

# A Proxy Server Structure and its Cache Consistency Mechanism at the Network Bottleneck

Joo-Yong Kim, Kyoung-Woon Cho and Kern Koh  
Dept. of Computer Science  
Seoul National University  
Seoul, 151-742, Korea  
{jykim,cezanne,kernkoh}@oslab.snu.ac.kr

## Abstract

*The response time, the bandwidth usage, and the server load are three representative performance metrics used for the proxy cache consistency mechanisms. However, the primary metric affecting the overall performance can be different depending on the situations.*

*At the network bottleneck such as the inter-continental backbone, the network bandwidth is limited. Also, the response time is a basic performance metric used for the proxy cache, and unsatisfactory response time can be a reason that the clients hesitate to use the proxy cache, which may increase the network traffic potentially. Hence, we should give higher priorities to the response time and the bandwidth usage at the bottleneck, compared to other metrics.*

*In this paper, we suggest a proxy server structure on the network bottleneck, which improves the response time even with a little network traffic reduction at the bottleneck. Its cache consistency mechanism is based on the periodic polling, performed at the polling server located beyond the bottleneck. Our approach may increase the network bandwidth beyond the bottleneck and the loads of the origin web servers. However, the faster response time and the reduced network traffic at the bottleneck can compensate for them.*

## 1. Introduction

The rapid growth of the WWW also leads the increasing needs for the network bandwidth. The proxy server cache was suggested for reducing the network traffic, which caches the popular internet objects. Besides that, the caching in the proxy server can be used to achieve the faster response and to reduce the load of the web server. However, the value of the caching is reduced if the objects in the cache are different with the original ones because of the updates

at the origin servers.

To solve this problem, some proxy cache consistency mechanisms have been suggested[13][7][2][4][1]. They can be classified into three types: *time-to-live field*, *client polling*, and *invalidation* protocol[5]. *Time-to-live(ttl) field* is an estimate of a cached object's life time. Each cached object is assigned a ttl value, and when the ttl elapses the next request causes the object to be requested from the origin server. It can be implemented easily using HTTP "Expires" header. *Client polling* is a mechanism where clients(or the proxy caches) check the validities of the cached objects, and it can be implemented using "If-Modified-Since" field of the HTTP header. If the strong consistency is required, *invalidation* mechanism is used. With the invalidation mechanism, each web server keeps the information about the client caching, and notifies all the clients if the objects are modified. However, the invalidation mechanism has some drawbacks. Firstly, servers have to keep track of where their objects are cached. Secondly, it should consider the temporarily unreachable servers. Finally, it requires the modifications of the servers. In these previous works, the response time, the bandwidth usage, and the server load were three representative performance metrics used for the proxy cache consistency mechanisms, and they were equally weighted.

In this paper, we consider the case of the network bottleneck such as the inter-continental backbone, where overall network bandwidth is limited. For the better response at the bottleneck, the expansion of the bandwidth can be considered. However, it is much more expensive compared with that of the normal link. Also, one of the ways for reducing the network traffic is to encourage the clients to use the proxy server cache, and the faster response may lead the clients to use the cache. Hence, we put higher values for the response time and the bandwidth usage at the network bottleneck compared to other metrics. Moreover, [9] insisted that the response latencies are hardly related with the web

server loads. It means that the server load may be less valuable than other metrics. Consequently, we assume that the response time and the bandwidth usage at the bottleneck are more important than others.

With this assumption, we suggest a proxy server structure and its cache consistency mechanism which can reduce the response time even with a little reduction of the network traffic at the bottleneck. We use the *polling server* to check the validities of the cached objects. The polling server periodically verifies from the origin servers whether each cached object has been updated since the last validation. If some objects has been updated, the polling server sends the invalidation message to the proxy server. Finally the proxy server removes the objects from the cache(or just marks them as stale). The proxy server guesses that all the cached object is fresh. Hence it can immediately respond to the clients as long as the requested object is in the cache, skipping the process to verify from the origin server. Also, the network usage at the bottleneck is slightly reduced because the verification messages from the proxy server are eliminated. The polling server generates many verification messages which increase the network traffic, hence it should be located on the other side of the bottleneck, not to increase the traffic at the bottleneck.

