Efficiency and Performance of Web Cache Reporting Strategies

John Chuang
UC Berkeley
chuang@sims.berkeley.edu

Steve Kafka
UC Berkeley
skafka@sims.berkeley.edu

Kim Norlen
UC Berkeley
knorlen@sims.berkeley.edu

Abstract

World Wide Web content providers often resort to "cache-busting" in order to obtain demographic information. Object usage reporting methods have been proposed to address this problem. We quantitatively compare strategies for reporting object hits from proxy caches back to origin servers, and propose novel strategies for improving reporting performance and efficiency. Examining hit-metering and usage-limiting approaches proposed in RFC 2227, we find a fundamental trade-off between reporting latency and efficiency. Further, we find the temporal locality in the server reference stream to be significantly stronger than that in the object reference stream. We propose a server report aggregation strategy that leverages this fact, and show that it can reduce reporting latency and improve efficiency by as much as 80% and 100% respectively. We also propose and evaluate additional strategies to improve performance. These include: dynamic reporting thresholds, report aggregation in a cache hierarchy, and piggybacking reports on existing HTTP messages.

1. Introduction

The effectiveness of web caching has been reduced by the common practice of "cache-busting," or marking web objects with a no-cache header. Publishers want to collect hit count information about their web objects, but allowing their objects to be cached eliminates a publisher's ability to track object popularity and usage patterns. There are many ways a publisher can prevent a page from being cached: using the *Cache-control*, *Pragma*, *Cache-expires*, or *Expires* fields of the HTTP header [1], or alternately setting the *Pragma* or *Expires* fields in the HTML body with a meta tag.

Examining the fifty most popular websites, as ranked by Media Metrix [2], reveals over half of the websites (26 out of 50) perform cache-busting on their main page, and a third (16 out of 50) do so even on their privacy statement page, which contains only static content. Finally, an additional two sites employ "web bugs," or 1x1 transparent gif's embedded in web pages for traffic monitoring purposes.

If web caches send detailed hit count reports back to the publishers, the publishers will no longer need to cache-bust or insert transparent gif's in order to monitor their site traffic. RFC 2227 details several options for caches and publishers to coordinate their reporting strategies [3]. Under the hit-metering strategy, a publisher can request that cache reports be sent at a regular time interval (by specifying a *timeout* value), while under the usage-limiting strategy, the publisher can request that reports be sent when a hit count threshold is reached (by specifying a *max-uses* value). Additional reporting triggers include object purges and conditional GET/HEAD messages. The cache reports are sent using HTTP conditional HEAD messages, with special headers for carrying reporting information.

This work provides the first quantitative study of the efficiency and performance of web cache reporting strategies. First, we analyze the RFC 2227 strategies, and find the hit-metering strategy provides predictable bounds on reporting latency, while the usage-limiting strategy provides predictable bandwidth efficiency levels. In general, there is a fundamental tradeoff between timeliness of reports and bandwidth use. Setting smaller *timeout* and/or *max-uses* thresholds results in lower reporting latency at the expense of higher bandwidth usage. Publishers can thus choose the appropriate strategy and threshold to meet their reporting latency requirements while minimizing reporting cost.

Using stack distance modeling techniques, we find in web reference streams a significantly higher degree of temporal locality at the server reference level than at the object reference level. This suggests that significant efficiency gains can be realized by aggregating reports for multiple objects from the same server into a single per-server report. Indeed, our simulations show that this technique alone can reduce latency and improve efficiency by as much as 80% and 100% respectively.

We also propose and evaluate other novel reporting strategies, including the use of dynamic thresholds under

the usage-limiting strategy, piggybacking reports on existing HTTP traffic, and aggregating reports in a caching hierarchy. These strategies provide additional latency and efficiency gains, and can be easily implemented as extensions to RFC 2227, or incorporated with proprietary reporting mechanisms used by content delivery networks (CDNs).

This paper is organized as follows: we provide a brief description of the methodology and data in Section 2. In Section 3, we define the key performance metrics for cache reporting, and provide quantitative evaluation of the hit-metering and usage-limiting strategies outlined in RFC 2227. We also propose and evaluate the use of dynamic usage limits. In Section 4, we quantify the degree of server-level temporal locality in web reference streams, and propose a per-server report aggregation strategy to leverage this locality. Finally, we examine reporting strategies involving piggybacking and hierarchical aggregation in Section 5 before concluding the paper.

2. Methodology and Data

We extend the trace-driven cache simulator used in [4] to implement the various reporting strategies described in RFC 2227. The RFC specifies five situations when the cache MUST send a usage report:

1. When the timeout period of an object has expired (hit-metering mode)
2. When the max-uses limit of an object has been exceeded (usage-limiting mode)
3. When the object (or hit count information) is purged from cache
4. When the cache forwards a conditional GET message from one of its clients
5. When the cache forwards a conditional HEAD message from one of its clients

In addition, we also implement the following in the simulator:

1. A baseline reporting strategy where a report is generated for every object cache hit
2. Dynamically adjustable usage-limits (Section 3)
3. Server-level reporting strategies where reports are generated according to per-server max-uses and timeout thresholds (Section 4)
4. Piggybacking reports on other HTTP messages (Section 5)
5. Running the simulator in two modes (leaf cache or parent cache) to study hierarchical cache reporting strategies (Section 5)

We use four different proxy traces from Boeing [5], DEC [6], NLANR [7], and an anonymous corporate proxy ("Big Corp.") in our simulations (Table 1) and obtain results consistent across the traces.

