The Cache Location Problem

P. Krishnan, Danny Raz, and Yuval Shavitt

Abstract— **This paper studies the problem of where to place network caches. Emphasis is given to caches that are transparent to the clients since they are easier to manage and they require no cooperation from the clients. Our goal is to minimize the overall flow or the average delay by placing a given number of caches in the network.**

We formulate these location problems both for general caches and for transparent en-route caches (TERCs), and identify that in general they are intractable. We give optimal algorithms for line and ring networks, and present closed form formulae for some special cases. We also present a computationally efficient dynamic-programming algorithm for the single server case.

This last case is of particular practical interest. It models a network that wishes to minimize the average access delay for a single web server. We experimentally study the effects of our algorithm using real web server data. We observe that a small number of TERCs are sufficient to reduce the network traffic significantly. Furthermore, there is a surprising consistency over time in the relative amount of web traffic from the server along a path, lending a stability to our TERC location solution. Our techniques can be used by network providers to reduce traffic load in their network.

I. Introduction

Caching improves network and system performance for WWW browsing by saving network bandwidth, reducing delays to end clients, and alleviating server load [13], [29]. Currently, the popular locations for caches are at the edge of networks in the form of browser and proxy caches, the ends of high latency links, or as part of cache hierarchies [8], [31]. Significant research has gone into optimizing cache performance [8], [33], [29], [10], co-operation among several caches [8], [23], [26], [17], [15], and cache hierarchies [31], [8], [28]. Web servers are also replicated to achieve load-balancing.

Placing caches inside the network is becoming more popular [34], [35], [13], [20]. Danzig et al. [13] 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). Their study was based on data from early 90's NSF backbone traffic. A multicast-based approach to adaptive caching in the network was also proposed recently [34], [35]. Heddaya and Mirdad [20] have proposed the use of network caches for load balancing. Clearly, how well caching inside the network will work depends on where the caches are located, and how data is disseminated to them.

P. Krishnan is with ISPSoft, Inc., 106 Apple Street, Suite 301, Tinton Falls, NJ 07724, USA. pk@ispsoft.com. This work was done while the author was with Bell Laboratories.

D. Raz is with Bell Laboratories, Lucent Technologies, 101 Crawfords Corner Rd., Holmdel, NJ 07733, USA. raz@research.bell-labs.com.

Y. Shavitt is with Bell Laboratories, Lucent Technologies, 101 Crawfords Corner Rd., Holmdel, NJ 07733, USA. shavitt@ieee.org

This paper studies the cache location problem with an emphasis on *transparent en-route caches* (TERCs). When using TERCs, caches are only located along routes from clients to servers, and are placed 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*. In the typical arrangement caches are co-located with routers or with L4 switches, and are maintained by network providers, who provide a better service without increasing the capacities on their links. Such a model has significant operational benefits since caches can be introduced easily into the existing infrastructure [9]. Almost all existing caching products include a transparent operation mode [6], [7]. TERCs are easier to manage than replicated web servers since they are oblivious both to the end-user and the server. It is important to 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. The effectiveness of TERCs depends on the Internet routing stability during the connection lifetime of an HTTP session. Our measurements (see Section V-B.4) and results by others [30], [24] suggest that for the short duration of an HTTP connection routing is mostly stable.

Our 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. With appropriate costs placed on the network edges, we can capture the general model of links with different bandwidths. Trying to find the best location by an accurate simulation using detailed logs of web activity is computationally infeasible. Hence, we formulate our 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. We associate a hit ratio with each flow to represent the performance of the caching algorithm with respect to it. This is because flows may have significantly different cachability, e.g., some flows may be coming from regional caches and thus maybe less cachable than those coming directly from end users.

In general, optimizing the location of k caches in the network graph for criteria like minimum average delay is intractable. The proof follows via reduction from the well-known p-median problem [18]. However, for some special cases an optimal solution can be found in polynomial time. We present optimal solutions for the line and ring topologies, and present closed form formulae for some special cases.

One particularly interesting case is a tree network with a single source. This is the case of a web server that wishes to minimize the average access delay for its clients. We present a computationally-efficient dynamic-programming algorithm for this case. The computational complexity is $O(n \cdot h \cdot k)$, 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.

In practice, an ISP that allocates a budget for k caches wants to place them in the locations that will minimize the traffic in its network. As pointed out by Breslau et al. [5] most of the traffic generated in the Internet comes from a handful of very popular web servers. Thus, using our algorithm for each of these popular sites will reduce the traffic significantly. In Section VI we discuss this problem and propose a more general heuristic solution.

We experimentally validate 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 observe that a small number of TERCs, when placed at optimal locations, are sufficient to reduce the network traffic, significantly. These optimal locations are *not* at the edge of the individual networks, where providers currently tend to place them. We also compare our optimal algorithm against a greedy algorithm and report on their relative performance.

An important issue with the practical impact of our results is the optimal solution "stability." If the optimal cache locations vary significantly over time it may make any off-line solution insignificant. We make two important observations. First, although the intersection between the sets of clients accessing the web server at different times can be very small, the relative traffic from the server along the different tree paths remains relatively stable. Due to this, we observe that although the optimal cache locations may change slightly over time, we can use cache locations calculated from recent history data with little penalty in performance. We expect that the results of this paper could be used constructively in deploying network caches.

The rest of the paper is organized as follows. In Section II, we present our model in detail, and describe the computational complexity of the problem. Section III studies the line and ring topologies. We present our algorithms for the web server case in Section IV, nd the experimental methodology and results in Section V. We conclude with discussions in Section VI.

A. Related Work

Transparent en-route caches were suggested for load balancing by Heddaya and Mirdad [20]. They point to the use of run-time code generation techniques to dynamically download high performance packet filters to the kernel [14]. Zhang et al. [34] also advocated the use of en-route caches, but do not address the placement problem.

In a typical transparent cache implementation [7], [6], traffic is filtered by a designated L4 switch and diverted to the cache. Commercial products that are built for transparent operation and aim at the ISP market include Cisco's CacheEngine, InfoLiberia's DynaCache 200, CacheFlow products, ArrowPoint CS series, and Lucent's IPWorX.

Zhang et al. [34] proposed an adaptive web caching structure using multicast for data dissemination to the caches. Methods similar to the one we present could also be used to optimally place their adaptive caches. Cunha [12] studied a similar location problem in the context of push servers [3] and load balancing. He presented simple local heuristics for load balancing that also reduce the overall traffic in the network; however, his work does not address the optimal location problem.

Our dynamic programming solution is similar to the one used by Tamir [32] for solving the p-median problem. This dynamic programming method can be used for solving the non-transparent cache location problem where requests are routed to the closest cache. However, it works only if the entire network were a tree, the number of sources were one, and the routing infrastructure were cache-aware.

II. Model and Definitions

In this section, we first present the model for general cache location, and then present the TERC location problem. We consider a general wide area network, where the internal nodes are routers and the external nodes are either servers, clients, or gateways to different subnets. A client can request a web page[1] from any of the servers, and the server, v_s, sends this page to the client, v_c, on the shortest path from the server to the client. When caches are present, a client can request the page from a cache, v_k, rather than from the server. If an up-to-date copy of the requested page is in the cache's local memory, the page is delivered to the client. Otherwise, the cache contacts the web server, refreshes its local copy, and sends the page to the client. Current protocols allow caches to validate the freshness of locally stored data [2], [16]. The performance of a caching scheme is a function of the network topology, the request pattern, the assignment of caches to requests, the cache sizes, and the cache replacement algorithms used.

Our goal here is to describe a model that will be clear and easy to realize, while at the same time maintain the essential parameters of the problem. We refer to the bytes sent to a client as the *flow* to the client. Our main goal is to find good locations for the caches. Caches are generally characterized by their *hit ratio*, which is the fraction of data that can be served from the cache's local memory. A higher hit ratio implies a lower load on the network, and much work has been done to improve cache hit ratios [33], [29], [1]. It has been shown [4], [5] that amongst all pages on a server only a small fraction are very popular, and our measurements support this observation. In other words, many clients request a small subset of pages from a server, and with high probability, these popular pages will be stored in most caches, accounting for most of the cache hits. Therefore, in our model, we make a simplifying "full dependency" assumption; i.e., if a page will be found in

[1] We use the term web page to denote any requested entity.

any cache, it will be found in the first cache on the way to the server.

We associate each *client flow* f with a single number, p_f, that is the cachability of this flow. In other words, p_f is the fraction of the flow that is comprised of the popular pages that are expected to reside in most caches. Therefore, when a flow f with cachability p_f passes through a cache, p_f fraction of the flow is satisfied from its local memory, hence we refer to p_f as the flow hit ratio. When several flows pass through the same node, their effective hit ratio is the weighted average of their individual hit ratios. Note, that the full dependency assumption implies that the upstream flow from a cache has an effective hit ratio of zero. In the simple case where all the flows have the same hit ratio, p, the hit ratio at any node in the network is also p.

