initial objection to the replacement of ranges, we

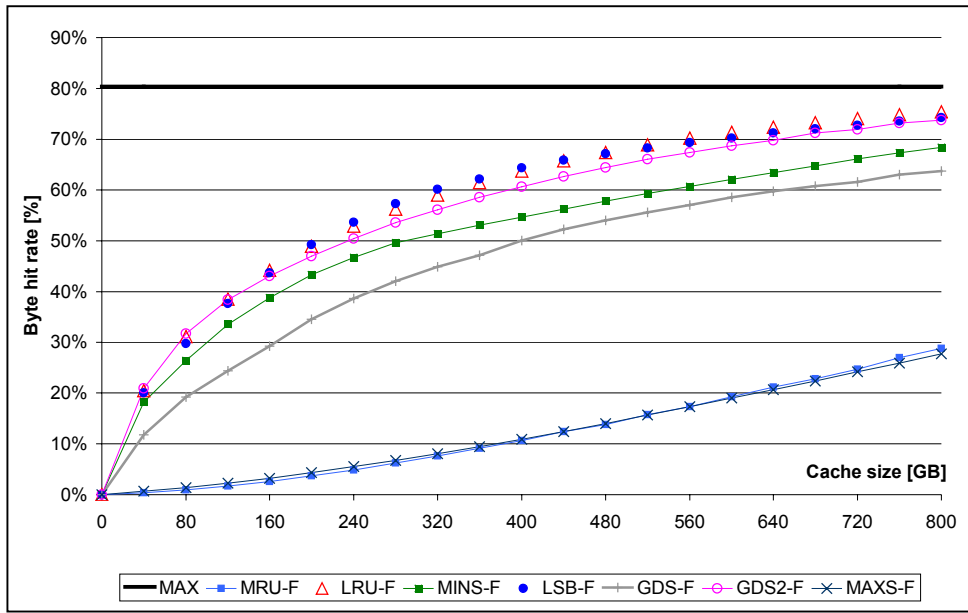


Figure 8: Comparison of file replacement strategies for partial caching.

have calculated the distribution of request sizes relative to the entire file size. To obtain this statistic, each request size was divided by the size of the entire file, and the resulting values were plotted as a cumulative distribution function (Figure 2). This statistic shows that the requested range size is not uniform: although small ranges form the bulk of all downloads, some requests include as much as half of the file. Note that, taking user aborts into consideration does lead to an increased number of small range requests but does not significantly change the plot presented in Figure 2. On the other side, the distribution for the beginning of the requested range is evenly distributed over the whole file. We conclude that, generally, range requests are short and ask for any portion of the file. Additionally, user aborts tend to increase the number of small requests.

When we refer to the granularity at which the cache operates, we use the term *file-based* replacement policy when the cache operates at a file granularity (as for Web objects) and *range-based* replacement policy when the cache operates at a file-range granularity. The initial assessment based on trace statistics of range request size and position seems to support the assumption that range-based policies are more effective. One goal of this paper is to verify this assumption.

One cost of range-based policies is a larger memory overhead to manage range metadata. However, this cost appears manageable for the real world deployments we have encountered and is dependent on cache and range metadata implementation specifics.

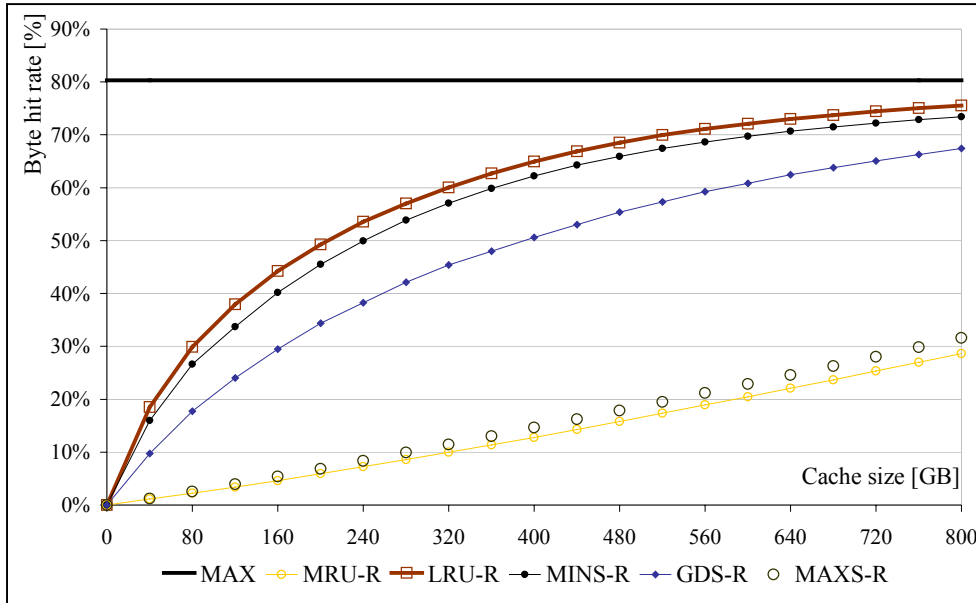


Figure 9: Comparison of range replacement strategies for partial caching.