Table 1. Trace summary

Trace	Start date	Duration	# Requests	Request rate
Boeing	3/1/1999	24 hrs	4.3 million	49.7/s
DEC	9/11/1996	89 hrs	4.3 million	13.4/s
NLANR	3/26/2002	33 hrs	3.3 million	27.7/s
Big Corp.	7/10/2001	12 hrs	3.3 million	74.3/s

3. Evaluation of Reporting Strategies

3.1. Metrics

There are several ways to measure the effectiveness of reporting strategies. From a publisher's perspective, the most important metric is latency, or the delay from when an object receives a cache hit to when that hit is reported to the publisher. Some publishers may require real-time or near-real-time reporting to support on-the-fly content customization or click-stream analysis. Other publishers may be satisfied with an hourly or daily report. We calculate the average latency of all reported cache hits for each reporting strategy.

Another important metric is the bandwidth cost savings realized from each strategy. We propose a bandwidth efficiency metric (E) based on the number of hits per report:

$$E = 1 - (x)^{-1} \quad (1)$$

where x is the number of hits per report. We define a baseline reporting strategy as generating a report for each cache hit. Thus for the baseline case of one hit per report, $x = 1$ and $E = 0$. Conversely, E approaches 1 as x approaches infinity.

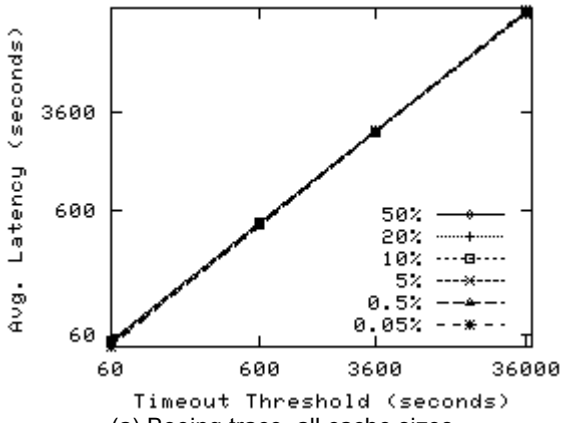
This efficiency metric can be used to compute actual bandwidth savings in terms of bytes, depending on the formatting of the reports. RFC 2227 proposes simply reporting the number of hits for a given page. In this case, all reports will be the same size. We propose generating a line for each hit reported containing the URL, client IP address, and timestamp. In this way, the publisher receives a more robust report and proxy administrators may combine hits on different objects from the same server into one report (Section 4). For this reporting format, each report has a size of $k_0 + k_1x$ where k_0 is the size of the header in bytes, k_1 is the size of a line for a reported hit in bytes and x is the number of hits in the report. We can calculate the average bytes per hit as bandwidth cost, $BW = k_0(1-E) + k_1$. Thus regardless of the reporting method, one can compare various strategies for bandwidth usage. The RFC 2227 reporting format is a special case where $k_1 = 0$ and $BW = k_0/x$.

A final metric to consider is the number of hits that are not reported during the simulation period. There are two reasons why a cache hit will not be reported. The first is that the trace is finite. When the simulation ends, there will be reports that have not yet reached their trigger, and thus are not sent. The second reason a hit may not be reported is that the trigger for that object may never be reached. For example, if a report is only sent when an object is purged from the cache, a popular object may never generate a report. While it is not possible to attribute particular unreported hits to one cause or the other, the percentage of unreported cache hits can be taken into consideration when comparing strategies.

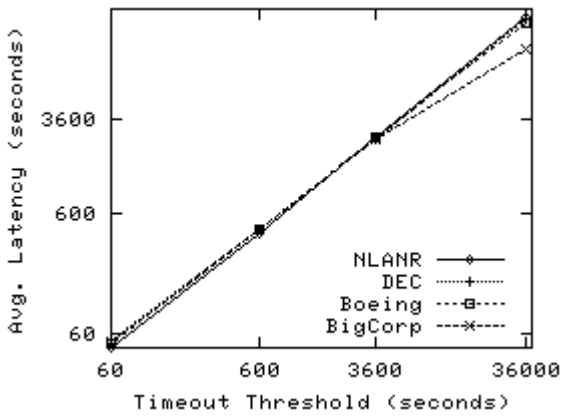
3.2. Hit-Metering vs. Usage-Limiting Strategy

Under the hit-metering strategy, the cache generates a report for an object every T seconds, where T is the timeout value specified by the publisher. This places an upper bound on the latency of object hits. Figure 1 plots the average latency versus the timeout value, with the timeout value ranging from 10 seconds to 10 hours. We observe that the average latency increases linearly with T, but is not impacted by cache size. Here, cache size is measured as a percentage of the total unique bytes of the request stream.

With increasing timeout threshold, each report is expected to contain a larger number of reported hits, thus increasing the bandwidth efficiency of the strategy. The efficiency will asymptotically approach unity as the timeout threshold approaches infinity. Figure 2 shows that a 10 hour reporting period can achieve an efficiency of 0.8-0.9, but the efficiency drops to as low as 0.2 if a publisher requires 10-second reporting periods. Note that for a given time threshold, Big Corp displays the highest efficiency while DEC produces the lowest. This can be attributed to the differing request arrival rates across the different traces. The higher the request arrival rate, the more hits recorded within each reporting period.