The reason caches are placed in the network is to improve performance, primarily in terms of reducing the load on the networks links. From a user point of view, performance is measured by the response time [27], i.e., the time it takes for a page to arrive. This time depends both on the link delays, and on the response time of the servers. In this paper, we consider only the delay due to the logical distance between the client and the server. This delay takes into account the propagation delay and the delay in the routers. Our objective is to minimize the average delay time for the user population weighted by their individual flows. This is equivalent to minimizing the total network flow; i.e., the sum of the web flows taken over all the links.

A. The Formal Model

The above discussion leads to the following formal model. The network is represented by an undirected graph $G = (V, E)$, where $V = \{v_i\}_{i=1}^{i=n}$ is the set of nodes, E is the set of edges, $d(e)$, the length of edge e, reflects the delay caused by this edge, and $d(v_i, v_j)$ is the sum of the link distances along the route between nodes v_i and v_j. We assume that shortest path routing is used. The request pattern is modeled by the demand set F, where $f_{s,c}$ is the flow to v_c or the amount of data (in bytes) requested by client v_c from server v_s, and $p_{s,c}$ is the hit ratio of that flow. We denote by K the set of at most k nodes where the caches are to be placed. The *cost* $c_{s,c}$ (in bytes $\times$ distance) of demand $f_{s,c}$ using a cache in location v_k is

$$c_{s,c} = f_{s,c} \cdot [p_{s,c} \cdot d(v_c, v_k) + (1 - p_{s,c}) \cdot (d(v_c, v_k) + d(v_k, v_s))].$$

An optimal assignment of a cache to a request is to assign cache v_k (or no cache at all) to the request such that the cost $c_{s,c}$ is minimal among all possible v_k in $K \cup \{v_s\}$. As explained earlier, we assume a full dependency of the caches. Hence, this model does not capture hierarchical structures [28], but does capture the push model [3]. Our overall objective here is to find a set K that minimizes the total cost, i.e., the sum C of all the costs, $C = \sum_{f_{s,c} \neq 0} c_{s,c}$.

The above discussion can be formalized as a graph optimization problem in the following way.

Problem II.1: **The general k-cache location problem.**

- Instance: An undirected graph $G = (V, E)$, a set of demands $F : V \times V \to \mathbb{N}$, a set of flow hit ratio $P : V \times V \to [0, 1]$, and the number of caches k.
- Solution: A subset $K \subset V$ of size k.
- Objective: Minimizing the sum of costs:

$$\sum_{s,c} \min_{v_k \in K \cup \{v_s\}} f_{s,c} \cdot [p_{s,c} \cdot d(v_c, v_k) + (1 - p_{s,c}) \cdot (d(v_c, v_k) + d(v_k, v_s))].$$

As pointed out in Section I, the assignment of caches to clients in the Internet creates many non-trivial management problems. This motivates the TERC model where the caches considered for a client request are only those located along the path from the client to the server. The formal description of the TERC location problem is almost exactly like the general cache location problem presented above, with one small difference in the objective function. For the TERC location problem, the minimization in the objection function is taken only over the nodes along the path from the client to the server.

Problem II.2: **The k-TERC location problem.** The formal definition of the TERC k-cache location problem is exactly as the general k-cache location problem (described in Problem II.1), except that the minimization in the objective function is over the set $\{v_k \in (K \cup \{v_s\}) \cap \{path(v_c, v_s)\}\}$.

Observe that the non cachable part of the flows is not effected by the caches, and therefore cannot affect the optimal cache location. We can thus replace each flow $f_{s,c}$ with hit ratio $p_{s,c}$ with a flow of $f_{s,c} p_{s,c}$ and a hit ratio of $p = 1$, as formally proved below.

Theorem II.1: The solution of problem II.2 with the demands $F = \{f_{s,c}\}$ and flow hit ratios $P = \{p_{s,c}\}$ is equivalent to solving the problem for $F' = \{f_{s,c} p_{s,c}\}$ with hit ratio of one.

Proof: Note that

$$\begin{aligned}
&\sum_{s,c} \min_{v_k \in K(s,c)} f_{s,c} \cdot [p_{s,c} \cdot d(v_c, v_k) + (1 - p_{s,c}) \cdot (d(v_c, v_k) + d(v_k, v_s))] \\
&= \sum_{s,c} \min_{v_k \in K(s,c)} f_{s,c} \cdot [d(v_c, v_k) + (1 - p_{s,c}) \cdot d(v_k, v_s)] \\
&= \sum_{s,c} \min_{v_k \in K(s,c)} f_{s,c} \cdot d(v_c, v_s) - f_{s,c} \cdot p_{s,c} \cdot d(v_k, v_s)
\end{aligned}$$

where $K(s, c) = (K \cup \{v_s\}) \cap \{path(v_c, v_s)\}$. The last transition relies on the fact that TERCs are on the path from the client to the server. Since the first term does not depend on the set K, the minimum is achieved for the placement K that maximizes $f_{s,c} \cdot p_{s,c} \cdot d(v_k, v_s)$. The solution for the problem with $F' = \{f_{s,c} p_{s,c}\}$ and a hit ratio of one is given by (by simple substitution)

$$\begin{aligned}
&\sum_{s,c} \min_{v_k \in K(s,c)} f_{s,c} p_{s,c} \cdot d(v_c, v_k) \\
&= \sum_{s,c} \min_{v_k \in K(s,c)} f_{s,c} p_{s,c} \cdot [d(v_c, v_s) - d(v_k, v_s)] \\
&= \sum_{s,c} \min_{v_k \in K(s,c)} f_{s,c} p_{s,c} \cdot d(v_c, v_s) - f_{s,c} p_{s,c} \cdot d(v_k, v_s)
\end{aligned}$$

	line	*Bounded degree tree*	*Tree*	*General Graph*
1 *server*	Poly	Poly	Poly	***NP***
m servers	Poly	***NP***	***NP***	***NP***

TABLE I
THE HARDNESS OF THE k-CACHE AND k-TERC LOCATION PROBLEMS (POLY STANDS FOR POLYNOMIAL TIME, AND NP FOR NP HARD)

Again, the first term does not depend on the set K, and the minimum is achieved for the placement, K that maximizes $f_{s,c} \cdot p_{s,c} \cdot d(v_k, v_s)$. □

Based on Theorem II.1, we assume without loss of generality for the analytical part of the paper (Sections III and IV) that all flows have the same hit ratio which we denote by p. An interesting observation is that the k-TERC location problem is a special case of the general k-cache location problem. This is true since if we assume that the shortest path is unique, and the distance of any other path is at least Δ longer, then choosing for all flows $p \leq \frac{\Delta}{\max d(v_i,v_j)+\Delta}$ forces the caches to be on the path from the client to the server. When the metric is minimum hop (i.e., $d(e) = 1,\ \forall e$) we can simply choose $p \leq \frac{1}{n}$.

B. Hardness Results

Given a set K of cache locations, determining the optimal (possibly non-TERC) cache for each request and computing the total cost can be done in $O(|F| \cdot k + n^2)$ steps, by a straight-forward computation. The real problem is to find the best set K. When k is small this is still tractable by checking all the possible K sets. In general, however, the problem is NP-hard. Table I shows a summary of the hardness results for the k-cache location problem.

Even the simple case where there is only one server in the network, and with $p = 1$ is NP-hard. In fact, it is not too difficult to show that this case is equivalent to the well known p-median problem [21], [18]. The negative result for the case of m servers and a tree graph is for a similar model where the caches are put on the edges of the graph, rather than at the nodes. This corresponds to caches that are related to a specific link. The proof is obtained via a reduction from the multicuts problem [11] and holds even if the trees are binary.

Interestingly, the TERC location problem is computationally as hard as the general cache location problem. The single server general graph case (for k-TERCs) is proved via a reduction from the vertex cover problem and is true for the model in which the caches are put on the edges.

III. REGULAR TOPOLOGIES

In this section, we analytically study some regular topologies. We omit some of the details which appear in [22].

A. Homogeneous line with a single source

The very simple case of a line network graph with a single source and hit ratio $p = 1$, demonstrates some of the difficulties of the cache location problem. We calculate the optimal cache location for this case, and compare it to an intuitive greedy algorithm that places caches on the line iteratively in a greedy fashion, without replacing already assigned caches.

Consider a line with one server (source) at one of its ends and $n-1$ equally active clients at every other node in the line. The $n-1$ links have the same cost, normalized to 1.