Section 2 describes the suggested structure of the proxy server and its cache consistency mechanism. Section 3 presents the simulation results using some proxy cache logs and analyzes the results. Section 4 explains some ongoing works. Section 5 summarizes the results.

## 2. Server Structure and its Consistency Mechanism

[13] suggested the *lightweight caching server*, which is consisted of *proxy server*, *cache manager* and *rcached* daemon. The *cache manager* uses different consistency mechanisms for the objects depending whether they are registered in *rcached* or not. For the registered objects, *rcached* located on each remote server, sends the invalidation messages when the objects are updated. For the unregistered ones, the cache manager periodically sends the messages with "If-Modified-Since" header(IMS) to the origin server and checks their validities. Hence, the lightweight cache can guess whether its cached objects are fresh, without contacting the origin server at the request time.

The mechanism proposed in our paper is similar to one of the mechanisms of the lightweight proxy server. The *polling server* in our server structure periodically checks the validities of the cached objects like the cache manager of the light weight caching server. However, we consider the limited bandwidth of the network bottleneck. Though the IMS messages are relatively small, the pollings for the bulk of the objects can seriously increase the network traffic at the

bottleneck. So, we put the polling server at the other side of the bottleneck(Figure 1).

When the polling server verifies the updates for some cached objects, it sends the invalidation messages to the proxy server. After receiving the invalidation messages from the polling server, the proxy server removes the objects from the cache(or just marks them as stale). All the cached objects can be considered as fresh. Hence, the proxy server can respond immediately for the clients' requests, which can provide the faster response to the clients. Moreover, it eliminates the network traffic caused by the IMS messages, which are unnecessary in this structure.

The polling server should maintain the *polling list* which includes the following information for the objects being polled.

- URL
- Modification time
- Last validation time

The information is kept on the polling server by extracting from the messages passing through. Such a server structure can incur some latencies because of the connection time between the proxy server and the polling server. However, such an supplementary server is already used in some previous works. The Japan Cache project[11] located an additional proxy server in North America and could get some benefits from it, regarding the issues of the access behavior differences, the cache space, the usage of the international link, and the interconnection in Japan. Also, the supplementary server is used in [10], which compresses the objects to reduce the network traffic. Under such environments, our consistency mechanism incurs no additional latencies. If there is not such an supplementary server, we should use some proper protocols between the proxy server and the polling server, such as the persistent TCP connection used in HTTP 1.1[6]. Such protocols can be used to minimize the latencies incurred by the polling server. However, we assume on later simulations that there is no latency caused by the polling server.

The polling server can increase the network traffic beyond the bottleneck, and also the web server load because of the periodically generated polling messages. We will discuss about the benefits and even the shortcomings in the next section.

## 3. Performance Evaluation

In this section, we compare the performance of the suggested server with the client polling explained in Section 1, assuming that all the cached objects have fixed TTLs. For this comparison, we used four metrics: the response time,

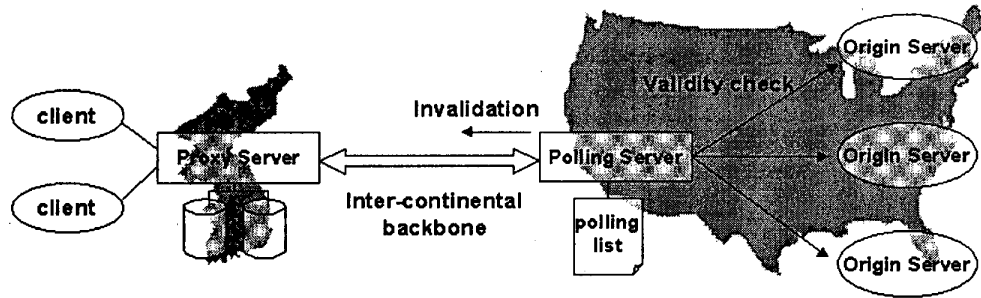


Figure 1. The proxy server structure.

the bandwidth usage at the bottleneck, the bandwidth usage beyond the bottleneck, and the server load.

