Transparent En-Route Caching in WANs?

P. Krishnan[*] *Danny Raz* *Yuval Shavitt*
{pk, raz, shavitt}@bell-labs.com
Bell Laboratories, Lucent Technologies
101 Crawfords Corner Road
Holmdel, NJ 07733-3030

Abstract

Hierarchical caching is being widely used today to improve web performance. The way hierarchical caches are deployed requires clients and caches to be aware of the location of nearby caches. This creates management and configuration issues, and may also lead to performance bottlenecks. In contrast, Transparent en-route caches (TERC) are devices that are placed at appropriate locations within the network and work obliviously. A TERC intercepts web requests and web data. If the requested item exists in the TERC's memory, the data is sent to the requester, otherwise the request is transparently forwarded up the routing path.

TERCs have the potential to be more efficient than plain proxy caches. In reducing latency, they provide the twin benefits of improving latency through cache hits and not introducing additional delay in re-routing requests from one cache to another. TERCs can also be deployed in a subnet without any cooperation from clients or the providers, and since they see all the web traffic, their effectiveness is enhanced. Deploying TERCs in LANs and WANs pose different issues. In this paper, we discuss issues like deployment and implementation, cache location, and hit rate variation, in evaluating a case for TERCs in WANs. We conclude that TERCs in WANs are a promising technique.

[*]Corresponding author, pk@bell-labs.com.

1 Introduction

The benefits of caching in the world wide web, namely, saving network bandwidth, improving client latency, and reducing server load, are well understood [7, 2]. In current deployments, popular locations for caches are at the edge of networks in the form of browser and proxy caches, the end points of high latency links, or as part of cache hierarchies [5, 3]. In cache hierarchies, the caches are configured to know their neighbors. Significant research has gone into optimizing cache performance [5, 21, 2, 6], co-operation among several caches [5, 13, 17, 10, 9], and cache hierarchies [3, 5, 1]. Web servers are also replicated to achieve load-balancing.

In this paper, we study the issues in using *transparent en-route caches* (TERCs) in WANs. When using TERCs, only caches that are located along routes from a client to a server are used, transparently to the servers and clients. An en-route cache intercepts any request that passes through it, and either satisfies the request or forwards the request towards the server along the *regular routing path*. The typical arrangement we envisage is for the caches to be co-located with routers. They could potentially be maintained by network providers, who could provide a better service without increasing the capacities on their links without requiring cooperation from its clients. Such a model requires literally no change to the existing infrastructure and standards, except the introduction of caches in the network at appropriate locations.

TERCs have several potential advantages. First, they are easier to manage than replicated web servers and hierarchical caches because they do not need to co-operate with other servers, proxies, or clients. The issue of configuring and re-configuring proxies to know about their hierarchy is not present with TERCs. Second, since the normal routing path of packets is essentially undisturbed, no additional latency is introduced due to the re-routing of packets from one cache to another. Note that in en-route caching, the requests may not be served by the "closest" cache, since we are not tampering with the regular routing of packets. Another advantage of TERCs over proxy caches is in cases of cache overloading. If the cache is too congested it becomes the bottleneck of the system, increasing the delay of data arrival. A smart TERC can sense that it is overloaded, and stop intercepting requests. TERCs are also more cost effective since they can be placed in locations where they give optimal yield in terms of shortening delay or lowering data flow.

TERCs, or approximations of TERCs, have been deployed in LANs effectively (using Level-4 switches and proxy caches, for example). However, deploying TERCs in WANs introduces several different issues. In this paper, we discuss some of these issues and present some results and observations which suggest that it may be a promising technique.

2 TERCs vs. Proxies

The current practice in using caches is to put proxy caches at the network edges. To benefit from the proxy caches, the users must set the "proxy cache" option on their browsers. In some cases, administrators force the use of proxy caching by filtering out all the HTTP traffic not originating from the cache. TERCs, on the other hand, are transparent to the user and require no cooperation whatsoever. Another advantage of TERCs over proxy caches is in cases of cache overloading. If the cache is too congested it becomes the bottleneck of the system, increasing the delay of data arrival. Such a proxy would like to signal some of its client to stop using it, and use some other proxies (or go directly to the servers). However, in the current deployment scheme this is impossible, since the choice of the proxy is done at the browser. A smart TERC, however, can sense that it is overloaded, and stop intercepting requests. TERCs are also more cost effective since they can be placed in locations where they give optimal yield in terms of shortening delay or lowering data flow.