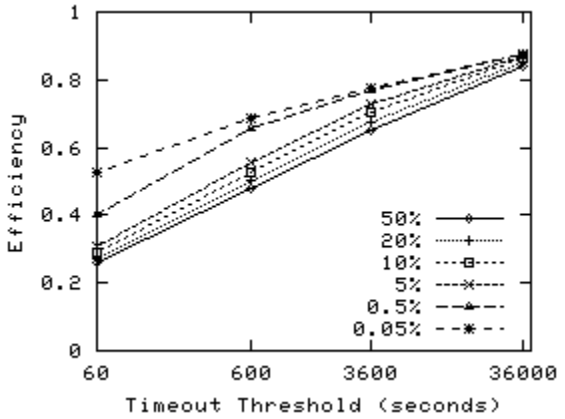


(a) Boeing trace, all cache sizes

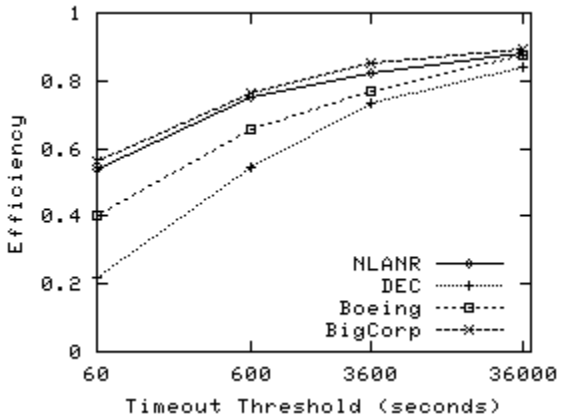


(b) All traces, cache size = 0.5%

Figure 1. Average reported hit latency v. timeout threshold for hit-metering strategy



(a) Boeing trace, all cache sizes



(b) All traces, cache size = 0.5%

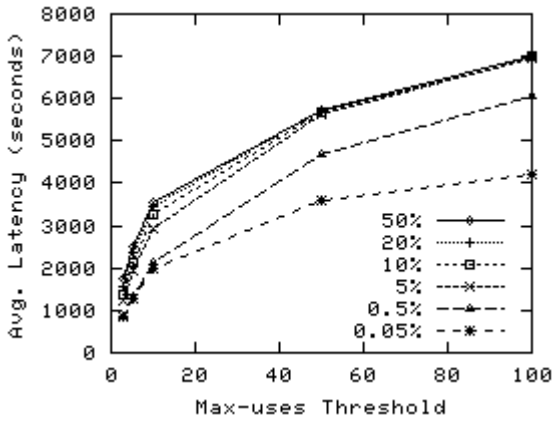
Figure 2. Efficiency v. timeout threshold for hit-metering strategy

Under the usage-limiting strategy, the cache generates a report for every N hits to an object, where N is the max-uses value specified by the publisher. Since all reports have exactly N hits, the efficiency of this strategy follows equation (1) with $x = N$, and is independent of request arrival patterns or cache size. However, this strategy does not provide any guarantees on the timeliness of reports. Figure 3 shows the increase in average hit latency with increasing hit threshold. Indeed, cache hits for a given object may never be reported if the max-uses threshold is not reached. Figure 4 shows the percentage of cache hits that are unreported under the two schemes. It is interesting to note that a usage-limiting strategy with max-uses threshold of 3 hits fails to report a larger proportion of hits (14.9%) than a hit-metering strategy with a timeout threshold of as long as 10 hours (11.9%).

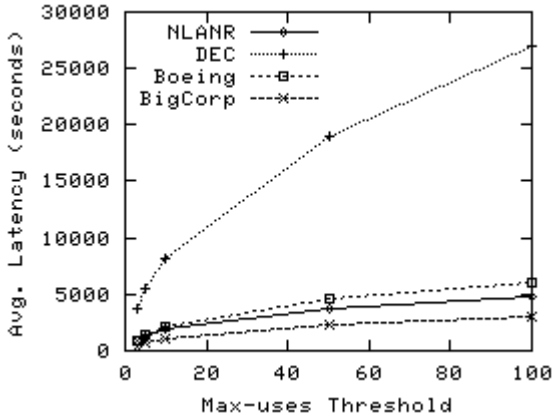
Comparing the two strategies, we see a fundamental tradeoff between reporting timeliness and efficiency. The hit-metering strategy allows the publisher to specify tight latency bound, at the expense of efficiency, while the usage-limiting strategy provides predictable bandwidth efficiency with no latency guarantees.

3.2.1. Dynamic Usage Limits. The effectiveness of the usage-limiting strategy is heavily influenced by the choice of N, the max-uses threshold. Ideally, the max-uses threshold should be chosen to match the popularity of the object. Earlier work has established that object popularity can be characterized using Zipf's Law [8]. A frequently requested object should be tagged with a large N to achieve higher bandwidth efficiency, while a rarely requested object should have a small N to minimize the number of unreported hits. Given the difficulty in predicting the short-term popularity of objects, we propose that the max-uses threshold be dynamically adjusted according to:

$$N(i) = \min[2^i, N_{\text{ceiling}}] \quad (2)$$

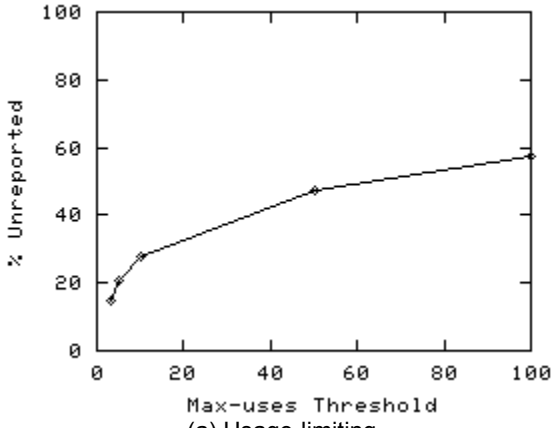


(a) Boeing trace, all cache sizes

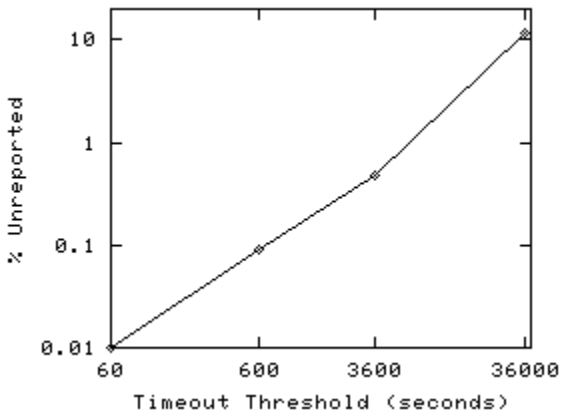


(b) All traces, cache size = 0.5%

Figure 3. Average reported hit latency v. timeout threshold for usage-limiting strategy



(a) Usage-limiting



(b) Hit-metering (log scale for y-axis)

Figure 4. Percent of cache hits unreported (Boeing trace, cache size = 0.5%)

where i is the number of prior reports that has been generated for the object, and $N_{ceiling}$ is the ceiling max-uses value. This means the number of hits per report doubles for each subsequent report generated, up until the ceiling. Note that N is reset to one if an object re-enters the cache after an earlier purge from the cache.

We compare the latency bounds and bandwidth efficiencies of a dynamic usage-limited strategy ($N_{ceiling}$ = 64) against two static usage-limited strategies (N = 3 and N = 100 respectively.) Specifically, we are interested in how the strategies perform for objects of different popularity levels. For non-popular objects, we find that the strategies with either a small static threshold (N = 3) or a dynamic threshold ($N_{ceiling}$ = 64) provide bounded latencies, but not the strategy with a large static threshold (N = 100) (Figure 5). Indeed, all objects with fewer than 100 hits will never receive reports under the N = 100 strategy. On the other hand, for popular objects, the large static threshold and the dynamic threshold provide much higher efficiency than the small static threshold (Figure 6). This confirms the suitability of dynamic thresholds for handling objects with varying degrees of popularity.

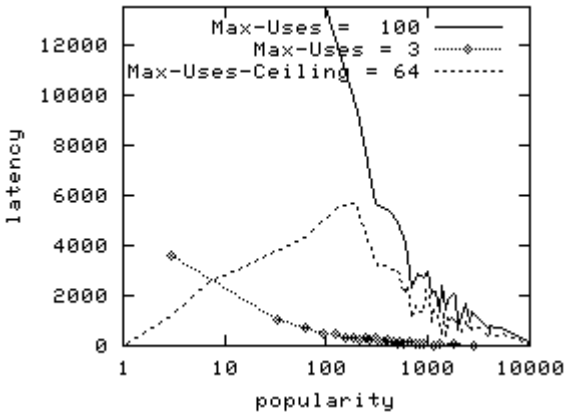


Figure 5. Average reported hit latency v. popularity (hit frequency) (Boeing trace, cache = 0.5%)

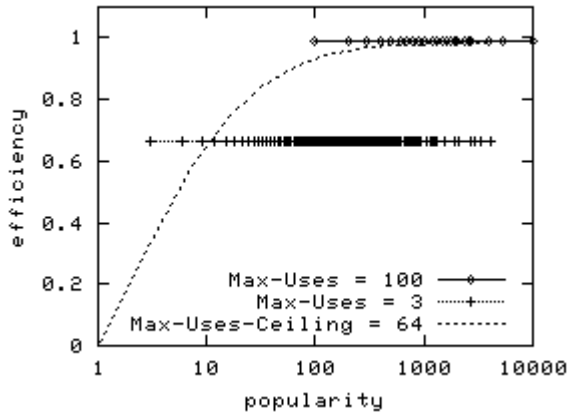


Figure 6. Efficiency v. popularity (hit frequency) (Boeing trace, cache = 0.5%)

3.2.2. Combining strategies. We combine a hit-metering strategy (with timeout = 1 hour) and a dynamic usage-limiting strategy (with $N_{ceiling}$ = 64 hits). Combining strategies implies more reporting triggers, and therefore lower reporting latency and higher reporting rate. The efficiency is also correspondingly lower than those of the standalone strategies (Table 2).

Table 2. Combining hit-metering and dynamic usage-limiting strategies (Boeing trace, 0.5% cache size)

Strategy	Av. latency	Average hits/ report	Efficiency	% un-reported
T = 3600s	2554 s	4.40	0.77	0.48%
$N_{ceiling}$ = 64	2615 s	3.94	0.75	17.11%
both	944 s	2.98	0.66	0.26%

3.3. Purge Trigger

In addition to the hit-metering and usage-limiting triggers, reports can also be generated upon object purges. The purge trigger is most effective in reducing the number of unreported hits when used in conjunction with large usage limits. Popular objects will have their reports generated by the usage limit trigger, while non-popular objects will have their reports generated by the object purge trigger. For example, with max-uses = 100, the purge trigger results in the reporting of an additional 23% of hits (Table 3). The purge trigger makes little difference when used in conjunction with a small max-uses threshold or with a timeout threshold, since even non-popular objects usually stay in cache for at least several hours. As a side note, we strongly advise against a reporting strategy that is based solely on purge triggers. Generating reports only upon object purges will result in a large number of unreported hits, mainly because popular objects might not get purged at all, and so all their hits go unreported.