Most cache replacement policies presented in the next two subsections can be used both for files and ranges.

4.4. Basic Replacement Policies

This section presents some of the traditional cache replacement policies that employed for Web caching [6, 5]. A cache *replacement policy* can be generally defined by a comparison rule that compares two cached items (two files for a file-based policy or two ranges for a range-based policy). Once such a rule is known, all objects in the cache can be sorted, and this is sufficient to define a replacement policy: the cache will remove the object of lowest value with respect to the given comparison rule.

Each cached item (a file or a range) has several attributes, such as access time (the last time when the object was accessed) or size. These

attributes are used by the replacement policies we present below.

The simplest replacement policies are easily expressed using comparison rules. Least Recently Used (LRU) and Minimum Size (MINS) are two such policies; their binary negations, Most Recently Used (MRU) and Maximum Size (MAXS) will also be included in the evaluation. Greedy-Dual Size (GDS [1]) replacement policy combines multiple characteristics of a cached object: its access history, file size, and freshness of the last access.

4.5. Specialized Replacement Policies

The basic policies described in the previous section do not exploit all the information available to a FastTrack cache. For example, a file stored in a cache may consist of several ranges with gaps in between and an important

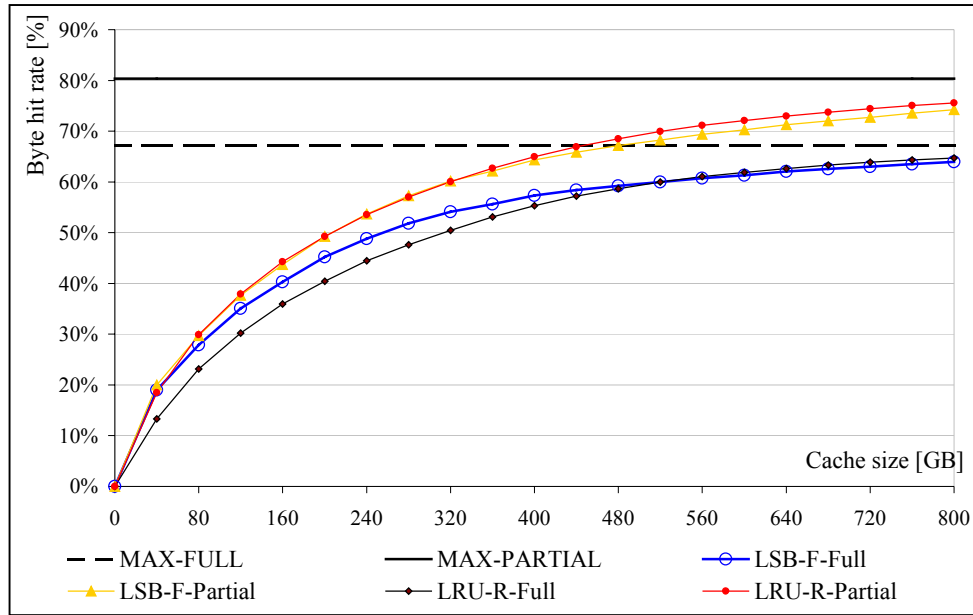


Figure 10: Best replacement strategies for full and partial caching.

piece of information is how much of the total file is stored in the cache. We maintain the following specialized attributes for objects stored in a FastTrack cache:

1. *maximum size*: the maximum size of the object - for files, it can be larger than the size of the object in the cache,
2. *transmitted bytes*: the amount of information that has been sent to users from this object. This can take into account user aborts: when an object is used to serve a hit, the number of bytes downloaded before the user sent an abort is added to the transmitted bytes of the object.
3. *scaled access time*: a number that takes into account the updated part of the object. When the object is accessed, the difference between the present time and the object's previous

access time is weighted by the portion of the object that has been requested and this number is added to the scaled access time. If requests are always made for entire objects, such as in Web caching, this policy is equivalent to LRU.