We used the logs of Seoul National University proxy cache, which were logged during 2 days(19-20th) in November, 1997. In our simulations, the proxy cache was warmed up with the logs of 2 previous days(17-18th, November). The disk is less valuable compared to that of the bandwidth at the bottleneck. We assumed the infinite size cache and didn't consider the cache replacement policy. 10 Gigabytes storage was sufficient to cache all the objects accessed during the 4 days.

In our simulations, we classify the requests into three classes, *HIT*, *MISS*, and *VERIFY*. *HIT* is a request for an cached object which the proxy cache considers fresh. *MISS* can be one of the two cases: the cold miss and the stale miss. The cold miss occurs when the requested object is not in the cache and the proxy server should retrieve the object from the origin server. The stale miss occurs when the requested object is cached but the cache cannot guarantee its freshness, and after contacting with the origin server, the proxy server verifies that the origin server has updated the object. In this case, the proxy server should retrieve the up-to-date object from the origin server. *VERIFY* occurs when the requested object is cached and cannot guarantee its freshness, but after contacting with the origin server, the proxy server verifies that the object is not updated. In this case, the proxy server can return the cached object to the client. Note that in our server structure, stale miss and *VERIFY* never occur if all the objects are polled from the polling server. However, if we use some selection algorithms for the polled objects, it can still experience the stale miss or *VERIFY*.

In [7], the requests were classified into three classes, *In Cache*, *Get*, and *Validate* which are similar to ours. In the paper, the response times are set by the relative values for that of *GET* whose value is 1. Consequently, the relative response times of *In Cache*, *Get* and *Validate* are 0, 1 and a value between 0 and 1 respectively, and they compared some algorithms based on theses values. However, [8] showed that the TCP connection time cannot be ignored.

Table 1. Relative overheads of HIT, MISS, VERIFY

	Response time	Bandwidth usage
HIT	0.179	0
MISS	1	1
VERIFY	0.425	0.030

It means that the response time for the clients is not 0 even though the fresh object is in the cache.

We used the value of the *elapsed time* in the proxy log to compute the response time, which is the time between the `accept()` and `close()` system calls for the TCP socket[12]. We got the overheads of the three request types based on our proxy logs, and recomputed them relatively with the value of *MISS* whose value is set by 1, similarly as in [7]. We showed the overheads in Table 1 and will use these values in the following experiments.<sup>1</sup>

The following results show the performance comparisons between the algorithm used in our server structure and the client polling using fixed TTLs,<sup>2</sup> for various fixed TTL values(or the polling periods). Before explaining the results, we should mention the differences between the *TTL* of the client polling and the *polling period* of our algorithm. With the client polling, the proxy cache assumes an object is fresh, if the following inequality is satisfied.

- Last Validation time > (Current Time - TTL)

Our polling server periodically polls whether cached objects are updated at the origin server, so we can guess the last polling time is similar to the last validation time. Also, the polling period of the polling server has similar feature with the TTL value of the client polling, in that both mechanisms regard an object as stale if it has been updated since

<sup>1</sup>We excluded the objects from the simulations whose URL included ".kr".

<sup>2</sup>In this section, we denote the client polling using fixed TTLs, simply by the client polling.

the TTL time before(or since the last polling time). Figure 2 shows their stale rates for some TTL values(or the polling periods). The stale rate is the number of known stale cache hits divided by the number of total requests. In this figure, we can find that the stale rates are not significantly different between client polling and the periodic polling for the same values of the TTL and the polling period.

Figure 3~ 6 compare the performance of our mechanism with that of the client polling. In these figures, the X-axes indicate the TTL values(the polling periods). The Y-axes indicate the relative values of the response times and the network traffic, compared with the value 1 when all the requests cause MISS and the proxy server has to retrieve all the objects from the origin servers.