Table 3. Effect of purge trigger on usage-limiting and hit-metering strategies (Boeing Trace, 0.5% Cache Size)

Strategy	Average latency (s)	Average hits/ report	Efficiency	% hits unreported
$N = 3$	872	3	0.67	14.85%
$N = 3$ & purge	950	2.67	0.63	12.89%
$N = 100$	6068	100	0.99	57.61%
$N = 100$ & purge	4406	8.08	0.88	34.10%
$T = 60$s	50	1.68	0.41	0.01%
$T = 60$s & purge	49	1.66	0.40	0.01%
$T = 3600$s	2544	4.40	0.77	0.48%
$T = 3600$s & purge	1959	3.48	0.71	0.42%

4. Aggregating Reports to Origin Servers

The composition of web pages of multiple embedded objects implies that HTTP requests for a given server tend to arrive in clusters. In this section, we aim to quantify this phenomenon, and use it to motivate a per-server reporting strategy.

4.1. Temporal Locality at the Server Level

While previous work [8-10] has studied the temporal locality of web page request streams, we are unaware of any literature regarding temporal locality of servers in a request stream. For web caching, the temporal locality of servers is unimportant; whether a page is cached or not is based on the characteristics of the object itself. For web cache reporting, on the other hand, temporal locality on the server level could allow for more efficient reporting strategies. Because reports will be sent to the server that published an object, combining reports for different objects that are to be sent to the same server could be more efficient.

Following [9-11] we use the stack distance model [12] to quantify the presence and degree of temporal locality. At the object level, we calculate the number of distinct objects referenced since the last time a particular object was requested. At the server level, we calculate the number of distinct servers referenced since the last time any object from a particular server was requested. We then plot the distribution of stack distances against the frequency of each distance in the trace.

Figure 7 shows the log-log plot of stack distance vs. frequency for the Boeing trace at the server and object

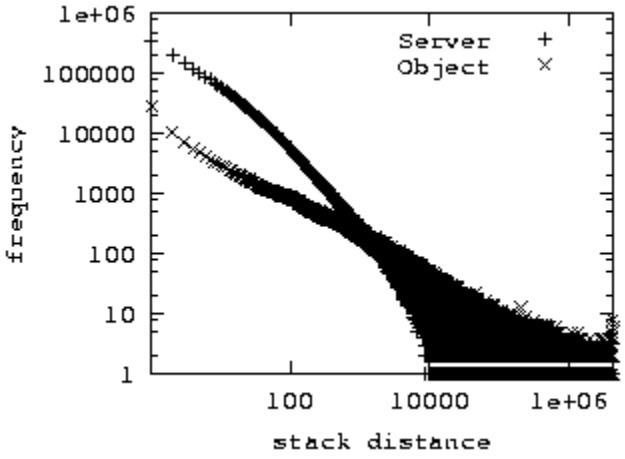


Figure 7. Stack distance v. frequency for request stream (Boeing trace)

levels. We can clearly see that the server stack distance plot has a steeper slope than the object stack distance plot. Using ordinary least squares regression we find that the server stack distance plots are almost twice as steep as the object stack distance plots (Table 4). We conclude that temporal locality is not only present at the server level, but clearly much stronger than at the object level. This is consistent with our expectation that clients tend to request multiple objects from the same site together or in close succession.

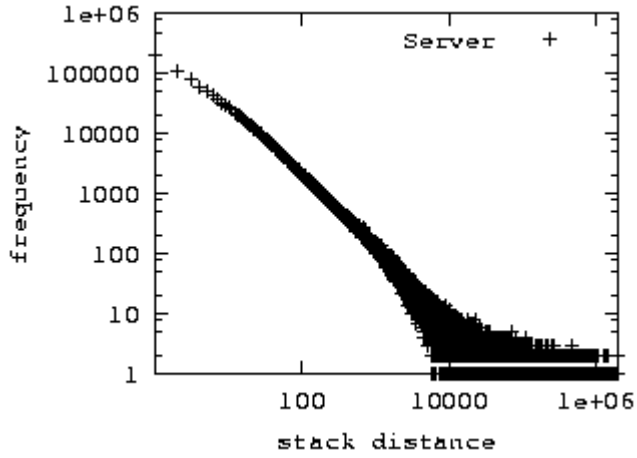
Table 4. Temporal Locality: Request stream

Trace	Slope (r^2)	
	Object Level	Server Level
Boeing	-0.74 (0.96)	-1.45 (0.97)
NLANR	-0.96 (0.96)	-1.38 (0.97)
DEC	-0.89 (0.96)	-1.59 (0.98)
Big Corp.	-0.80 (0.97)	-1.41 (0.97)

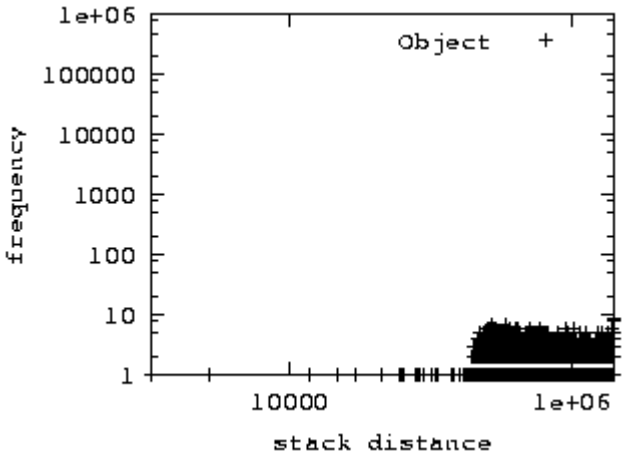
The cache hit stream also displays stronger temporal localities at the server level over the object level. Surprisingly, even the purge stream exhibits temporal localities at the server level (Figure 8a) even though none can be observed at the object level (Figure 8b).