Another important point is that when proxies, and even more when proxy hierarchies are used, the end to end delay may increase rather than decrease. This can happen not only because of the response time of the proxies, but also due to the fact that the route from the client to the server (if the page was not found in any of the cashes), may be longer than the regular Internet path, from the client to the server. Pros and cons of using TERCs vs. proxies have also been presented in [20].

3 Hardware Issues

The feasibility of building TERCs depends on the ability to perform high speed filtering, and high speed table look-up. These two fields have seen major improvements in the last few years, making implementation feasible at backbone rates. Implementation at LAN speeds have been available for some time now.

The emerging next generation of routers exhibit both high performance and rich functionality, such as support for virtual private networks and QoS [13, 14]. To achieve this, per flow queueing and fast IP filtering are incorporated into the router's hardware [14, 16]. An example of products that provide such functionality today is Lucent's PacketStar 6400 IP switch[1] that incorporates the above features in hardware to support up to 128 T3 rate line or 128 100-BaseT lines.

One of the major problems in implementing a fast caching engine, either a proxy or a TERC, is the need to handle many open TCP connections. Techniques such as TCP splicing [18] alleviate the load from the proxy and improve performance. Another problem is the extensive use of memory copies that might reach the memory limit. Advanced buffer management techniques [4] could be used to eliminate some data copies.

To verify the deployability of TERCs, and further investigate the problems involved, we built a TERC prototype under FreeBSD using the Squid proxy caching server [3]. Although we built a low speed software device, all the components of our solution are available in some high speed backbone routers, which indicates that high speed TERCs can be built.

4 Deployment

While TERCs in WANs could be deployed at the gateway between networks, we have observed significant benefits in not doing so. (This issue of TERC location is discussed briefly in Section 6, and was the subject of another paper [12].) For maximal benefits, we must have the ability to place TERCs with any node in the network. This introduces some new problems, not found in LAN environments (where TERCs are typically placed at the edge of the LAN).

[1] see `www.lucent.com/dns/products/ps6400.html`.

There are two approaches to deploying TERCs. In the first approach, a TERC hijacks the HTTP session. In other words, when an HTTP connection is opened (a SYN packet is sent), it is intercepted by a TERC (or a router associated with the TERC), that emulates the server towards the client. If the data is not available in the TERC's memory, it opens a new HTTP connection to the server to retrieve the data. (This is similar to what is done in LANs with Level-4 switches and is easier to implement.) A subtle issue is that since the normal routing path from the server to the client might not be through the TERC, from a purist's point of view the TERC is not "en-route." However, from the latency point of view, we don't expect a significant difference between the two routing paths.

Another possibility for deployment is for the TERC to snoop on packets, and not disrupt the initial SYN packet from getting to the server. Upon seeing the GET request, the TERC hijacks the connection if it can satisfy it. (The TERC can snoop and store the data from the server's reply.) This solution is sensitive to the normal routing path from the server to the client passing through the TERC. It also has the undesired effect of the server seeing SYNs that may be eventually tackled by the cache.

In both deployment scenarios, there is a relationship to routing that is important. Due to Internet routing characteristics, the path that successive packets take to the server may not go through the same TERC. This phenomenon is not very likely, as discussed in the next section.

5 Stability

As described earlier in Section 4, the effectiveness of TERCs depends on the stability of Internet routing. Since a TERC is snooping on TCP packets, a route change may result in disconnections. Paxson's measurements [19] suggest that most routes remain essentially stable. Labovitz et al. [15] studied BGP route changes and concluded, similarly, that in reality Internet routing is stable. They found that 80% of the routes change at a frequency lower than once a day. To verify the Internet routing stability in the context of caching we measured the short term stability of Internet routes using the following simple experiment. We performed three consecutive `traceroute` from Bell-Labs to 13533 destinations. On the average the time between the start of the first traceroute and the last was about one minute. Initially, we found that over 90% of the routes did not change during that period. Using equivalences (eliminating differences that are due to multiple interfaces of the same router) we observed that almost 93% of the routes are actually stable in our measurements. We expect the real number to be higher, since:

1. Our equivalence include only interface pairs we could positively identify as equivalent and we believe that we missed many more.