In these figures, *polling all objects* shows the performance results of the suggested mechanism in this paper, and the *Client polling* shows the results of the *client polling* using fixed TTLs. The *selective polling* shows the results when a very simple selection policy for the polled object is adopted, which polls the objects from their second requests.

Figure 3 compares the response times. In this figure, the response times are improved especially when the TTL value(the polling period) is small. With the value of 12 hour, the response time is improved about 9.2%(6.5% with the selective polling) compared with that of the client polling. This is because our algorithm removes verifying process at the request time. Also, our polling server can reduce the network traffic at the bottleneck with the same reason. Figure 4 compares the network traffics at the bottleneck. With the value of 12 hour, our server can reduce the traffic at the bottleneck about 1.9%(1.3% with the selective polling).

However, periodic polling can increase the network traffic beyond the bottleneck because it should generate the polling messages periodically. Figure 5 compares the network traffic beyond the bottleneck. With the value of 12 hour, our server increases the network traffic beyond the bottleneck about 18.8%(7.9% with the selective polling). Figure 6 compares the total number of the requests to the origin servers, which can be used to measure the server loads. With the value of 12 hour, the polling server increases the requests about 389%(166% with selective polling).

The results show that our server structure can get faster response and the less traffic at the bottleneck, sacrificing the server load and the bandwidth beyond the bottleneck. To reduce the negative effects, we can use some intelligent selection algorithms for the polled objects. In previous figures, the *selective polling* used a very simple selection algorithm for the polled objects, which polls the objects from their second requests. Even with such a simple algorithm, we could still get the reasonable results, reducing significantly the negative effects of our server. This is because more than half of the objects are accessed only once during

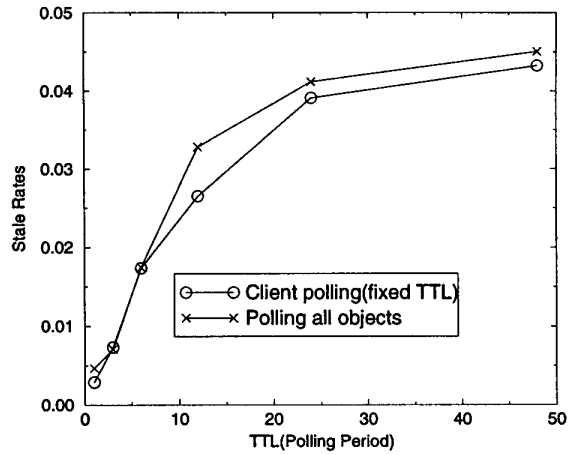


Figure 2. Comparison of the staleness rates.

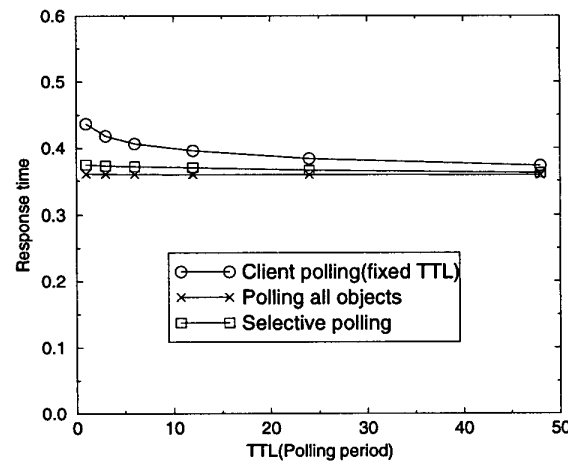


Figure 3. Comparison of the response times.