4.2. Per-Server Report Aggregation

Given the increased temporal locality of request, hit and purge streams at the server level, we propose a reporting strategy whereby hits for all objects from a given server are combined in any report sent to the server. In this scenario, the reporting format may include the URL and hit count for the individual objects, so that the report recipient can determine the allocation of hits among the different objects.



(a Server Level



(b) Object level

Figure 8. Stack distance v. frequency for purge stream (Boeing trace, cache size = 10%)

Under the hit-metering strategy, each server has a single timeout value *T*, and a report will be generated every *T* seconds as before. However, the report will contain all hits for all the objects that originate from the server, rather than for a single object. The efficiency improvement ranges from 12% at *T*=36000s to 105% at *T*=10s (Figure 9). In general, the benefit of server-level reporting is greatest at short reporting periods.

Under the usage-limiting strategy, the collection of all objects from a server has a single max-uses value *N*, and a report is triggered whenever *N* hits are registered for the server, regardless of which individual objects are requested. In Figure 10, we see latency reductions of 70% to 80% for the entire range of max-uses thresholds when switching from per-object reports to per-server reports.

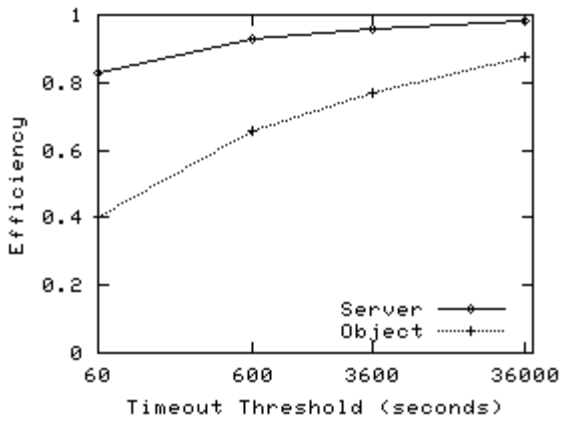


Figure 9. Efficiency v. timeout threshold (Boeing trace, cache size = 0.5%)

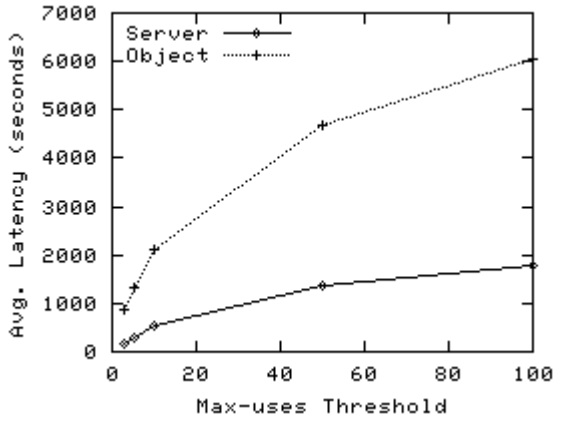


Figure 10. Avg. latency v. max-uses threshold (Boeing trace, cache size = 0.5%)

Similarly, per-server reporting produces better efficiency and latency numbers than per-object reporting in conjunction with the use of dynamic usage-limits. Figures 11 and 12 show efficiency improvements of 26% to 30% and latency reductions of 22% to 56% across the range of dynamic max-uses thresholds we examined.

5. Piggybacking and Hierarchical Cache Reporting

In this section, we consider a class of reporting strategies that maximizes efficiency by piggybacking reports on existing HTTP traffic:

- per-object report piggybacked on conditional GET (or If-Modified-Since) message (as specified in RFC 2227)

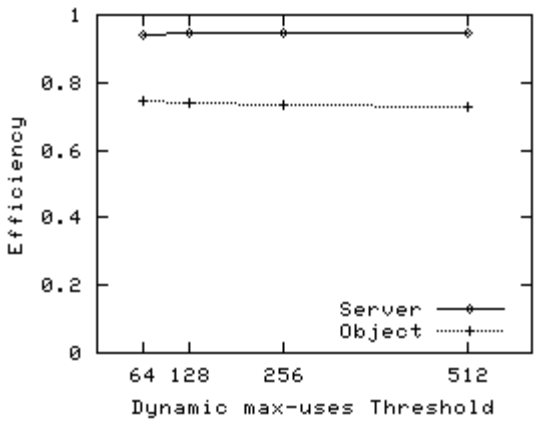


Figure 11. Efficiency v. dynamic max-uses threshold (Boeing trace, cache size = 0.5%)