The overall flow for the line is simply

$$\mathcal{F} = \sum_{i=1}^{n-1} i = \frac{(n-1)n}{2} \tag{1}$$

When k caches are placed on the line in nodes $t_1, \ldots, t_k$ the flow, $\mathcal{F}$, is given by

$$\mathcal{F} = \sum_{j=1}^{k+1} \sum_{i=t_{j-1}}^{t_j-1} i - t_{j-1} = \sum_{j=1}^{k+1} \frac{(t_j - t_{j-1} - 1)(t_j - t_{j-1})}{2} \tag{2}$$

where $t_0 \triangleq 0$, and $t_{k+1} \triangleq n+1$. Clearly $\mathcal{F}$ is minimized when the t_is are equally spaced, i.e.,

$$t_i^{opt} = \frac{in}{k} \tag{3}$$

and the minimal overall flow is

$$\mathcal{F} = (k+1)\frac{1}{2}\frac{n}{k+1}\left(\frac{n}{k+1} - 1\right) = \frac{n(n-(k+1))}{2(k+1)} \tag{4}$$

We compare the optimal solution with a greedy solution that calculates the optimal location of the ith cache without the ability to change its decision about the location of the $i-1$ previously placed caches. If a greedy approach is taken, the first cache location is optimal. The ith cache is placed in the center of the largest current gap. Interestingly this algorithm gives the optimal location for $2^m - 1, m = 1,\ 2,\ \ldots$, but suboptimal locations for all other cases.

The cost for the greedy solution is given by:

$$\mathcal{F} = \frac{n(n-b)}{2b} - \frac{k+1-b}{b}\frac{n^2}{4b} \tag{5}$$

where $b = 2^{\lfloor \log_2 k+1 \rfloor}$ is the next point after k where the optimal and the greedy algorithms give the same result.

Figure 1 depicts the differences between the cost of the optimal and the greedy solutions. The Y axis shows the cost in relation to the situation when no caches are used. As can be seen in this case, most of the savings is achieved by the first few caches. This phenomenon is also observed in real network structures, as reported in Section V-B.2.

B. Homogeneous line with multiple sources

A more general case is when we have multiple sources. In this section we analyze a line with homogeneous traffic

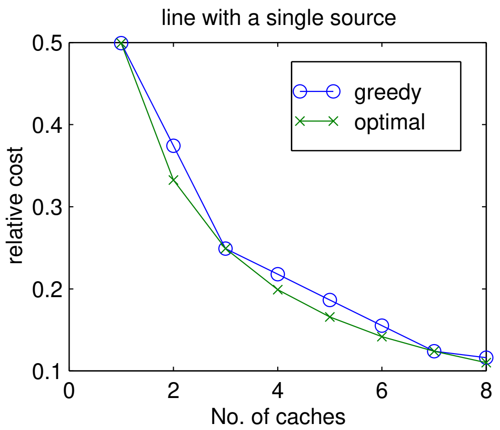

Fig. 1. A line network with a single source and homogeneous client population. We compare the greedy algorithm with the optimal placement.

requirements, i.e., between every possible pair of nodes the traffic requirement is identical. In such a case the flow on the links is bidirectional. We can distinguish between two types of caches: a single interface cache that handles only one directional traffic, and a multi-interface cache that handles all the traffic through a router. We concentrate on the multiple interface cache, and refer the reader to [22] for the single interface cache analysis.

Due to symmetry, when only a single cache is available its position is obviously in the line center. When $k > 1$ we follow [19] and analyze the continuous line rather than a discrete setting. For $k = 2$, the cost of the flow in the line when two caches are put at points t_1 and t_2 ($t_1 < t_2$) is given by:

$$\begin{aligned}
\mathcal{F} &= \int_0^{t_1}\int_r^{t_1} 2(s-r)\,ds\,dr + \int_{t_1}^{t_2}\int_r^{t_2} 2(s-r)\,ds\,dr \\
&+ \int_{t_2}^{1}\int_r^{1} 2(s-r)\,ds\,dr \\
&+ \int_0^{t_1}\int_{t_1}^{t_2} (p+2(1-p))(s-r)\,ds\,dr \\
&+ \int_{t_1}^{t_2}\int_{t_2}^{1} (p+2(1-p))(s-r)\,ds\,dr \qquad (6)\\
&+ \int_0^{t_1}\int_{t_2}^{1} 2(1-p)(s-r) + p((s-t_2)+(t_1-r))\,ds\,dr \\
&= \frac{1}{3} + \frac{p}{2}\left(t_1^2(t_2-2) - t_1(t_2-2)t_2 - t_2(1-t_2)\right)
\end{aligned}$$

Deriving $\mathcal{F}$ by t_2 and comparing to zero yields

$$t_2^{opt} = \frac{1+t_1}{2} \qquad (7)$$

substituting t_2^{opt} in $\mathcal{F}$ and deriving by t_1 gives the optimum at $t_1^{opt} = 1/3$, and substituting t_1^{opt} in Eq. 7 yields $t_2^{opt} = 2/3$.

C. The General Line

In this section we give an optimal solution to the general cache location problem with multiple interface caches on the line, i.e., clients and servers can be located on any node and in any number. We use the full dependency assumption explained in section II. Under this assumption the optimal location does not depends on the hit ratio, thus, for convenience, we assume a hit ratio of one.

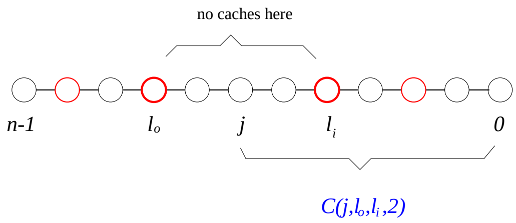

Fig. 2. The definition of $C(j,\ l_o,\ l_i,\ k')$.

Consider a line of n nodes numbered from 0 to $n-1$. The input is the flow requirement from (up to) n servers located at the nodes to (up to) n clients located at the nodes. A node can accommodate both a client and a server. From the input it is easy to calculate the flow requirement on segment $(i-1, i)$, denoted by $FR(i)$.

We use bottom up dynamic programming method, to build an optimal solution to the segment $[0, j]$, from the optimal solution for shorter segments, i.e. $[0, j-1]$. Let $C(j,\ l_o,\ l_i,\ k')$ be the overall flow in the segment $[0, j]$, when k' caches are located optimally in it, and the closest cache to the segment border node from the left (assume node 0 is the right most node) is located at node l_o, and the closest cache to the right is located at node l_i (inside the segment). Figure 2 shows an example of such a segment. Note that $n-1 \geq l_o \geq j \geq l_i \geq 0$, placing caches at the endpoints (0 and $n-1$) will not help, and we do not need to consider the case where $k' > j$.

The overall flow in the optimal k-location problem is $\min_{0 \leq l_i < n} C(n-1, n-1,\ l_i,\ k)$, and what we seek is the location of the k caches in that case. Recall that $FR(i)$ is the flow on the segment $(i-1, i)$. In a similar way $FC(i,\ l_o,\ l_i)$ is the flow on the segment $(i-1, i)$, where the closest caches are at l_o, and l_i, with $n-1 \geq l_o \geq j > l_i \geq 0$. This flow can be easily computed from the input since we assume that the hit ratio p is one. Note that $FR(i) = FC(i, n-1, 0)$.

For the base case, $j = 1$, it is easy to see that for all $n-1 \geq l_i \geq 1$, $C(1, l_i, 1, 1) = FC(1, 1, 0)$, and $C(1, l_i, 0, 0) = FC(1, l_i, 0)$. For $j > 1$, we have:

Claim III.1:

$$\begin{aligned}
C(j,\ l_o,\ l_i,\ k') = \min\{ & \qquad (8)\\
& C(j-1,\ j,\ l_i,\ k'-1) + FC(j,\ j,\ l_i), \\
& C(j-1,\ l_o,\ l_i,\ k') + FC(j,\ l_o,\ l_i)\ \}
\end{aligned}$$

Proof: The optimal placement of k caches in the segment $[0, j]$, can either put a cache at the jth location and $k-1$ caches in the segment $[0, j-1]$, or put all k caches in the segment $[0, j-1]$. Therefor the optimal cost is the minimum cost of these two cases. □

The algorithm now is straight forward: first compute $C(1, l_i, 1, 1)$ and $C(1, l_i, 0, 0)$ for for $n-1 \geq l_i \geq 1$. Next for each $j > 1$ compute $C(j,\ l_o,\ l_i,\ k')$, for all $k \geq k' \geq 0$, and $n-1 \geq l_o \geq j \geq l_i \geq 0$. The complexity of this

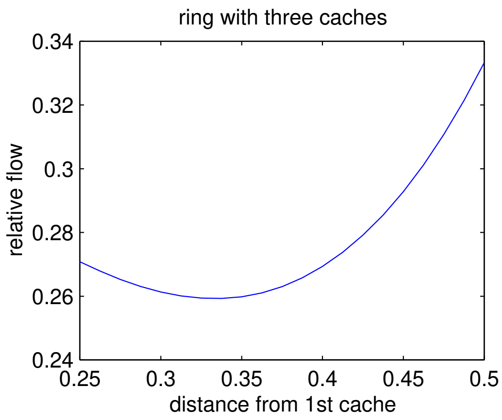

Fig. 3. The relative flow in a ring network with 3 caches as a function of the distance of two caches from the third. (assuming the distances of the two caches from the first are the same.)

algorithms is $O(n^3)$ to compute the base case, and $O(n^3 \cdot k)$ to compute $C(j,\ l_o,\ l_i,\ k')$.

