Cache Replacement Policies Revisited: The Case of P2P Traffic

Adam Wierzbicki
Polish-Japanese Institute of Information Technology
adamw@icm.edu.pl

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 if not most of today's Internet traffic. The large volume of this traffic (thus the high potential benefits of caching) and the large cache sizes required (thus nontrivial costs associated with caching) only underline that efficient cache replacement policies are important in this case. P2P file-sharing traffic has several characteristics that distinguish it from well studied Web traffic and that require a focused study of efficient cache management policies. This paper uses trace driven simulations to compare traditional cache replacement policies with new policies that try to 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 development of caching. Research in this field resulted in caching policies well suited for the characteristics of Web traffic. However the relatively small size of Web objects and the decreasing cost of disk and memory make today's Web caches able to store most cacheable content and to rarely need 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. One current trend observed for Web traffic, the increases popularity of dynamically created, non-cacheable content decreases the potential benefits of caching.

Today, the traffic volume of the most popular P2P file sharing protocol, FastTrack (used by file sharing applications like Kazaa, iMesh and Grockster which, according to www.slyck.com, totaled more than 4.5M users in January 2004) 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 overall traffic volume, whose cacheability is further reduced by the presence of dynamically generated content. As a consequence, there is a 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 are 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 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. Yet to date there has been only limited work on cacheability of this traffic [9].

We use a trace-driven simulation approach and aim to evaluate the cache replacement policies that were successful for Web traffic, and to introduce new policies specialized for FastTrack traffic. We focus on the technical aspects without considering legal or security issues that can be relevant causes of concern for cache deployment. Note, however, that the same issues have generated concerns for Web caching, although the issue of intellectual property rights was never as significant as in the case of file sharing.

The paper is organized as follows: the next section presents the most relevant characteristics of the FastTrack protocol. Section 3 discusses the traces used. Section 4 presents the main questions about cache operation that the paper attempts to answer, and describes the replacement policies studied. Simulator design and simulation results are described in Sections 5 and 6.

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 regardless of their advertised file name 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). All Kazaa user agents establish a channel with their local supernode over which they inform the supernode of the files they share, and over which they send their search requests for files. This channel may be viewed as a *control* channel whose purpose is to enable actual file transfers carried out over *data* channels established directly between two Kazaa user agents (a downloader and uploader) and is therefore a pure peer-to-peer channel. Since the actual 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:

- When Kazaa agent A is started, it establishes a persistent control channel with its supernode.
- Assume user at agent A is interested in Mozart 40 symphony. Agent A will send a search request to its supernode over the control channel.
- 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.
- Agent A establishes a data channel to some of the agents specified in the reply, and requests different ranges of the file 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.
- Once user agent A obtains the complete file, it updates its supernode of the new file it shares.

The splitting of a single file download into multiple TCP sessions 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.

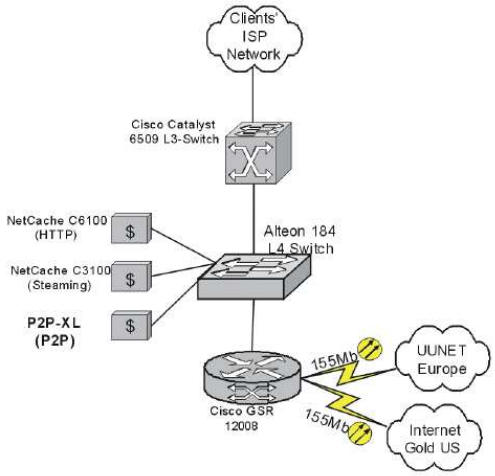

Figure 1. FastTrack cache installation used for trace gathering.

3. Trace Collection and Traffic Statistics

The FastTrack traces we analyze have been obtained from a P2P proxy cache installed at a large Israeli ISP. This installation has been active for above a year, and handles on average 2000 concurrent download sessions with about 80 Mbs of generated traffic. A server is installed at the border between the local user base of the ISP and the Internet cloud (see Figure 1). Based on destination port number for each TCP connection a Layer 4 switch 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 analyze cover a 26-day period from 1/25/2003 to 2/20/2003. They consist of about 4.2 million sessions over which 12.2 TB of data were transferred. The traces were divided into two parts: the first part covers the first period of 11 days during which 4,7 TB were transmitted, and the second part covers the remaining days during which 7,5 TB were transmitted. The first part was used to fill the cache, while the second part was used to evaluate replacement policies on a warmed-up cache.