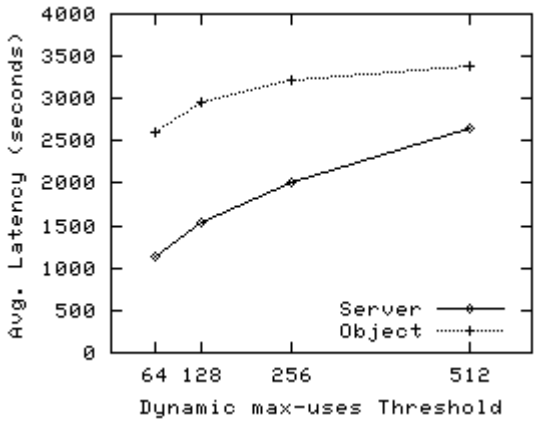


Figure 12. Avg. latency v. dynamic max-uses threshold (Boeing trace, cache size = 0.5%)

- per-server report piggybacked on conditional GET (or If-Modified-Since) message
- per-server report piggybacked on GET message (due to cache miss)
- parent cache report piggybacked on child cache report (in cache hierarchies)

By piggybacking a report on a message that is already being sent by the cache back to the origin server, we can achieve maximum bandwidth savings since there is no message header overhead. Thus, no matter how many hits are contained in each report, the efficiency is always 1. In this study we consider IMS (If-Modified-Since) messages and regular GET messages due to cache misses, but the technique can be readily applied to other types of messages exchanged between caches and servers, such as those used in the Web Cache Invalidation Protocol [13].

5.1. IMS and Cache Miss

Caches routinely send IMS (If-Modified-Since) messages to origin servers to query the freshness of objects. The IMS strategy entails piggybacking a report on an IMS message that is already being sent to the server. In the per-object IMS strategy, only hits for that particular object are reported, while in the per-server IMS strategy, all hits for the server's objects are reported. As in Section 4, we see that the per-server IMS reporting strategy realizes a 80% latency reduction over the per-object IMS strategy (Table 5).

Caches send HTTP GET messages (in response to cache misses) on a more frequent basis. Therefore we also evaluate the effectiveness of these messages as vehicles for piggybacking reports. Per-object reports do not apply in this case, since it is unlikely for a cache to have any hits to report for an object it doesn't have. However, per-server reports can be piggybacked on GET messages whenever the cache needs to fetch new objects from the server. As shown in Table 5, this strategy achieves very low reporting latencies and unreported rates without compromising on efficiency.

Table 5. IMS strategy (Boeing trace)

Strategy	Avg. latency (s)	Avg. hits/ report	Efficiency	% un-reported
IMS (object level)	2495	2.42	1.0	23.21%
IMS (server level)	515	4.10	1.0	7.78%
Cache miss (server level)	186	3.05	1.0	1.31%
IMS + Cache Miss (server)	133	2.44	1.0	1.19%

5.2 Hierarchical Cache Reporting

Another situation where existing traffic presents an opportunity to piggyback reports is in a caching hierarchy. Since reports from each child cache are forwarded through the parent cache, adding the parent's reports to the child's can improve both latency and bandwidth efficiency.

To simulate a two-level hierarchical cache, we run the simulator with traces from five sibling caches collected at Boeing in March 1999. Each sibling cache uses the same

reporting strategy and spans the same time period. The sibling caches generate reports and cache misses, which are forwarded to the parent cache. The parent uses the same reporting strategy and relative cache size as its children. In addition, the parent cache may choose to operate in one of two modes: (i) forward all child reports (no piggyback), or (ii) adds its own cache hits, if any, to the child report (piggyback) (Figure 13).

We find that hits to the parent cache are responsible for about a third of all hits reported to the publisher. When used in conjunction with dynamic usage-limiting thresholds (i.e., the use of $N_{ceiling}$), piggybacking parent cache hits on child reports provides latency reductions in the range of 25% to 75% (Table 6). Only modest latency reductions are realizable, however, for the other reporting strategies. Hierarchical cache reporting also does not significantly impact bandwidth efficiency in any way.

Finally, we note that this technique can be extended to cooperative caching schemes, where object queries may be exchanged between sibling caches using ICP [14] or CARP [15] messages. The sibling caches may choose to aggregate their hit reports to further improve reporting latency and efficiency.

6. Conclusion

This paper quantitatively analyzes different web cache reporting strategies specified in RFC 2227 and introduces novel approaches for improving both reporting latency and efficiency. There are several key findings in this work. First, for the two reporting strategies specified in RFC 2227, we find the hit-metering strategy provides predictable bounds on reporting latency, while the usage-limiting strategy provides predictable bandwidth efficiency levels. A lower latency bound can be achieved at the cost of lower bandwidth efficiency. Publishers can thus choose the appropriate strategy and threshold to meet their application needs at minimum bandwidth cost. In addition, we propose the use of dynamic thresholds under the usage-limiting strategy, which yields significant latency and efficiency improvements over static thresholds for objects with varying or unpredictable popularity levels.

Second, we find that temporal locality in the reference stream is significantly stronger at the server level than at the object level. We can leverage this fact by aggregating hits for multiple objects into a single per-server report, and achieve latency reductions and efficiency improvements by as much as 80% and 100% respectively.