2. Due to the packet re-orderings caused by route changes, many ISPs implement route caching for TCP connections, i.e., even when routing entries change, existing "connections" use the old routing path. The route caching did not affect our measurements as `traceroute` uses UDP packets. Note that fluttering (or rapidly-oscillating routing) if not combined with route caching can create problems for TERC effectiveness; however, fluttering creates many performance problems for TCP in general [19].

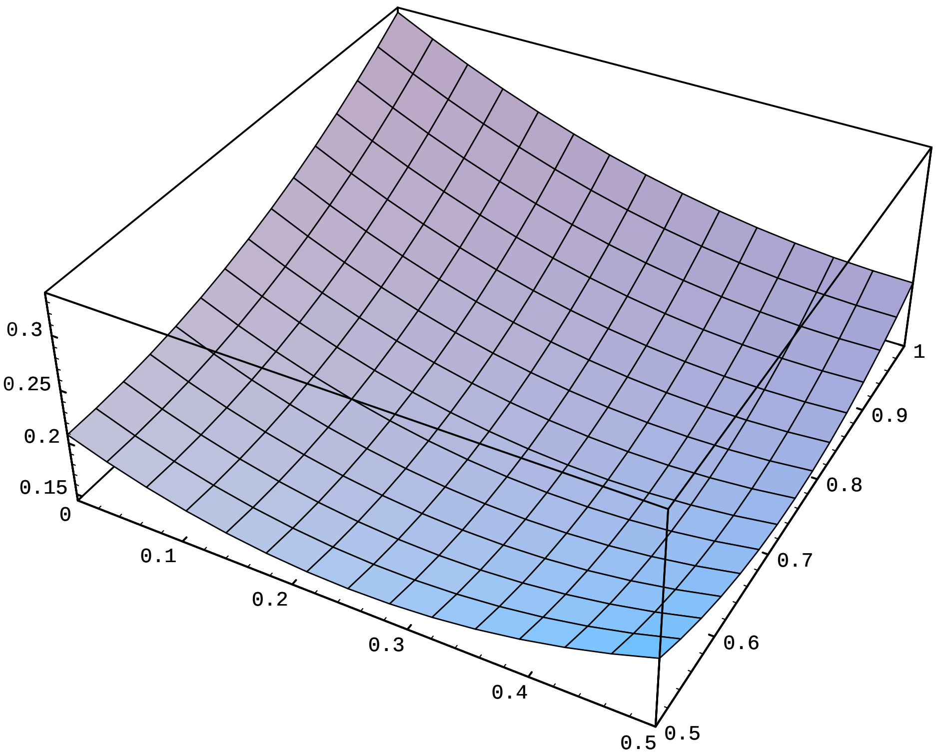

Figure 1: Flow as a function of the location of two caches on a homogeneous [0,1] line. The caches are placed in [0, 0.5] and [0.5, 1.0], respectively.

6 Location

An important issue in en-route caching is determining the appropriate locations for the caches. The goal is to optimize the gain for the system by minimizing the overall traffic in the network, and reducing the average delay to the clients. A simple example, used purely for illustration purposes, shows how the cache location affects the gain. Consider a line network where Web servers and clients are homogeneously distributed. For simplicity, assume the line is continuous and that its length is one unit. Figure 1 shows how the overall flow in this line changes as a function of the location of two TERCs. Note that if the TERCs are placed on the edges, no advantage is gained while when they are placed optimally (at locations $\frac{1}{3}$ and $\frac{2}{3}$) the flow (and also the average delay) is reduced by about 50%. (An experimental study is presented later in this section.)

The rest of this section contains a summary of an extensive study we did on the cache location problem as reported in [12].