D. Ring Networks

The case of a ring with homogeneous load and caches that cache the data of their two interfaces is straight forward. Due to symmetry considerations, the caches should be placed at equal distance on the ring, regardless of their number, k, or of the hit probability, p.

Figure 3 depicts the relative flow with three caches as a function of the relative location of the caches in a bidirectional ring. Fixing one cache on the ring, the X-axis is the distance the two other caches are placed at relative to the first cache. The optimum is achieved at 1/3, with an almost 75% reduction in traffic. At X=$\frac{1}{2}$, the two additional caches are Co-located. This depicts the traffic gain for two caches, which is a third of the original traffic.

Symmetry considerations are not straightly applied to unidirectional rings (or bidirectional rings with single interface caches). However, regardless of k and p, the caches should still be spread at equal distances to achieve optimal performance. For simplicity we prove the case where $p = 1$.

Putting the first cache in the ring breaks the symmetry. Without loss of generality (w.l.o.g), we can assume the cache is put at location 0. The flow in the ring, when a second cache is put at location $0 \leq x \leq 1$, is given by Expression 9.

$$\begin{aligned}\mathcal{F} &= \int_0^x \int_s^x t - s\, dt\, ds + \int_0^x \int_x^1 t - x\, dt\, ds + \\ &\quad \int_0^x \int_0^s t\, dt\, ds + \int_x^1 \int_x^s t - x\, dt\, ds + \\ &\quad \int_x^1 \int_s^1 t - s\, dt\, ds + \int_x^1 \int_0^x t\, dt\, ds \\ &= \frac{2 - 3\,x + 3\,x^2}{6} \qquad (9)\end{aligned}$$

The optimal location $x = \frac{1}{2}$ is obtained by deriving $\mathcal{F}$ and comparing to zero.

Next we prove that the optimal location of k caches in a unidirectional ring requires the caches to be placed homogeneously. For this end, examine three neighboring caches located at locations 0 (w.l.o.g.), x, and y. It is sufficient to prove that $x = \frac{y}{2}$. The flow in the segment $[0,\ y]$ is given by

$$\begin{aligned}\mathcal{F} &= \int_0^x \int_0^t t - s\, ds\, dt + \int_0^x \int_t^1 t\, ds\, dt + \\ &\quad \int_x^y \int_x^t t - s\, ds\, dt + \int_x^y \int_t^{1+x} t - x\, ds\, dt \\ &= \frac{3\,x^2\,(2-y) - 3\,x\,(2-y)\,y + (3-y)\,y^2}{6} \qquad (10)\end{aligned}$$

The optimal location, $x = \frac{y}{2}$, is obtained comparing the x derivative of $\mathcal{F}$ to zero.

In the more general setting, where the ring is not necessarily homogeneous, we can use our dynamic programming algorithms from Section III-C and from [22]. As we mentioned, the first located cache breaks the ring into a line. Therefore, we can ran these algorithms, for any possible first cache location with an additional factor of $O(n)$ to their complexity.

IV. The Single Web Server Case

The case of optimizing performance for clients of one web server is of particular interest both from the theoretical and practical points of view. Consider a popular server that gets many requests and responds with large quantities of data (like big software and news distribution servers). As the number of requests to the server and the data it serves increase, the performance of the server declines sharply. One way this problem is tackled is to replicate the server. Explicit replication creates problems with managing the different copies and redirecting the client to different locations.

Automatic caching is an attractive proposition. An important question with caching that may have a big impact on the over-all improvement in performance is: where should one put the caches? If they are put very close to the server, the server load may decrease but network congestion will remain a problem. If they are put too close to the clients, there will be a lot of caches, and each cache (i.e., copy of the document) will be under-utilized. Finding the optimal locations for the caches involves looking at both these issues, and translates exactly to solving the k-cache location problem on the network graph induced by the server and its clients.

Most of the web traffic is generated by a small number of servers [5]. Therefore, an ISP that wishes to reduce the traffic in its network can use our algorithm to reduce the traffic to these handful of servers. The same algorithm can also be used by content providers. These are companies that provide hosting services with the promise of fast content delivery to the end-user. Using transparent caches in optimal locations for their clients can minimize the average access delay.

As mentioned in Section II-B, even the case when we have a single server is NP-hard for general networks. We can, however, solve this case on a tree graph. Fortunately, if the shortest path routing algorithm implied by the Internet is stable, the routes to various clients as viewed by any single server should be a tree graph. Thus we can apply an algorithm for the tree graph for the one server case. As we will see in our experiments reported in Section V, some heuristics are needed to apply our algorithm in practice.

We present two algorithms for this problem: A natural greedy algorithm in Section IV-A and an optimal dynamic-programming algorithm in Section IV-C. The solution to the cache location problem depends heavily on the request pattern. One might, therefore, argue that if this pattern is constantly changing, there is no real meaning to an "optimal" cache location. As we will demonstrate in our experimental results in Section V-B, it turns out that this is not true. Although the actual set of clients changes a great deal, the request pattern is stable. In particular, the flows do not change that much at places that really matter, lending stability to the solution.

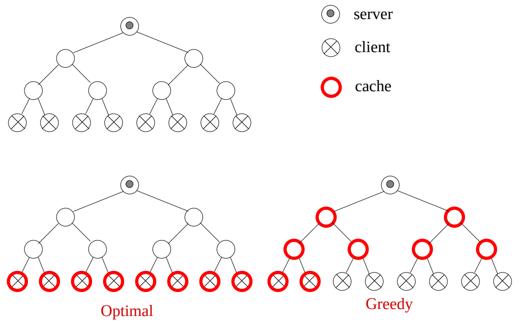

Fig. 4. A worst case example of Greedy vs. Optimal.

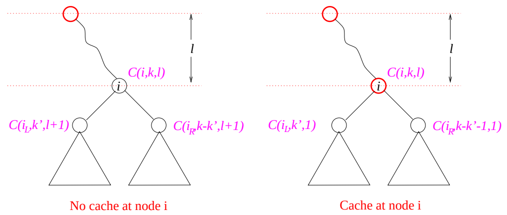

Fig. 5. A depiction of the dynamic programming optimization for $C(i,\ k,\ l)$ in a tree.

A. A Simple Greedy Algorithm

The intuitive greedy algorithm places caches on the tree iteratively in a greedy fashion, without replacing already assigned caches. That is, it checks each node of the tree to determine where to place the first cache, and chooses the node that minimizes the cost. It assigns the first cache to this node, updates the flows on the network due to this cache, and looks for an appropriate location for the next cache. Recall that we model the effect of a cache by the hit ratio p alone. That is, $(1-p)$ of the flow into a cache is propagated up the tree to the server. The complexity of the greedy algorithm is $O(nk)$.

B. Motivating the Optimal Algorithm

As we showed in Section III-A for a line graph, algorithm Greedy is suboptimal, but the difference is not significant. In theory, the approximation ratio of Greedy (i.e., $\lim\sup\{\mathrm{cost}(\mathrm{Greedy})/\mathrm{cost}(\mathrm{Optimal})\}$) is unbounded, where a bad example is a full homogeneous binary tree with $n = 2^i$ leaves and n caches. Clearly, the optimal solution will put a cache in each leaf resulting in 0 cost, while the greedy algorithm will occupy the nodes from the root downward, ending with a cost of $n-1$. It is true, however, that Greedy always needs at most twice as many caches to get the same cost as the optimal algorithm (see Figure 4).

C. The Optimal Dynamic-Programming Algorithm

Given a tree of n nodes, a set of (at most n) flows representing demands satisfied by a single server located at the root of the tree, and the number of caches, k, we need to compute the optimal locations for the caches and the total cost. We use a bottom up dynamic programming approach in the spirit of [32].