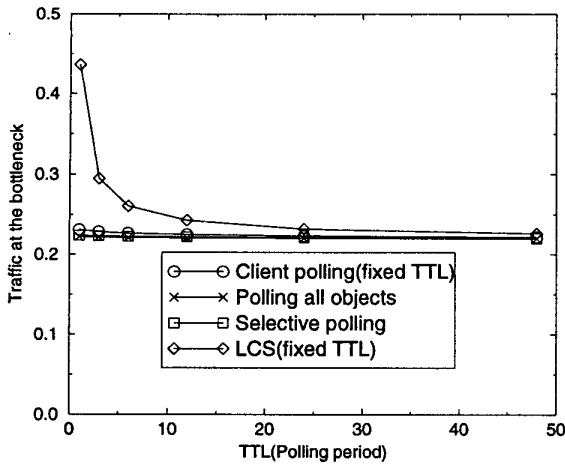


Figure 4. Comparison of the network traffic at the bottleneck.

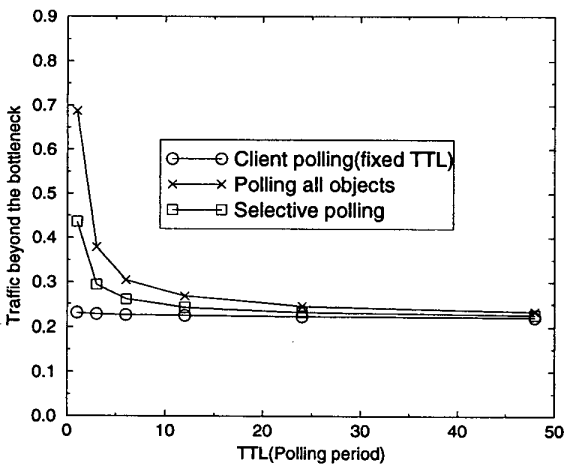


Figure 5. Comparison of the network traffic beyond the bottleneck.

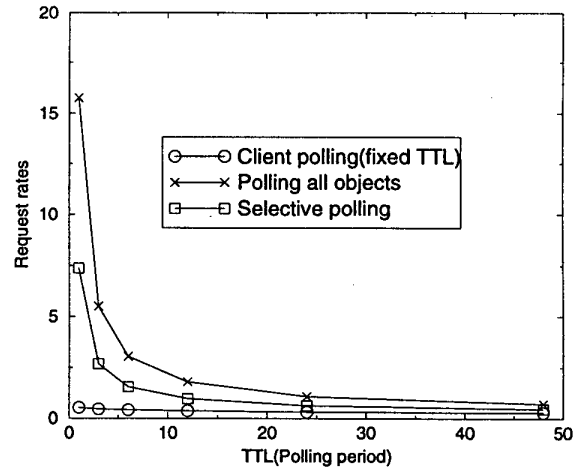


Figure 6. Comparison of the number of the requests

the simulation period.

In section 2, we explained the lightweight caching server, which can be compared to our server. The lightweight caching server uses adaptive TTLs which are adjusted values according to the ages and the access rates of the objects, and they didn't explain the selection algorithm for the registered objects. In this paper, we assumed the TTLs (or the polling periods) are fixed, hence we can't compare the servers directly. However, we can guess that the network traffic will be seriously increased at the bottleneck if lightweight caching server is used, because all the verification messages should pass through the bottleneck. In Figure 4, we showed the traffic (denoted by LCS (fixed TTL)) when lightweight caching server is used, assuming the objects have the fixed TTLs and the server uses the same simple selection algorithm of ours. We guess that it will show the similar results even with the adaptive TTLs and we are studying about various cases including it. For other metrics, LCS (fixed TTL) shows almost the same results to those of our mechanism.

Before finishing this section, we should add some more comments on the negative results for the server load and the network traffic. The latency time in the web environment is not directly related with the server load. [9] showed that the *byte-latency* is nearly identical even between the servers whose loads are 20 times different. It means that the latency on the web hardly depends on the server load. Related with the bandwidth usage, we should consider the bandwidth at the bottleneck is much valuable than that beyond the bottleneck. We think that even a small traffic reduction at the bottleneck compensates the traffic increase beyond the bottleneck.