Trying to find the best location for caches by an accurate simulation using detailed logs of web activity is computationally infeasible. We formulated the cache location problem by looking at the network as a graph, and modeling the flow of data from servers to clients as flows on this network graph. These flows are affected by en-route caches, in that a client request that can be satisfied by a TERC is not propagated to the server. In [12], we studied the problem of optimally locating en-route

cache type	*Number of caches*	*Overall Traffic*
Boundary Proxies	360	70.4%
Backbone TERCs	2	79.6%
Backbone TERCs	5	74.2%
Backbone TERCs	12	72.4%

Table 1: The overall traffic by using optimally placed TERCs and by putting proxy caches on the boundary. The hit ratio of 30%.

caches for the case of a web server that wishes to minimize the average access delay for its clients. This case translated to solving the TERC location problem on a tree, with the server at the root of the tree. We presented a computationally-efficient dynamic-programming algorithm for this case with computational complexity $O(n \cdot h \cdot k)$, where n is the number of nodes in the tree, h is its height, and k is the number of caches to be placed in the tree. We experimentally validated our algorithms for two Bell Labs web servers, `www.bell-labs.com` and `www.multimedia.bell-labs.com`. The tree structure was derived by performing `traceroute`s from the respective web servers to the accessing hosts, and the flows were obtained from the access logs of these servers.

We observed that a small number of TERCs were sufficient to reduce the network traffic significantly. We also observed that the optimal location for caches when considering one server's traffic is *not* at the edge of the individual networks, where providers currently tend to place them. At the time of the experiment, the `www.bell-labs.com` server was connected to BBNPLANET[2]. There were about 360 relevant points at which the BBNPLANET network was connected to different parts of the Internet, so putting caches on the edge would have required about 360 caches in this case. This reduces the cacheable traffic inside this network by almost 30%, assuming a 30% hit ratio, since almost all of the the requests for web pages come from outside the BBNPLANET network. However, just five TERCs at the right places would have reduced this traffic by 25.8%. Table 1 shows the savings in overall web traffic obtained by deploying different numbers of TERCs, assuming a hit ratio of 30%.

Although the optimal locations change slightly with time, we can use cache locations calculated from recent history data with little penalty in performance. The results of this study created the issue that for benefiting from TERCs in WANs, it is advisable to study the deployment issues mentioned in Section 4.

7 Hit Rate Variation

An issue that comes up with TERCs is: How does the hit rate vary along the routing path. (This issue does not exist with hierarchical caches, since the cache hierarchy determines the hit ratio.) To check this, we used the data and technique as described below.

[2]BBNPLANET is now part of GTE.

Data. As also used in [12], we collected data from web server `www.bell-labs.com`: a medium size site, that receives about 200K–300K cacheable (i.e., non-cgi) requests a week. We considered two weeks of server logs from the server corresponding to "non-holiday" periods. Over these two weeks, an average of 14000 unique hosts per week accessed the server and 1Gbytes of cacheable data per week were retrieved. Note that requests to the web server were post proxy cache, and hence the traffic the servers see has already passed, in part, through existing caches.

To obtain the network graph (in this case, the tree along which data is sent from the server to the client) we did the following: For each of the unique hosts that accessed the servers, we ran `traceroute` from the respective server to the hosts. Although routing tables change over the course of time, and with network upgrades, we expect the traceroute information to provide a reasonably good picture of the network graph as seen by the web server. In an ideal world with an unchanging network and perfect shortest path routing, we would get a tree rooted at the web server by putting together the traceroute information. What we obtained, however, was not a tree due to several reasons. Some of the routers have several parallel links and multiple interfaces that make the network graph a directed acyclic graph (DAG) rather than a tree. This was easily corrected by using equivalence lists that specified the multiple interfaces for a router. The more difficult problems occur due to destinations that had two routes with similar cost, and were therefore both observed by our traceroutes. This phenomena was observed mostly in the traceroutes from server `BL`, since it was bi-homed through two ISPs (BBNPLANET[3] and ALTERNET) during the time of the experiments; some backbones like MCI that the traceroute traversed also gave rise to this phenomenon. We reduced the DAG to a tree structure by breaking multiple paths to a node, by choosing the path through which more bytes traversed. The trees that we obtained had about 32,000 nodes for server `BL`.