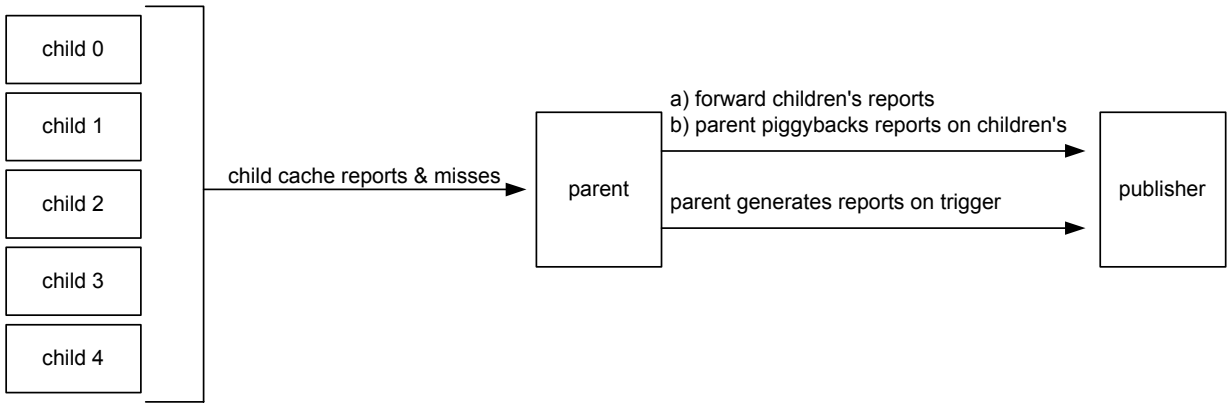


Fig 13. Two-level hierarchical proxy cache simulation

Table 6. Latency and efficiency improvements for piggybacking reports in a caching hierarchy

Strategy	0.5% cache size				50% cache size			
	Average latency (s)		Efficiency		Average latency (s)		Efficiency	
	No piggyback	Piggyback	No piggyback	Piggyback	No piggyback	Piggyback	No piggyback	Piggyback
$T = 60$	49	49	0.423	0.449	54	54	0.029	0.051
$T = 3600$	2447	2262	0.779	0.782	2536	2449	0.665	0.686
$N = 3$	482	439	0.667	0.694	925	795	0.667	0.698
$N = 100$	2544	2806	0.990	0.991	2900	2708	0.990	0.991
$N_{ceiling} = 64$	864	574	0.672	0.692	750	566	0.535	0.558
$N_{ceiling} = 64$ & $T = 3600$	179	46	0.260	0.267	155	45	0.611	0.637

Third, we find that piggybacking reports on existing HTTP traffic such as IMS and GET messages can further improve latency and efficiency. Report aggregation in caching hierarchies can produce 25-75% latency reductions, but only when using the usage-limiting strategy with dynamic thresholds. Negligible gains are realizable for the other reporting strategies.

In general, we find that aggregation and piggybacking techniques can be exploited in a variety of contexts and used in conjunction with a variety of protocols (e.g., HTTP, ICP, CARP, WCIP) to improve the performance and efficiency of cache reporting strategies. The strategies and techniques considered in this analysis can be easily implemented as extensions to RFC 2227, or incorporated into proprietary reporting schemes used by content delivery networks. Efficient and reliable reporting mechanisms, when deployed widely, will eliminate the need for publishers to cache-bust for traffic monitoring purposes.

Acknowledgement

This work is supported by the U. S. National Science Foundation under Cooperative Agreement Number ITR-0085879. We thank the anonymous reviewers for their insightful comments.

References

[1] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol -- HTTP/1.1. RFC 2616, June 1999.

[2] Jupiter Media Metrix. U.S. Top 50 Web and Digital Media Properties Unique Visitors. April 2002. http://www.jmm.com/xp/jmm/press/mediaMetrixTop50.xml

[3] J. Mogul and P. Leach. Simple Hit-Metering and Usage-Limiting for HTTP. RFC 2227, October 1997.

[4] P. Cao and S. Irani. Cost-aware WWW proxy caching algorithms. Proceedings of the 1997 USENIX Symposium on Internet Technology and Systems, 193-206.

[5] Boeing Proxy Logs, ftp://researchsmp2.cc.vt.edu/pub/boeing/

[6] Digital Equipment Cooperation Web Proxy Traces, ftp://ftp.digital.com/pub/DEC/traces/proxy/webtraces.html

[7] National Laboratory for Applied Network Research Proxy Logs, ftp://ftp.ircache.net

[8] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web Caching and Zipf-like Distributions: Evidence and Implications. In IEEE Infocorn, pages 126 134, March 1999.

[9] V. Almeida, A. Bestavros, M. Crovella, and A. de Oliveira. Characterizing Reference Locality in the WWW. Proceedings of 1996 International Conference on Parallel and Distributed Information Systems (PDIS '96), 92-103.

[10] S. Jin and A. Bestavros. Temporal Locality in Web Request Streams: Sources, Characteristics, and Caching Implications. Proceedings of ACM Sigmetrics 2000, Santa Clara, CA, June 2000.

[11] L. Cherkasova and G. Ciardo. Characterizing Temporal Locality and its Impact on Web Server Performance. Proceedings of ICCN2000, Las Vegas, NV October 2000.

[12] R. Mattson, J. Gecsei, D. Slutz, and I. Traiger. Evaluation techniques and storage hierarchies. *IBM Systems Journal,* 9:78-117. 1970.

[13] D. Li, P. Cao, M. Dahlin. WCIP: Web Cache Invalidation Protocol. Internet Draft <draft-danli-wrec-wcip-01.txt>, March 2001.

[14] D. Wessels and K. Claffy. Internet Cache Protocol (ICP), version 2. RFC 2186, September 1997.

[15] V. Valloppillil and K. W. Ross. Cache Array Routing Protocol v1.1. Internet Draft <draft-vinod-carp-v1-03.txt>, February 1998.