The first specialized policy we present is a file-based policy that takes into account the proportion of the file stored in the cache. If the cache stores almost the whole file, then it has the best chance of serving a range request for that file. We name this policy Minimum Relative Size (MINRS): it removes from the cache the files that have the smallest cached content relative to the entire file size. (For range-based policies this is the only specialized policy we evaluate).

Another possibility is to take into account how much data was served from a cached object. For

Web caching, this is the equivalent of a frequency-based policy (such as LFU). However, objects in a FastTrack cache have to take into account user aborts and can change their sizes when new ranges are added. The policies of Least Sent Bytes (LSB) and Least Relative Sent Bytes (LRSB) use the transmitted bytes of an object. This attribute is increased whenever the object is used to serve a hit, by the amount of downloaded bytes before the user sent an abort. LRSB divides that amount by the maximum file size.

The observation that P2P traffic does not follow Zipf's law (Section 3.1) may indicate that frequency-based policies would not perform well for this traffic. Breslau et al. [11] use a simple, Zipf-based model and an independence assumption for Web traffic to argue that frequency-based policies perform best for large Web caches. Our results show that P2P traffic is less concentrated on popular objects than a Zipf model.

5. Comparison of Replacement Policies

We use trace driven simulations to compare various cache replacement strategies. We use CacheSim [4, 12], a Java-based simulation and traffic statistics package that has been used to study HTTP traffic and cache filtering. We

extended CacheSim with the capability to process FastTrack traces and to simulate file- and range-based policies. CacheSim code is released under the GNU public license and is available from the authors on request.

The results of the comparison of replacement policies are presented in Figures 6-10. Figures 6 and 7 present byte hit rates of various policies for full caching, while Figures 8 and 9 present corresponding results for partial caching. Results for both file- (suffix '-F' in the plots) and range-granularity (suffix '-R') for replacement policies are presented. Figure 10 presents together the performance of the best policies for partial and full P2P caching. All figures also show the ideal hit rates achievable for an infinite cache for the trace used.

These figures present byte-hit rates for a warmed-up cache. Traces have been divided into two parts: roughly the first third of the trace (4.7 TB of generated traffic) is used to warm-up the cache while the rest is used to evaluate replacement policies on a warmed-up cache.

Results for range-based full P2P caching show good performance for LRU. On the other hand, the performance of Minimum Size (MINS) is surprisingly good, while Maximum Size (MAXS) performs poorly. Consider how the cache determines that a hit occurred and the distribution

of beginnings of range requests for a possible explanation. A cache needs to have the entire requested range in order to serve the request. However, range requests are evenly distributed across the entire file. Therefore, cache entries that are large have a better chance of serving a request. The policy that removes large cache entries performs poorly, while a policy that removes small cache entries performs well.

The poor performance of the Greedy-Dual Size (GDS) policy can be explained similarly. GDS prefers to remove larger cache entries, and pays the same performance penalty as Maximum Size. We have simulated the GDS policy with various parameter values without observing a significant impact on the results, which further supports the observation that the policy is unsuitable for FastTrack traffic.

Minimum Relative Size (MINRS) does not perform as well as MINS; the reason could be that this policy discriminates against the inclusion of large objects. For large objects, new ranges are very small relative to entire file size and will be first removed by MINRS.

For full caching, the best performance in terms of byte-hit rate was obtained for Least Sent Bytes. This policy has the advantage that it considers available information about user aborts. Its good performance indicates locality in user

aborts: perhaps some large files on slow links are aborted more frequently than other files. However, this issue requires more detailed investigation. The results indicating the superiority of LSB are in contrast with the results obtained in [9], where LRU, a frequency-based policy similar to LFU, and MINS were compared on a live P2P cache. In that study, LRU performed slightly better than the frequency-based policy on the outbound portion of the traffic, while the two policies performed similarly on the inbound traffic. The author describes several variants of the frequency-based policy used, and states that the best results were obtained by a policy that used the number of requests from unique clients as a measure of frequency. Thus, results obtained in [9] are not directly comparable with our results, since LSB uses the amount of sent bytes taking into account user aborts. However, not that the worse performance of simple frequency-based policies observed in [9] is consistent with the fact that FastTrack object popularity does not follow Zipf's law.

We have investigated a modified version of Greedy-Dual Size that uses information about the number of downloaded bytes. The resulting policy (called GDS2 in the figures) performed much better than GDS and was the best policy for very small caches sizes.

Range-based policies did not perform significantly better than file-based policies overall. However, for full caching some of the range-based policies (notably LRU) significantly outperform their file-based equivalents. Also, for partial caching the best policy (for large caches) was range-based LRU which slightly outperformed LSB.