For the hit ratio we used our traceroute tree from `www.bell-labs.com`, and the requests from the users. For each node v, we count $n_v(x)$, the number of times a page x was routed through node v in the tree in a week. We assume that R times in a week this page should have been refreshed (e.g., due to aging). Using this, the calculated hit ratio for node v is

$$p_v(R) = \frac{1}{f_{s,v}} \sum_{\text{all pages } x} \max\{n_v(x) - R, 0\} \cdot ||x||,$$

where $||x||$ denotes the size of page x, and $f_{s,v} = \sum_{\text{all pages } x} n_v(x) \cdot ||x||$.

We check in two ways how the hit ratio changes. We first calculate the average hit ratio for each depth, which, as depicted in Figure 2, is quite stable from depth 5 and on.

A more meaningful result is depicted in Figure 3. Here we show the hit ratio for the 50 nodes with maximal flow. These are the nodes that are most likely to have caches assigned to them (assuming the number of caches is below 20). With a few exceptions, the hit ratio for the nodes is between 20% and 35%. The first few nodes that exhibit high hit ratio are the ones that are very close to the server.

[3]BBNPLANET is now part of GTE.

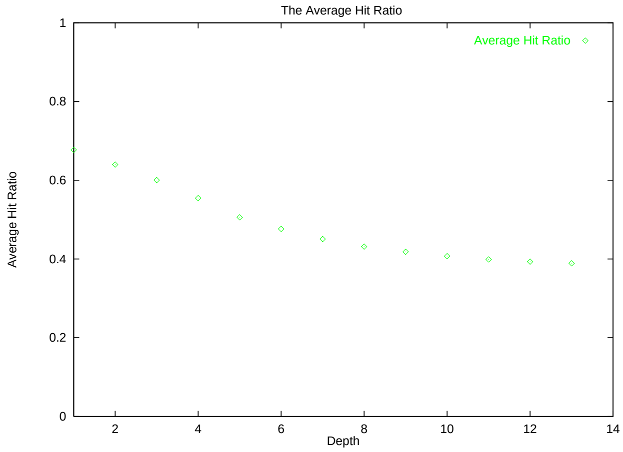

Figure 2: The average hit ratio vs. depth for $R = 7$.

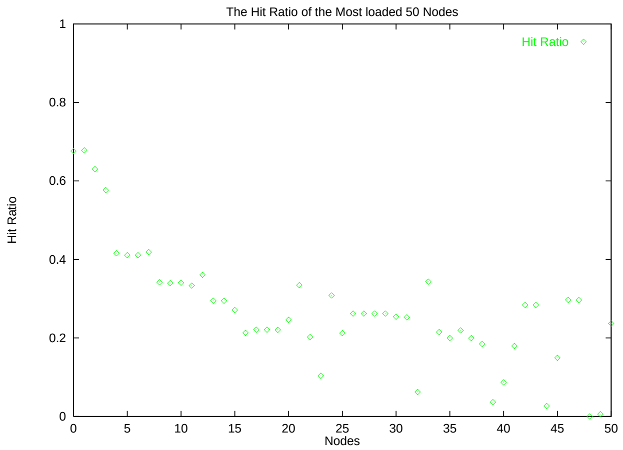

Figure 3: The hit ratio for the 50 nodes with maximal flow for $R = 7$.

In combination with other proxy caching studies, our results hint that we should see roughly the same improvement in latency due to hit rate improvement with TERCs as with cache hierarchies. The additional possible benefit in latency reduction due to non-redirection of requests from the routing path has not yet been quantized.

8 Related Work

Transparent en-route caches were suggested for load balancing by Heddaya and Mirdad [11]. They point to the use of run-time code generation techniques to dynamically download high performance packet filters to the kernel [8]. Zhang et al. [22] also advocated the use of en-route caches, but their focus is on using multicast for data dissemination to the caches. Danzig et al. [7] had observed the advantage of placing caches inside the backbone rather than at its edges. They showed that the overall reduction in network FTP traffic is higher with caches inside the backbone (core nodes) rather than with caches on the backbone edges (external nodes).

9 Discussion

We have presented the issues and some possible solutions for placing transparent caches in wide area networks. The advantages of simpler cache configurations, and gaining from not having to re-route requests make this an interesting proposition. Additionally, the ability to deploy such a solution in an ISP subnetwork locally and independently is valuable. We have observed that there are several issues in wide area networks that are different from LANs while deploying TERCs. Our studies indicate that based on advances in the last few years, TERCs in WANs should be feasible.