First, the general tree is converted into a binary tree by introducing at most n dummy nodes. We then sort all the nodes in reverse breadth first order, i.e., all descendants of a node are numbered before the node itself. For each node i having children i_L and i_R, for each $\tilde{k}$, $0 \le \tilde{k} \le k$, where k is the maximum number of caches to place, and for each l, $0 \le l \le h$, where h is the height of the tree, we compute the quantity $C(i, \tilde{k}, l)$. This quantity $C(i, \tilde{k}, l)$ is the cost of the subtree rooted at i with $\tilde{k}$ optimally located caches, where the next cache up the tree is at distance l from i. With each such optimal cost we associate a flow, $F(i, \tilde{k}, l)$, which is the sum of the demands in the subtree rooted at i that do not pass through a cache in the optimal solution of $C(i, \tilde{k}, l)$. It is not too difficult to verify that if no cache is to be put at node i, then the optimal solution for $C(i, \tilde{k}, l)$ is the one where

$$\min_{0 \le k' \le \tilde{k}} \quad (C(i_L, k', l+1) + C(i_R, \tilde{k} - k', l+1) + (l+1)(F(i_L, k', l+1) + F(i_R, \tilde{k} - k', l+1)) + l \cdot f_{s,i})$$

is achieved (see Figure 5). If we do put a cache at node i, the optimal solution is the one where

$$\min_{0 \le k' \le (\tilde{k}-1)} \quad (C(i_L, k', 1) + C(i_R, \tilde{k} - 1 - k', 1) + F(i_L, k', l+1) + F(i_R, \tilde{k} - k', l+1))$$

is achieved. While running the dynamic program we should also compute the appropriate $F(i, \tilde{k}, l)$, and keep track of the location of the caches in these solutions. The amount of data we have to keep is $O(nhk)$. At each node, for each $0 \le k' \le k$, and $0 \le l \le h$, we have to check all possible partitions of k' to the left and right subtrees. Therefore the overall time complexity is bounded by $O(nhk^2)$. However, using a clever analysis from [32] we can reduce the bound to $O(nhk)$. This is based on the observation that one cannot put in a subtree more caches than the number of nodes in it. Thus, for small subtrees (that have less than k nodes) we have to work less. Combining this with a counting argument that shows that the number of "big" (i.e., both children of the node have more than $k/2$ nodes in their subtrees) is $O(n/k)$, one can show that the actual complexity of the algorithm is $O(nhk)$ (see Lemmas 1 and 2 in [32]). This is much better than the $O(n^3k^2)$ complexity of a different dynamic programming algorithm for this problem proposed in [25].

This dynamic programming algorithm has been implemented, and it can solve the cache location problem on a tree consisting of several tens of thousands of nodes, with a depth of sixteen, and $k = 30$ caches in a few minutes on a Sun Ultra-1 machine. Our algorithm can also be used in more general cases, as described in the next section.

The same basic dynamic programming technique can, in fact, be used to handle the generalization of our model where we replace the cost of a hop from unity to any distance metric. This change does not affect the computational and space complexities of the algorithm.

V. Experiments and Results

In this section we describe our data collection method and the results from our experiments. Recall that reducing flow and lowering the average delay are equivalent in our model and we use these terms synonymously.

A. Data Collection and Methodology

We collected data from two web servers: a medium size site, `www.bell-labs.com` that receives about 200K–300K cachable (i.e., non-cgi) requests a week, and a smaller site, `www.multimedia.bell-labs.com` that receives up to 15000 cachable requests a week. We denote the `www.bell-labs.com` site by `BL`, and the `www.multimedia.bell-labs.com` site by `MM` for convenience. We considered two weeks of server logs from server `BL` (from late 1997 and early 1998) corresponding to "non-holiday" periods. Over these two weeks, an average of 14000 unique hosts per week accessed the server and 1Gbytes of cachable data per week were retrieved. We similarly chose seven weeks of server logs from server `MM`. Over these seven weeks, an average of 400 unique hosts per week accessed the server and 180 Mbytes per week were retrieved. The log files provided us with the server to client flows required by our model. 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. 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 specifying for each router, the lists of its multiple interfaces. The more difficult problems occur due to destinations that were alternating between two (or more) routes. This phenomena was observed mostly in the traceroutes from server `BL`, since it was bi-homed through two ISPs (BBNPLANET[2] and ALTERNET) during the time of the experiments; the same phenomenon was observed, to a lesser degree, in routes passing through the MCI backbone. When multiple paths to a node are identified we left in the graph only the path with the maximum aggregated flow, and prune the rest. The trees that we obtained had about 32,000 nodes for server `BL`, and about 12,500 nodes for server `MM`. This technique of creating the topology tree from a set of traceroutes has been used before, e.g., in [12].

For each data set, we computed the 100, 300, and 500 most popular pages at the server. Following our model, we assumed that all caches host these popular pages. The cachability of each flow is, thus, defined as the portion of the popular pages in the flow.

In Section V-B, we present the results of our experiments using this data. Intuitively, nodes contributing a small flow have minimal chance of impacting the solution, but add to the running time. We therefore studied simple heuristics for speeding up our algorithm by pruning the tree and eliminating nodes with little contributing flow. The exact method for pruning was to discard nodes that contributed less than $x\%$ of the total flow into the server. We observed that for sufficiently small values of x, like $x \leq 0.1\%$, the solution computed by our dynamic programming algorithm did not change, but the number of nodes to process decreased. Pruning also helps in visualizing the important parts of the tree.

B. Results

The results can be viewed under several categories. We first demonstrate the amount of traffic reduction that can be obtained by using TERCs. We then show that choosing the "right" location is non-trivial, and our approach has advantage over the common use of caches at the edges of the network. We compare the optimal algorithm with the *greedy* cache location algorithm. We also show that the cache location solution is stable; i.e., an offline calculation of the cache location based on past data is meaningful.

B.1 Traffic Reduction

To demonstrate the amount of traffic that can be saved using TERCs, we computed the total cost after putting TERCs in the optimal location with respect to the cost without caches. Recall that our cost is computed in terms of the bytes of data times the number of hops it travels. This was done for several cache sizes and is presented in Figure 6 for the week of December 1997. For this week, caching the top 100, 300 and 500 pages requires caches of size 0.6, 3.5, and 7.9Mbytes, respectively. For the week of Jan. 1998 the respective cache sizes are 1.2, 14.4, and

[2]BBNPLANET is now part of GTE.

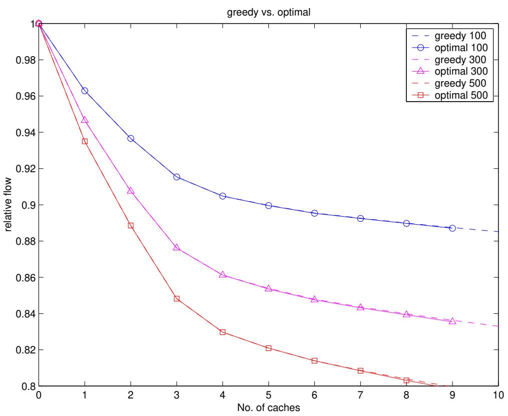

Fig. 6. The relative amount of traffic remains when using TERCs for server BL, when the top 100, 300, and 500 pages are cached (week of Dec. 1997).

22.5Mbytes (see explanation below). It can be seen, for example, that in caching the top 500 pages, putting just three caches (in the appropriate locations) reduces the overall traffic by more than 15% for the Dec. 1997 week. Similar savings can be achieved by using 6 caches that hold the top 300 pages. Better improvements were observed for the Jan. 1998 week. More significant reduction in traffic is achieved when we consider only the traffic of a single ISP as demonstrated in Figure 9. One can also observe that the greedy algorithm works well in both cases (Figures 6 and 9) within 3% of the optimal.

The reason for the large difference between the two weeks arises from a sub-tree in the server that contains a collection of large files; each file contains slides to accompany a chapter of an operating system book. In the second week, the demand for these slides rose ten-fold and advanced 20–30 very big files, each 0.4-1Mbytes, in the popularity chart from below the top 500 for the first week to places in the range of 123–360. This also influences the cachability figures: for the week of December 1997 the percentage of the flow (bytes$\times$hops) which is stored by caches with top 100, 300, and 500 pages is 21%, 31%, and 38%, respectively, while for the January 1998 week these numbers are 17%, 46%, and 55%, respectively. The cachability increases when the caches include over 300 pages but so does the required cache size.

B.2 Comparison with Common Practice

Having realized the benefit of putting TERCs in the network, we would like to demonstrate the importance of where the caches are located. A commonly used solution is to put caches at the edges of the network. Putting caches at all the boundary nodes of a network is an example of this solution and would presumably reduce the provider network traffic significantly. However, there are many such connection points, requiring a large number of caches to be placed.