The analysis of other traces from the same source [2] has shown a high reference locality: the ideal byte

hit rate of a cache was estimated at 67%. [2] also estimates 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.

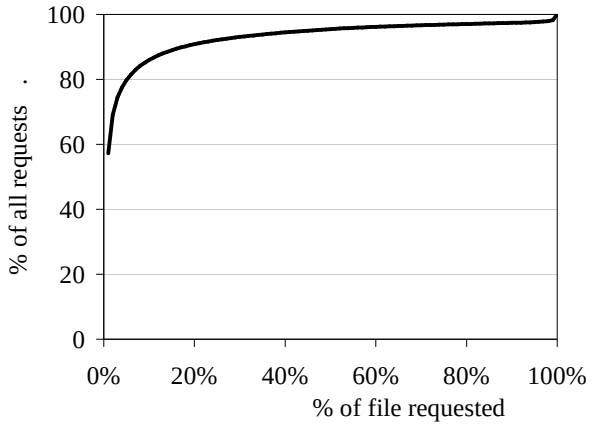

Figure 2. Cumulative distribution function (CDF) for the percentage of the file requested in each request. Observe that 80% of all requests ask for 10% of the file or less.

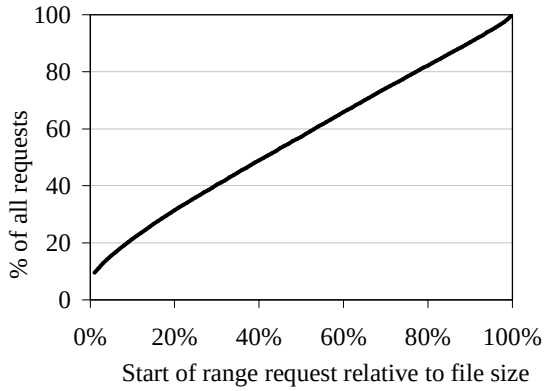

Figure 3. Cumulative distribution function (CDF) for the start of range requested relative to file size

Sessions each trace are ordered by their termination time. For each download session we use the following information:

- The unique file 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

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

This section presents the cache replacement policies we investigate as well as the main aspects of P2P cache operation that impact on performance. Apart from the question: "What is the best replacement policy?", this study aims to study three different issues on P2P caching that have not been relevant for Web caching. We present these issues first then we look into effectiveness of cache replacement policies.

4.1. When Does a Hit Occur?

In the case of FastTrack traffic, the answer to the simple question "When does a hit occur?" is no longer as obvious as for caching of regular Web objects. The request is made for a range of a file and the cache may contain various ranges that might 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* (there are no changes in the download protocol) and *passive* (the cache does not originate download requests). 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 download protocol to 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. In brief 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 generally is issued when the user agent has found a better source for the requested information. Therefore, in this case a cache that is serving a miss could 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 would be a form of prefetching. Since range requests of FastTrack user agents frequently overlap, ignoring aborts could increase the byte hit rate.

If the main goal of caching is reducing generated network traffic, then the a decision on how the cache should handle user aborts can be made once it is known how ignoring aborts affects the amount of data downloaded by the cache. In other words, would the increased byte hit rate that results from more caching compensate for the increased download traffic to 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 hypothesis would fail if the reference locality of FastTrack requests would always focus on entire files instead of a range of the file. If FastTrack user would always (or very 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 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). The statistic shows that the requested range size is not uniform: although small ranges form the bulk of all downloads, some requests might include as much as half of the file. The statistic of range request size was prepared in two variants. In one of the variants, user aborts are ignored and the size of the entire requested range is used; in the other variant, only the number of transmitted bytes is used. Figure 2 presents the second variant only, since the resulting difference is not significant. User aborts lead to an increased number of small range requests and an increased number of requests to download the whole file.

We have also investigated the distribution for the beginning of the requested range, relative to the file size (Figure 3), which is much more evenly distributed. It can be concluded 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 in caching of 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 seem to support the assumption that range-based policies could be more effective. One of the goals of this paper is to verify this hypothesis.

A range-based policy would require a larger memory overhead for range metadata, but in this paper we shall ignore this aspect since it is specific to an implementation of the cache and of the range metadata. Most cache replacement policies presented in the following two subsections can be used both for files and ranges.