Several problems are still open, for which we have presented some observations. One is the issue of deployment and interaction of TERCs with Internet routing. The second is to accurately quantize the effect on hit rate with TERCs in WANs.

References

[1] A distributed testbed for national information provisioning. `http://www.nlanr.net`.

[2] Nlanr web caching workshop. `http://www.nlanr.net/Cache/Workshop97/`.

[3] Squid internet object cache. `http://www.nlanr.net/Squid`.

[4] J. C. Brustoloni and P. Steenkiste. User-level protocol servers with kernel-level performance. In *IEEE INFOCOM'98*, pages 463 – 471, 1998.

[5] A. Chankhunthod, P. B. Danzig, C. Neerdaels, M. F. Schwartz, and K. J. Worrell. Hierarchical internet object cache. In *Usenix Technical Conference*, 1996.

[6] E. Cohen, B. Krishnamurthy, and J. Rexford. Improving end-to-end performance of the Web using server volumes and proxy filters. In *SIGCOMM*, pages 241 – 253. ACM, Sept. 1998.

[7] P. B. Danzig, R. S. Hall, and M. F. Schwartz. A case for caching file objects inside internetworks. In *SIGCOMM*, pages 239 – 243. ACM, Sept. 1993.

[8] D. R. Engler and M. F. Kaashoek. Dpf: Fast, flexible message demultiplexing using dynamic code generation. In *ACM SIGCOMM'96*, Aug. 1996. Stanford University, CA, USA.

[9] L. Fan, P. Cao, J. Almeida, and A. Z. Broder. Summary cache: A scalable wide-area Web cache sharing protocol. In *SIGCOMM*, pages 254 – 265. ACM, Sept. 1998.

[10] S. Gadde, M. Rabinovich, and J. Chase. Reduce, reuse, recycle: An approach to building large internet caches. In *Proceedings of the 1997 Conference on Hot Topics in Operating Systems*, 1997.

[11] A. Heddaya and S. Mirdad. Webwave: Globally balanced fully distributed caching of hot published documents. In *17th IEEE International Conference on Distributed Computing Systems*, May 1997. Baltimore, MD, USA.

[12] P. Krishnan, D. Raz, and Y. Shavitt. Transparent en-route cache location. 1998. submitted.

[13] P. Krishnan and B. Sugla. Utility of co-operating web proxy caches. In *Proceedings of the 7th International World Wide Web Conference*, Brisbane, Australia, Apr. 1997.

[14] V. P. Kumar, T. V. Lakshman, and D. Stiliadis. Beyond Best Effort: Router Architectures for the Differentiated Services of Tomorrow's Internet. *IEEE Communications Magazine*, 36(5):152–164, May 1998.

[15] C. Labovitz, G. R. Malan, and F. Jahanian. Internet routing instability. In *ACM SIGCOMM'97*, Aug. 1997.

[16] T. V. Lakshman and D. Stiliadis. High speed policy-based packet forwarding using efficient multi-dimentional range matching. In *SIGCOMM'98*, Sept. 1998.

[17] R. Malpani, J. Lorch, and D. Berger. Making world wide web caching servers cooperate. In *Proceedings of the Fourth International World Wide Web Conference*, Dec. 1995.

[18] D. Maltz and P. Bhagwat. TCP Splicing for Application Layer Proxy Performance, Mar. 1998. IBM RC 21139.

[19] V. Paxson. End-to-end routing behavior in the Internet. *IEEE/ACM Transactions on Networking*, 5(5):601 – 615, Oct. 1997.

[20] B. Williams. Transparent web caching solutions. In *Proceedings of WCW'98*, June 1998.

[21] S. Williams, M. Abrams, C. R. Stanridge, G. Abdulla, and E. A. Fox. Removal policies in network caches for World-Wide Web documents. In *ACM SIGCOMM'96*, 1996.

[22] L. Zhang, S. Floyd, and V. Jacobson. Adaptive web caching. In *NLANR Web Cache Workshop*, June 1997. `http://www-nrg.ee.lbl.gov/floyd`.