An alternative approach is for the network provider to use the algorithms presented in this paper and determine a small number of nodes at which to put TERCs. We show

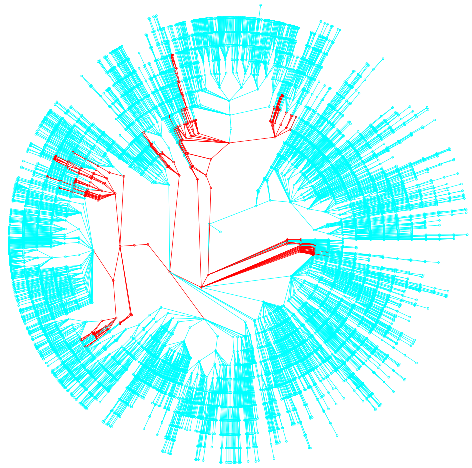

Fig. 7. The outgoing traffic from server BL with the portion of the BBN network emphasized.

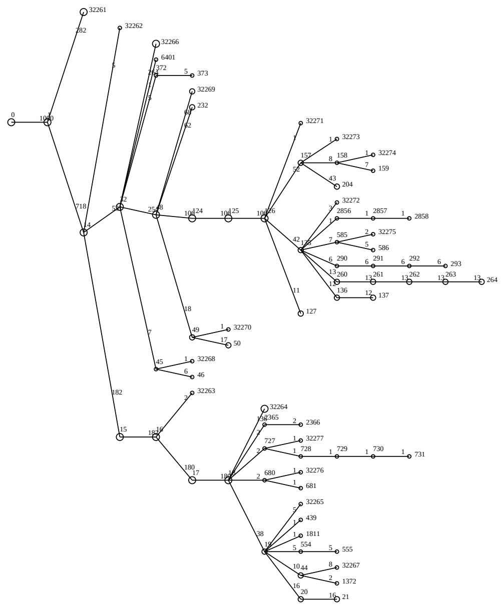

Fig. 8. The portion of the BBN network as seen by traffic from server BL. Only edges that carry at least 0.1% of the flow at the root are shown.

that such an approach can save almost the same amount of traffic using significantly fewer caches. For this experiment, we considered only the traffic inside the network of one of the ISPs (BBNPLANET) for server BL. Figure 7 shows the entire outgoing tree from server BL up to depth of 16, where the BBNPLANET network is emphasized. Out of the more than 11,000 nodes in the figure, 415 nodes belong to BBNPLANET. Figure 8 shows part of the outgoing traffic tree as viewed by server BL. In all tree figures in this paper (excluding Figure 7), the number next to a node is its unique id, and the number near an edge is the normalized traffic on this edge. The radius of the node is proportional to the traffic through it. The server is always located at node 0. For clarity, we only present the part of the tree which is

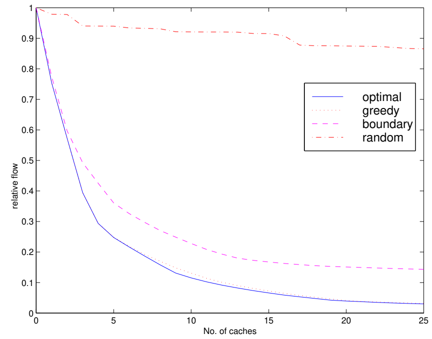

Fig. 9. A comparison of several placement strategies for a single ISP network (BBN in this case).

the most relevant.

There are about 360 relevant points at which the network is connected to different parts of the Internet, so putting caches on all these edges would enable us to reduce the cachable traffic in the BBN network practically to 0. We compared four cache placement strategies (see Figure 9): Optimal, greedy, optimal on the boundary only, and random. The optimal and the greedy algorithm are the ones discussed in sections IV-C and IV-A, respectively, applied only to the BBN portion of the network. For the special case of boundary placement, we observe that the greedy strategy is optimal. This is true since here no two caches can be placed on the same path from the server. The random strategy simply selects locations uniformly at random. We average five random selections for each point.

Figure 9 shows the relative cachable flow in the BBN network after placing a number (between 0 and 25) of caches according to one of the four strategies discussed above. For placing four caches and more, the boundary strategy is trailing the optimal and greedy strategies by over 10% of the cachable flow, which translates to over 4% of the overall traffic. From a different angle, to get the traffic reduction achieved with 7 caches placed in optimal locations we need to place 15 caches on the boundary. Note that, as expected, random placement is extremely inefficient since with high probability caches are placed in low traffic regions.

B.3 Comparing the Algorithms' Performance

We now compare the greedy algorithm presented in Section IV-A and the optimal cache placement algorithm presented in Section IV-C. We present a few examples of the locations found by the optimal and the greedy algorithms, and measure the actual benefits of using them, taking into account the traffic stability. In Tables II and III, we present the optimal cache locations and the cache locations obtained by the greedy algorithm, along with the normalized cost of the resulting configuration for server BL using the first week of data.

In Table II we can see that until the sixth cache (fourth in Table III) both algorithms behave the same. If we look at the resulting costs, however, it turns out that the difference is only 1%; not as dramatic as one might expect. Figure 6 plots the cost of both algorithms as a function of

#	*Greedy*		*Optimal*	
	cost	node	cost	node location
0	1	-	1	-
1	0.901	32	0.901	32
2	0.830	18	0.830	18 32
3	0.767	4	0.767	18 32 4
4	0.740	126	0.740	18 126 32 4
5	0.727	48	0.727	18 126 48 32 4
6	0.716	74	0.716	74 18 126 48 32 4
7	0.708	23	0.708	74 18 33 126 48 103 4
8	0.701	204	0.700	74 18 33 126 48 103 23 4
9	0.695	52	0.693	74 18 33 204 126 48 103 23 4

TABLE II

Comparing the greedy and the optimal. Data from the server BL for the 1997 week, caching the top 500 pages.

#	*Greedy*		*Optimal*	
	cost	node	cost	node location
0	1	-	1	-
1	0.969	32	0.969	32
2	0.948	18	0.948	18 32
3	0.930	4	0.930	18 32 4
4	0.922	126	0.922	18 126 32 4
5	0.917	48	0.917	18 157 48 32 4
6	0.914	74	0.914	74 18 157 48 32 4
7	0.912	204	0.911	74 18 33 157 48 103 4
8	0.909	23	0.909	74 18 33 157 48 103 23 4
9	0.907	52	0.907	74 18 33 204 126 48 103 23 4

TABLE III

Comparing the greedy and the optimal. Data from the server BL for the 1997 week, caching the top 100 pages.

the number of TERCs, Figure 9 plots larger differences for the BBN portion of the network. In Table II the first time the two algorithms behave differently is when the seventh cache is placed. There, the optimal algorithm removes the cache from node 32, and puts two caches at nodes 33 and 103 which are the children of node 32. This is a common transformation that the optimal algorithm performs. Table III exhibits another typical behavior (though observed in fewer cases than the first) in the transformation from four to five caches. A cache in node 126 is replaced by a cache in node 48 which is its great grandparent and in node 157 which is its child (see Figure 8). Later the transformation from eight to nine caches replaces a cache in node 157 with a cache in its parent node 126 and its child 204. The reason for these transformations is that the route 32-48-124-125-126-157-204 has a large portion of the traffic, with some heavy splits along it. Node 48 is a BBN backbone router that receives a third of the traffic from BL, node 204 is the BBN interface to MAE-east that receives about 5% of the traffic. The transformation from six to seven caches involves (as in Table II) the replacement of a cache in node 32 with two caches at its children nodes, 33 and 103.

Overall the difference between the two algorithms is very small and typically in the range of 0.75-3%. We checked over 30 cases by considering a combination of daily and weekly data for both servers and detected the same phenomenon.

As discussed in Section IV-B, in theory, the approximation ratio of greedy is unbounded. However, in practice, based on our experiments, the greedy algorithm behaves quite well. This is probably due to the fact that the scenarios in which the greedy algorithm performs badly are somewhat pathological and do not appear in practice. Specifically, the example where Greedy performs miserably is for a balanced tree, while, in practice, we noticed that the flow trees tend to be highly imbalanced.

B.4 Stability

In this section, we show that over time, the flow pattern from the source to the clients is stable, at least in the part of the tree that has most of the flow and is therefore relevant to caching.

As we said in the introduction, although the client population changes significantly from day to day and from week to week, the flow in the outgoing tree from the server to the clients remains pretty much stable in the branches that carry most of the traffic. This means that the part of the tree relevant for caching does not change by much as time progresses.

Figures 10 and 11 show the trees obtained from the logs for server `BL` for two days in January 1998, the 13th and the 14th. The two trees are visibly similar (actually, mirror images, since the two gateway nodes, 2 and 14, are reversed in the two plots).