## 4. Future works

In this section, we mention some of the ongoing works related with the suggested server structure.

### Adaptive TTL

In this paper, we used the same TTL values and polling periods for all the objects. However, many consistency mechanisms have shown better results by using different TTL values according to the ages and the access rates of the cached objects. If we use such an adaptive TTL for the objects, we may reduce the network bandwidth and the server load less.

### Policy for the polling

We have shown two polling policies: polling all the cached objects, and polling the objects from their second requests. However, if we use some intelligent policies, we may reduce the network traffic beyond the bottleneck and the web server load with still notably fast response times. When the fixed TTL is used, the access rate can be the only major factor for the selection. However, when the adaptive TTL is used, both the modification rate and the access rate can be the major factors. Currently, we are studying about the relation between the factors and the optimal cost-effective selection algorithm considering the network bottleneck.

### Usage of the polling server

We assumed that the polling server has just one function which periodically checks the validity of the cached objects. However, we can add some more functions to the polling server, such as the prefetching. Some prefetching mechanisms can be used to reduce the response time and network traffic during the peak time. They may prefetch the objects using peak times difference between the time zones and between the continents. It is also possible to prefetch the stale objects, which can help reduce the number of stale misses.

## 5. Summary and Conclusions

In this paper, we proposed a proxy server structure which improves the response times at the network bottleneck such as the inter-continental backbone. For this purpose, we put the polling server beyond the bottleneck, which periodically checks the validities of the cached objects and preserves the consistency. We showed by simulations that we could get faster response time and reduce the network traffic at the bottleneck.

On the contrary, our server structure can increase the network traffic beyond the bottleneck and the loads of the origin servers. However, the bandwidth beyond the bottleneck is less valuable than that of the bottleneck. And it is known that the server load hardly affects the response times.

There are some more topics to study related with this server structure. Currently we are studying about the selection algorithms for the polled objects considering the network bottleneck, and some more functions which can be added to the polling server, like the prefetching.

## References

- [1] M. Abrams, C. R. Standridge, G. Abdulla, S. Williams, and E. A. Fox. Caching proxies: Limitations and potentials. Technical report, Dep. of Computer Science, Virginia tech., 1995.
- [2] P. Cao and C. Liu. Maintaining strong cache consistency in the world-wide web. In *Proceedings of the 1997 International Conferences on Distributed Computing Systems*, 1997.
- [3] A. Chankhunthod, P. Danzig, C. Neerdaels, M. F. Schwartz, and K. J. Worrel. A hierarchical internet object cache. In *Proceedings of the USENIX 1996 Annual Technical Conference*, 1996.
- [4] A. Dingle and T. Partl. Web cache coherence. In *5th International World Wide Web Conference*, 1996.
- [5] J. Gwertzman and M. Seltzer. World wide web cache consistency. In *Proceedings of the USENIX 1996 Annual Technical Conference*, 1996.
- [6] F. R. G. J., M. J., F. H., and B. L. T. *Hypertext Transfer Protocol-HTTP/1.1*. RFC 2068, 1997.
- [7] B. Krishnamurthy and C. E. Wills. Study of piggyback cache validation for proxy caches in the world wide web. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, 1997.
- [8] B. Liu, G. Abdulla, T. Johnson, and E. A. Fox. Web response time and proxy caching. Technical report, Virginia Tech., 1998.
- [9] S. Manley. Web facts and fantasy. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, 1997.
- [10] J. C. Mogul, F. Douglis, A. Feldmann, and B. Krishnamurthy. Potential benefits of delta-encoding and data compression for http. In *ACM SIGCOMM'97*, 1997.
- [11] M. Nabeshima. The japan cache project: An experiment on domain cache. In *Proceedings of 6th International WWW conference*, 1997.
- [12] D. Wessels. Squid frequently asked questions. <http://squid.nlanr.net/Squid/FAQ>.
- [13] D. Wessels. Intelligent caching for world-wide web objects. In *Proceedings of INET-95*, 1995.