4.4. Basic Replacement Policies

This section presents some of the cache replacement policies that have been studied for Web caching [5,6]. A 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 in an increasing order, and this is sufficient to apply 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) is a replacement policy (described in detail in [1]) that has been used with success for Web caching. GDS incorporates in a simple way the most important characteristics of an object in the cache: its access history, file size, and recentness 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. An important piece of information is how much of the total

file is stored in the cache. The objects stored in a FastTrack cache can have the following specialized attributes:

- maximum size: the maximum size of the object - for files, it can be larger than the size of the object in the cache,
- 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.
- 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 tries to take into account the proportion of the file that is 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. The policy will be called 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, we have evaluated only one specialized policy, that of MINRS.

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.

5. Simulator Design

We use a trace driven simulation to compare various cache replacement strategies. We use CacheSim a Java-based simulation and traffic statistics package that has been used to study HTTP traffic and cache filtering [4]. We extended CacheSim with the capability to process FastTrack traces and to simulate file- and range-based policies

CacheSim implements replacement policies as priority queues. The ordering in the priority queue is determined by a comparison rule. This implementation allows easily adapting CacheSim to range-based caching for the FastTrack protocol.

Cache replacement is executed after objects are inserted. In other words, it is possible that a new object causes a temporary cache overflow that is followed by a reduction of cache size. No high/low watermarks are used, therefore the cache starts to replace objects when it reaches maximum capacity and stops removing objects when it has sufficient space to store the new object. The main reason for executing cache replacement after insertion is that before insertion it is difficult to tell how much space will be needed. When a new file range is inserted, it could overlap with ranges already in the cache, and then the cache requires less space than the entire range size.

CacheSim code is released under the GNU public license and is available from the authors on request.

6. Comparison of Replacement Policies

The results of the comparison of replacement policies are presented in Figures 4-6. Figures 4 and 5 present byte hit rates of various policies for full and partial caching, respectively. Results for both file- (suffix '-F' in the plots) and range-granularity (suffix '-R') for replacement policies are presented. Figure 6 presents together the performance of the best policies for partial and full P2P caching. All figures show the ideal hit rates achievable for an infinite cache for the trace used. As mentioned in Section 3, the first part of the trace is used to warm-up the cache, so the presented results are byte hit rates for a warmed-up cache.

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

Relative Size (MINRS) does not perform as well as MINS; the reason could be that this policy may discriminate against the full inclusion of large objects that generate a lot of byte hits in the cache. 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 may indicate that there exists a locality in the user aborts - perhaps some large files on slow links are aborted more frequently than other files. 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-