This similarity is surprising given the fact that the client population varies significantly from day to day. We observe that there were only between 2.7 – 7.5% “repeat clients”; i.e., there was a very small intersection between the client populations of any two days for the two week data from server `BL`. In particular, for the two days shown in Figures 10 and 11, there were only 7.48% repeat clients. A similar effect was seen for server `MM`. Tables VI and VII in the appendix show details of this phenomenon.

To measure stability of the cache placement solution, we do the following. We calculate the optimal cache locations using the entire two week data for server `BL`, assuming for convenience a constant hit ratio for all flows. For each day, we compute the cost for that day using these cache locations, and compare it with the cost of the optimal location for that day. Table IV shows this ratio for placement of 6 and 12 caches. The difference is between 1% and 55%. However, most of the big differences occur in the weekend days when traffic volumes are smaller and where traffic patterns are somewhat different. Table V compares the performance of the cache location calculated based on the seven week data from server `MM` with respect to the optimal location per week. For 6 caches, in five of the weeks the difference is less than 10%. For 12 caches the differences are between 13% and 42%.

day	date	6 caches	12 caches
SUN	Jan 11	1.33221	1.40064
MON	Jan 12	1.06318	1.24182
TUE	Jan 13	1.0331	1.11151
WED	Jan 14	1.18086	1.12338
THU	Jan 15	1.02191	1.05215
FRI	Jan 16	1.01368	1.15499
SAT	Jan 17	1.36181	1.55984
SUN	Nov 30	1.09162	1.30728
MON	Dec 01	1.28618	1.43075
TUE	Dec 02	1.09031	1.202
WED	Dec 03	1.08761	1.19723
THU	Dec 04	1.10327	1.18992
FRI	Dec 05	1.04525	1.10881
SAT	Dec 06	1.14363	1.32723

TABLE IV
THE RATIO BETWEEN THE PLACEMENT SOLUTION THAT IS BASED ON THE TWO WEEK DATA AND THE OPTIMAL DAILY SOLUTION FOR SERVER `BL`.

week	6 caches	12 caches
8	1.03795	1.14502
9	1.06289	1.16789
10	1.06615	1.26899
11	1.10764	1.22822
12	1.18493	1.29394
49	1.04459	1.12884
50	1.34613	1.41774

TABLE V
THE RATIO BETWEEN THE PLACEMENT SOLUTION THAT IS BASED ON THE ACCUMULATED 7 WEEK DATA AND THE OPTIMAL WEEKLY SOLUTION FOR SERVER `MM`.

It is instructive to view these results in the context of the traffic improvement numbers from Figure 6. By placing caches based on historical data, one can reduce the network traffic by 15% using only 6 TERCs, assuming a 40% hit ratio for all flows.

B.4.a Routing Stability. 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 [30] suggest that most routes remain stable over the small lifetime of an HTTP connections. Labovitz et al. [24] 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. To do so, we performed three consecutive `traceroute` from Bell-Labs to 13533 destinations. On the average the time between

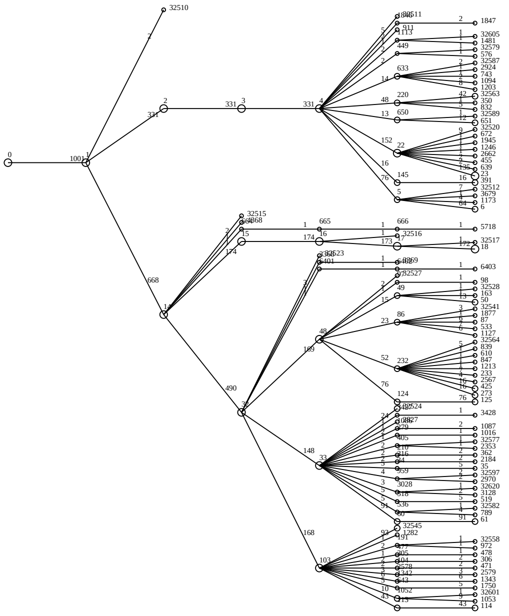

Fig. 10. The routing tree for January 13th, 1998 for server BL.

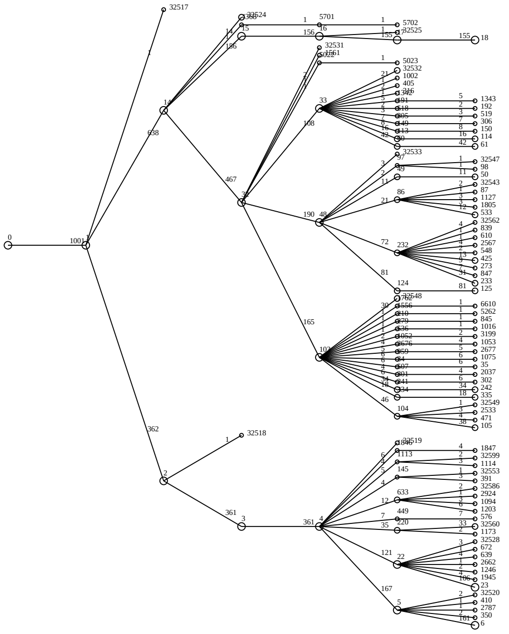

Fig. 11. The routing tree for January 14th, 1998 for server BL.

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 our equivalence include only interface pairs we could positively identify as equivalent and we expect that we missed many more.

Due to the packet re-orderings caused by route changes, many ISPs implement route caching for TCP connections (e.g., using NetFlow Switching in the Cisco 7200 & 7500

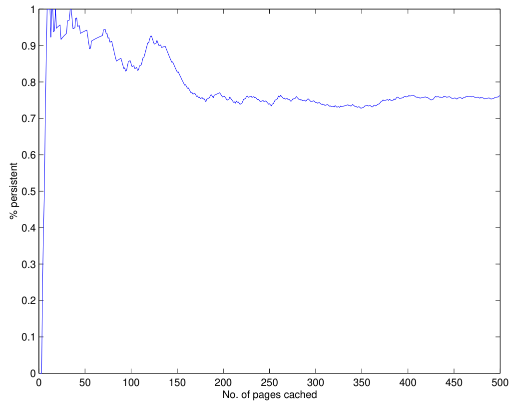

Fig. 12. Popularity stability for server BL.

series routers), i.e., even when routing entries change, existing "connections" use the old routing path. The route caching did not effect 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 [30].

B.4.b Popularity Stability. Figure 12 shows stability of the most popular pages in site BL. For every number x, $1 \leq x \leq 500$, it plots the portion of the pages that are on the top x popular page list in both weeks (Nov. 1997 and Jan. 1998). For example, 7 out of the top 8 list of one week are also on the top 8 list of the other week, and therefore, the persistence plotted is 7/8. The figure shows that for the first 50 pages, the popularity list of both weeks always share more than 90% of the pages, and in most cases more than 95%. In general the popularity lists are, at least, 75% identical. Note that the two weeks compared are not consecutive (they are 5 weeks apart).

VI. Discussion

Web caching is a very cost-effective method to deal with network overload. Solutions like TERCs have the advantage that they do not require changes (like a proxy assignment) by a user, and are easy to install and manage locally within a provider network. Therefore, they are attractive building blocks to any future caching strategy. Once installed, the benefit from a device will determine the further use of this solution. We identify that the location at which the caches are placed play a prime role in the resulting traffic and load reduction. Thus, addressing the location problem of caches is an important part in the campaign for web caching.

In this paper we laid the groundwork for research in this area by defining the model, and devising a computationally-efficient algorithmic solution for the important practical case of one server. We have experimentally demonstrated the advantage of using TERC like caching devices in today's WWW, and the importance of the cache location problem.

Clearly, there are still many open questions. The most important problem is how to optimally locate TERCs in the case when there are many servers. That is, where should the caches be put inside a provider network that supports many clients and servers. Our results suggest that the following iterative heuristic *ℓ-Greedy*, which is an adaptation of the greedy technique, should work well in practice. For $\ell = 0$, algorithm *ℓ-Greedy* is the standard Greedy algorithm described in Section IV-A. For general ℓ, algorithm *ℓ-Greedy* greedily replaces some ℓ already assigned caches with $\ell + 1$ caches. That is, caches that are already assigned can be moved around in a limited way to improve the objective cost function. The intuition for this algorithm stems from our observation that, in practice, the optimal solution for our single server experiments was always obtained by *2-Greedy*. For example, the optimal solutions in Tables II and III are obtained by *1-Greedy*. The main problem in evaluating any multi-server placement algorithm is that it is harder to obtain general network web traffic data.

Another important issue is our objective function: What do we want to optimize in a wide area network to get better performance? Our algorithm would work for any average benefit function that corresponds to a global criterion but will not work for worst case measures like improving the most loaded link, or the most loaded server. Other interesting directions for further research include the extension of the model to enable it to capture hierarchical caching structures and multicast traffic. Techniques like the ones used in active networks and the continued process of memory cost reduction may lead to a scenario in which caches can be dynamically moved in the network. This will require local distributed techniques to deal with the dynamic optimal cache location problem.