Results for full P2P caching indicate a maximum byte hit rate of 67% (this is similar to the estimate in [2]). However, when compared to [2], the cache size necessary for a byte hit rate that is close to maximum is different. In our simulations, the size of 200GB (as proposed in [2]) leads to a lower byte hit rate. Only a cache that is twice larger (400GB) can obtain a byte hit rate about 15% smaller than the theoretical maximum. This difference is explained by an increase in sizes of transmitted files since the observations reported in [2].

A FastTrack cache able to serve partial hits (requests for ranges that overlap with the ranges available in the cache) can achieve a higher byte hit rate. This result indicates that the performance penalty for maintaining cache transparency is significant. The best policy for full caching was the file-based policy of LSB. For partial caching the difference between LSB and LRU was small. The LRU and MINS range-based policies

performed slightly better than their file-based variants for larger cache sizes.

We also simulated a cache operation that ignores user aborts. This approach however leads to a sharp increase in the number bytes downloaded by the cache. When the cache does not ignore user aborts, an infinite cache generates about 1.5 TB of traffic. When the cache ignores user aborts, byte hit rate grows to as much as 90% however the generated traffic grows to 30TB. We conclude that this form of prefetching is not desirable when the goal is traffic reduction.

6. Summary

The results presented in this paper are only a first step in exploring cache replacement policies for P2P file-sharing systems. The large volume of this traffic, thus high potential caching benefits, and the large cache sizes, thus nontrivial operational costs associated with large caches, only underline that efficient cache replacement policies are relevant for this type of traffic. Additionally, file-sharing traffic does not encounter the consistency problems that are now prevalent for Web traffic.

This study has focused on the FastTrack protocol. Before we summarize our findings, let us briefly discuss the relevance of this work to other file sharing protocols. Gnutella [19],

eDonkey [18] and BitTorrent [17] use downloads of file ranges, like the FastTrack protocol. For that reason, the issue of granularity of replacement policies and of full and partial caching may be of relevance to these protocols. In [20], the authors have used a technique similar to caching: a passive peer that does not originate requests, but caches all peer responses and serves the cached information to other peers. This approach is more useful for closed file sharing protocols. The authors of [21] report high performance improvements of their approach for the winny protocol, a file sharing application popular in Japan.

A possible explanation of the high hit rates observed in our study is effect of free-riders. Several studies [21,7,8]. have found that a majority of users of file sharing networks are free-riders that do not share files they have downloaded from other peers. For this reason, a file can be downloaded several times, leading to high reference locality. This phenomenon seems common to many file sharing applications and it is therefore possible that the performance of caching for other file sharing protocols could be high, like for the FastTrack protocol.

We have found that P2P traffic does not follow Zipf's law, and is less concentrated on popular objects. On the other hand, high byte-hit

rates can be achieved by caching of FastTrack objects, which implies that FastTrack traffic has a high reference locality. This is supported by the observation that the set of popular objects contains a subset of "all-time-favorites" that remain popular over long periods of time. Targeting this set with specialized cache replacement policies (for example, using statistical analysis) is a promising direction of future research.

Comparing the ideal byte-hit rate for full hits with the ideal byte-hit for partial hits shows that the latter approach could improve the byte-hit rate by about 13%. However, a cache can serve partial hits only at the expense of losing transparency. This motivates an extension of the FastTrack protocol with control messages that notify the requesting user agent that only parts of the requested range is served. The user agent could then initiate requests for the missing parts of the range and the cache would still operate transparently.

Range-based replacement policies do not perform significantly better than the best file replacement policies. However, range-based variants of basic policies performed better when associated with full P2P caching.

The best replacement policies for FastTrack traffic are yet to be discovered. The possibility of

specialization is large, and the potential of range-based policies that offer more flexibility is not yet fully exploited. The best policy proposed in this paper, which is a variant of a frequency-based policy that uses information about the number of downloaded bytes before a user abort, performs better than traditional policies used for Web caching, which shows the validity of the specialization approach.