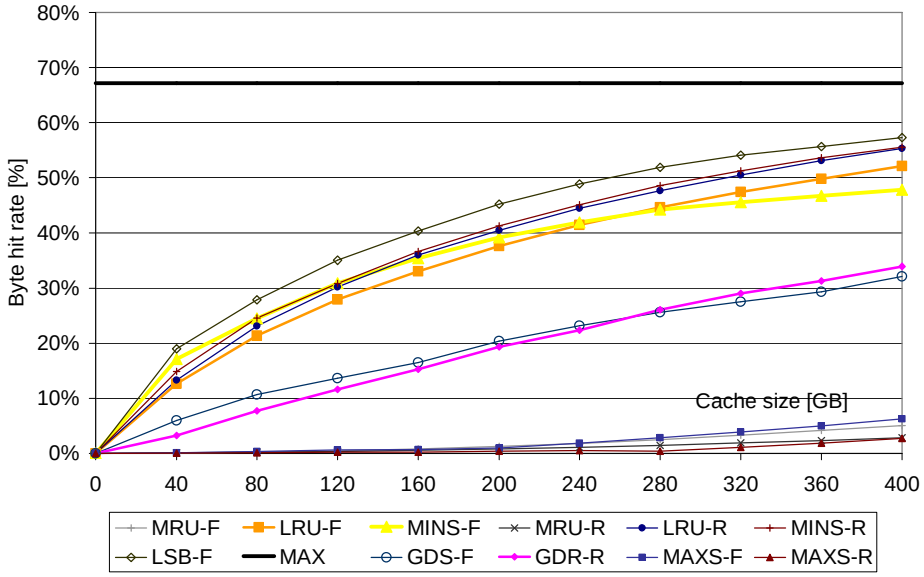

Figure 4: Comparison of replacement strategies for full caching

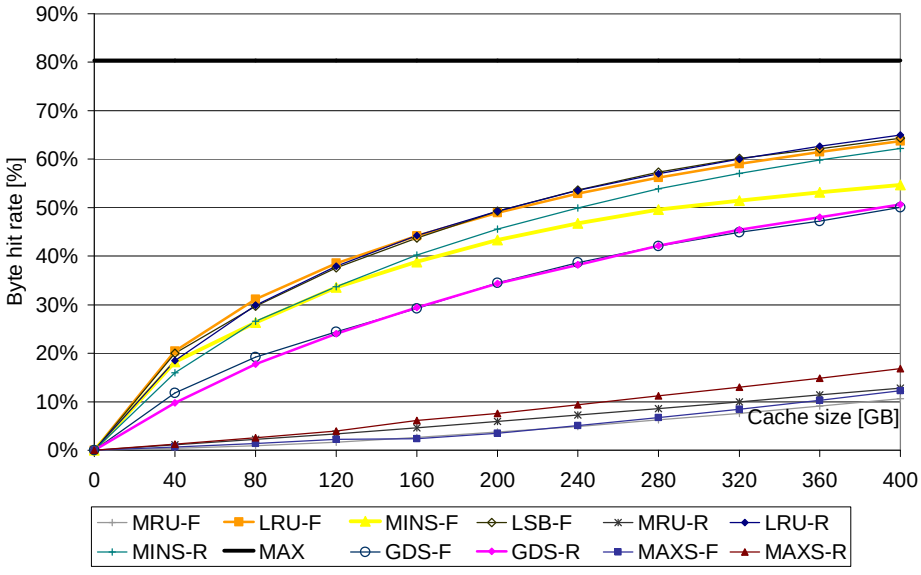

Figure 5: Comparison of replacement strategies for partial caching.

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.

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) could 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]. When the cache sizes we study are compared to the disk and memory sizes of the cache used in the experimental setup (200GB and 1GB, respectively) it becomes evident that effective cache replacement policies are can bring significant savings.

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 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 downloaded about 1.5 TB (for the first log). When the cache ignores user aborts, byte hit rate grows to as much as 90% however the generated traffic grew to 30TB. We infer that this form of prefetching is not desirable when the main goal is traffic reduction.

7. Summary and Future Work

The results presented here are only a first step in exploring cache replacement policies for FastTrack traffic. The large volume of this traffic (thus high potential caching benefits) and the large cache sizes (thus nontrivial costs associated with caching) 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.

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 a control message that notifies the requesting user agents that

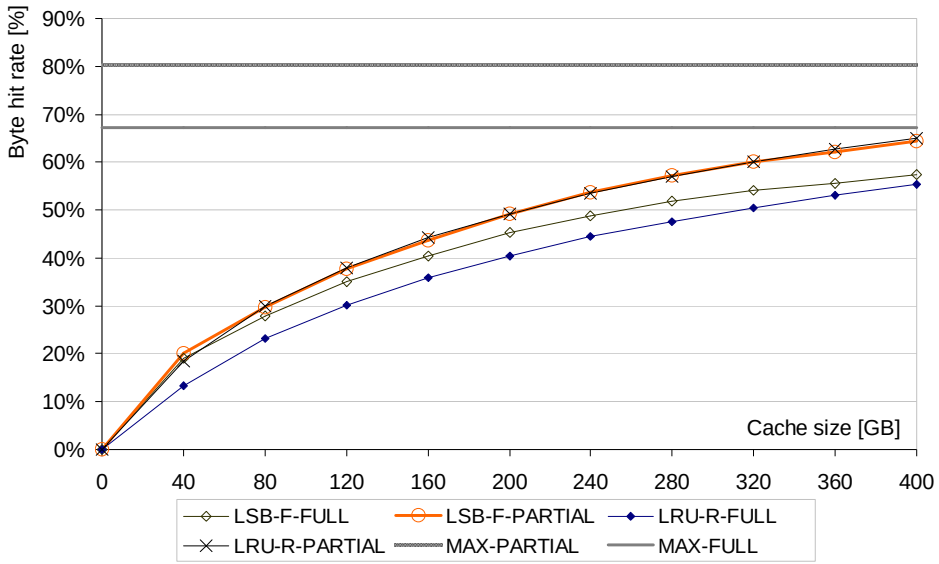

Figure 6: Best policies for partial and and full P2P caching

only parts of the requested range are 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 did not perform 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. An important target of our future work is the development of other specialized policies. We also hope to validate our results on other traces and by comparing predicted hit rates with live cache performance.

8. 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 Kazaa 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