Appendix

I. Stability Statistics

wk	8	9	10	11	12	49	50
8		6.30	4.75	5.48	3.92	2.21	2.29
9	6.30		7.68	5.59	4.37	2.13	2.07
10	4.75	7.68		6.17	4.65	1.76	1.71
11	5.48	5.59	6.17		6.62	2.08	2.16
12	3.92	4.37	4.65	6.62		2.16	2.09
49	2.21	2.13	1.76	2.08	2.16		7.44
50	2.29	2.07	1.71	2.16	2.09	7.44	

TABLE VII
The percent of clients that appear in the logs of any two weeks for server MM.

References

[1] Proceedings of the usenix symposium on internet technologies and systems, Dec. 1997.
[2] T. Berners-Lee, R. Fielding, and H. Frystyk. Hypertext Transfer Protocol – HTTP/1.0, May 1996. RFC 1945.
[3] A. Bestavros. Demand-based document dissemination to reduce traffic and balance load in distributed information systems. In *Proceedings of SPDP'95: The 7^{th} IEEE Symposium on Parallel and Distributed Processing*, San Anotonio, Texas, Oct. 1995.
[4] A. Bestavros. Speculative data dissemination and service to reduce server load, network traffic and service time for distributed information systems. In *Proceedings of ICDE'96: The 1996 International Conference on Data Engineering*, New Orleans, Louisiana, Mar. 1996.
[5] L. Breaslau, P. Cao, L. Pan, G. Phillips, and S. Shenker. Web caching and Zipf-like distributions: Evidence and implications. In *IEEE INFOCOM'99*, pages 126 – 134, Mar. 1999.
[6] Proxy cache comparison. `http://www.web-caching.com/proxy-comparison.html`.
[7] Internet caching resource center. `http://www.caching.com/`.
[8] A. Chankhunthod, P. B. Danzig, C. Neerdaels, M. F. Schwartz, and K. J. Worrell. Hierarchical internet object cache. In *Usenix Technical Conference*, 1996.
[9] M. Chatel. Classical versus Transparent IP Proxies, Mar. 1996. RFC 1919.
[10] 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.
[11] G. Călinescu, C. G. Fernandes, and B. Reed. Multicuts in unweighted graphs with bounded degree and bounded tree-width. In R. E. Bixby, E. A. Boyd, and R. Z. Rmos-Mercado, editors, *Integer Programming and Combinatorial Optimization*, pages 137 – 152, June 1998. LNCS 1412.
[12] C. Cunha. *Trace Analysis and Its Applications to Performance Enhancements of Distributed Information Systems.* PhD thesis, Boston University, 1997.
[13] 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.
[14] 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.
[15] 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.
[16] R. T. Fielding, J. Gettys, J. C. Mogul, H. F. Nielsen, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1, Jan. 1997. RFC 2068.
[17] 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.
[18] M. R. Garey and D. S. Johnson. *Computer & Intractability: A Guide to the Theory of NP-Completeness.* W H Freeman, Nov. 1979.
[19] M. W. Garrett and S.-Q. Li. A study of slot reuse in dual bus multiple access networks. *IEEE Journal on Selected Areas in Communications*, 9(2):248 – 256, Feb. 1991.
[20] 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.
[21] O. Kariv and S. L. Hakimi. An algorithmic approach to network location problems, part II: p-medians. *SIAM Journal on Applied Mathematics*, 37:539 – 560, 1979.
[22] P. Krishnan, D. Raz, and Y. Shavitt. Transparent en-route cache location in regular network topologies. In *DIMACS Workshop on Robust Communication Networks: Interconnection and Survivability*, Nov. 1998.
[23] 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.
[24] C. Labovitz, G. R. Malan, and F. Jahanian. Internet routing instability. In *ACM SIGCOMM'97*, Aug. 1997.
[25] B. Li, M. J. Golin, G. F. Italiano, X. Deng, and K. Sohraby. On the optimal placement of web proxies in the internet. In *IEEE INFOCOM'99*, pages 1282 – 1290, Mar. 1999.
[26] 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.
[27] I. Melve. Why internet service providers should integrate web caches into their networks. In *web caching in the internet conference*, sep/oct 1996.
[28] A distributed testbed for national information provisioning. `http://www.nlanr.net`.

day	0111	0112	0113	0114	0115	0116	0117	1130	1201	1202	1203	1204	1205	1206
0111		4.35	4.00	3.78	3.69	3.55	3.73	3.25	3.12	3.21	2.96	3.01	2.79	3.36
0112	4.35		6.93	6.06	5.66	5.34	3.58	2.77	4.40	3.85	3.86	3.87	4.02	3.33
0113	4.00	6.93		7.48	6.10	6.12	4.26	3.28	4.58	4.25	4.16	4.34	4.25	2.96
0114	3.78	6.06	7.48		7.33	6.48	4.07	3.03	4.21	4.23	4.28	4.34	4.25	3.15
0115	3.69	5.66	6.10	7.33		7.41	4.30	2.77	3.71	4.02	4.25	3.98	4.20	2.88
0116	3.55	5.34	6.12	6.48	7.41		5.38	3.13	4.21	4.56	4.12	4.10	4.36	3.25
0117	3.73	3.58	4.26	4.07	4.30	5.38		3.36	2.99	3.14	2.86	2.88	3.18	3.46
1130	3.25	2.77	3.28	3.03	2.77	3.13	3.36		4.32	4.08	4.15	3.42	3.49	4.23
1201	3.12	4.40	4.58	4.21	3.71	4.21	2.99	4.32		7.00	6.34	6.06	4.97	3.58
1202	3.21	3.85	4.25	4.23	4.02	4.56	3.14	4.08	7.00		6.88	5.89	5.35	3.94
1203	2.96	3.86	4.16	4.28	4.25	4.12	2.86	4.15	6.34	6.88		7.01	5.58	3.48
1204	3.01	3.87	4.34	4.34	3.98	4.10	2.88	3.42	6.06	5.89	7.01		7.15	3.95
1205	2.79	4.02	4.25	4.25	4.20	4.36	3.18	3.49	4.97	5.35	5.58	7.15		4.82
1206	3.36	3.33	2.96	3.15	2.88	3.25	3.46	4.23	3.58	3.94	3.48	3.95	4.82	

TABLE VI

The percent of clients that appear in the logs of any two days for server `BL`.

[29] NLANR. *NLANR Web Caching Workshop*. `http://www.nlanr.net/Cache/Workshop97/`.

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

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

[32] A. Tamir. An $O(pn^2)$ algorithm for the p-median and related problems on tree graphs. *Operations Research Letters*, 19:59 – 64, 1996.

[33] 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.

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

[35] L. Zhang, S. Michel, K. Nguyen, A. Rosenstein, S. Floyd, and V. Jacobson. Adaptive web caching: Towards a new global caching architecture. In *3rd International WWW Caching Workshop*, June 1998.

P. Krishnan P. Krishnan (who is called “Krishnan” or “PK”) is currently with ISPSoft, Inc. He received his Ph.D. in Computer Science from Brown University, and his B. Tech in Computer Science from the Indian Institute of Technology, Delhi. Prior to joining ISPSoft, he was with the Networking Research Center at Bell Labs, Lucent Technologies. His research interests include IP networking and management, the development and analysis of algorithms, prefetching and caching, and mobile computing.

Danny Raz Danny Raz (IEEE M'99) received his doctoral degree from the Weizmann Institute of Science, Israel, in 1995. From 1995 to 1997 he was a post-doctoral fellow at the International Computer Science Institute, (ICSI) Berkeley, CA, and a visiting lecturer at the University of California, Berkeley. Since October 1997 he is with the Networking Research center at Bell-Labs, Lucent technologies. His primary research interest is the theory and application of management related problems in IP networks.

Yuval Shavitt Yuval Shavitt (IEEE S'88–M'97) received the B. Sc. in Computer Engineering (cum laude), M. Sc. in Electrical Engineering and D. Sc. from the Technion — Israel Institute of Technology, Haifa, Israel in 1986, 1992, and 1996, respectively.
From 1986 to 1991, he served in the Israel Defense Forces first as a system engineer and the last two years as a software engineering team leader. After graduation he spent a year as a Postdoctoral Fellow at the Department of Computer Science at Johns Hopkins University, Baltimore, MD. Since 1997 he is a Member of Technical Stuff at Bell Labs, Lucent Technologies, Holmdel, NJ. His recent research focuses on active networks and their use in network management, QoS routing, and Internet mapping and characterization.