7. References

- [1] L. Cherkasova. *Improving WWW Proxies Performance with Greedy-DualSize Frequency Caching Policy*, HP Laboratories Report No. HPL-98-69R1, April, 1998.
- [2] N. Leibowitz, A. Bergman, R. Ben-Shaul, and A. Shavit, *Are File Swapping Networks Cacheable? Characterizing P2P Traffic*, presented at 7th International Workshop on Web Content Caching and Distribution (WCW'03), Boulder, CO, 2002.
- [3] N. Leibowitz, M. Ripeanu, A. Wierzbicki, *Deconstructing the Kaza network*, in proceedings of 3rd IEEE Workshop on Internet Applications, (WIAPP'03), San Jose, California, June 2003.
- [4] M. Kurcewicz, A. Wierzbicki, W. Sylwestrzak, *Filtering algorithms for proxy caches*, Elsevier, Computer Networks and ISDN Systems, vol. 30, no. 22-23, 1998,
- [5] G. Barish, K. Obraczka, *World Wide Web Caching: Trends and Techniques*, IEEE Communications Magazine Internet Technology Series, May 2000.
- [6] J. Wang, *A Survey of Web Caching Schemes for the Internet*, ACM Computer Communication Review, vol. 25, no. 9, pp. 36-46, 1999
- [7] K. P. Gummadi, R. J. Dunn, S. Saroiu, S. D. Gribble, H. M. Levy, J. Zahorjan, *Measurement, Modeling, and Analysis of a Peer-to-Peer File-Sharing Workload*, Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP-19), October 2003.
- [8] S. Saroiu, K. P. Gummadi, R. J. Dunn, S. D. Gribble, and H. M. Levy, *An Analysis of Internet Content Delivery Systems*, Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI), December 2002
- [9] R. J. Dunn, *The Effectiveness of Caching on a Peer-to-Peer Workload*, Masters Thesis, University of Washington, December 2002
- [10] G. K. Zipf. *Human Behavior and the Principle of Least Effort*, New York, Hafner Pub. Co., 1965
- [11] L. Breslau, P. Cao, L. Fan, G. Phillips and S. Shenker. *Web Caching and Zipf-like Distributions: Evidence and Implications*, Proceedings of IEEE INFOCOM 99, Volume 1 pp.126-134, 1999
- [12] A. Wierzbicki, N. Leibowitz, M. Ripeanu, and R. Woźniak, *Cache Replacement Policies Revisited: The Case of P2P Traffic*, 4th Global and Peer-to-Peer Computing Workshop, April 2004, Chicago, IL.
- [13] <http://www.slyck.com>, January 2004.
- [14] P. Cao and S. Irani, *Cost-Aware WWW Proxy Caching Algorithms*, USENIX Symposium on

- Internet Technologies and Systems (USITS), Monterey, CA, pp. 193-206, December 1997.
- [15] M. Arlitt and C. Williamson, *Internet Web Servers: Workload Characterization and Performance Implications*, IEEE/ACM Transactions on Networking, Vol. 5, No. 5, pp. 631-645, October 1997.
- [16] Eric Schlachter, *Cache-22: Copying and storing Web pages is vital to the Internet's survival -- but is it legal?*, Intellectual Property Magazine, August 1996.
- [17] *BitTorrent protocol specification*,
bitconjurer.org/BitTorrent/protocol.html,
28.08.2004
- [18] A. Klimkin, **Unofficial* eDonkey Protocol Specification v0.6.2*,
mesh.dl.sourceforge.net/sourceforge/pdonkey/eDonkey-protocol-0.6.2.html, 31.08.2004
- [19] *The Gnutella protocol specification v0.4.*,
www9.limewire.com/developer/gnutella_protocol_0.4.pdf, 31.08.2004
- [20] A. Tagami, T. Hasegawa, T. Hasegawa, *Analysis and Application of Passive Peer Influence on Peer-to-Peer Inter-domain Traffic*, Proceedings of Fourth International Conference on Peer-to-Peer Computing (IEEE P2P'2004), Zurich, August, 2004
- [21] E. Adar and B. Huberman, *Free riding on Gnutella*, Xerox PARC Technical Report, 2000

8. Author Biographies

Adam Wierzbicki has received a PhD degree from Warsaw University of Technology in 2003. His research interests include P2P computing,

multimedia streaming, content delivery networks, and telecommunication networks design. He is currently an assistant professor at the Polish-Japanese Institute of Information Technology and works part-time as a programmer and analyst.

Matei Ripeanu (matei@cs.uchicago.edu) is a Ph.D. candidate in Computer Science at The University of Chicago. Matei is broadly interested in distributed computing with a focus on self-organization and decentralized control in large-scale Grid and peer-to-peer systems.

Nathaniel Leibowitz holds an MA in computer science from Tel-Aviv University (2000). In the computer industry, Nathaniel has investigated the characteristics of p2p traffic from its early stages as Napster clients in 2000 till its current status as the dominant portion of internet traffic. Nathaniel has contributed to the first papers proposing and analyzing the caching of p2p traffic and has lead R&D teams in Expand Networks and PeerAppliance developing caching algorithms for p2p traffic.

Rafał Woźniak is a student of the post-graduate programme of the Polish-Japanese Institute for Information Technology. His thesis concerns caching of FastTrack traffic. He is the administrator of the student's research laboratory.