DIMACS Series in Discrete Mathematics
and Theoretical Computer Science

Transparent En-Route Cache Location for Regular Networks

P. Krishnan, Danny Raz, and Yuval Shavitt

Abstract. Caching improves network and system performance for WWW browsing. The way caches are currently deployed requires clients or caches to be aware of the location of nearby caches. This creates management and configuration problems, which 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 snaps 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.

In this work, we study the important problem of where to place network caches, and in particular, TERCs. This problem is intractable for general networks, and thus we concentrate on regular topologies namely lines and rings. We give both a general dynamic programming algorithm, and closed formulas for some special cases. Our results demonstrate both the significant saving achieved by using TERCs, and the importance of placing the caches in the optimal locations.

1. Introduction

Caching improves network and system performance for WWW browsing by saving network bandwidth, reducing delays to end clients, and alleviating server load [**DHS93, NLA**]. 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 [**CDN**$^+$**96, Squ**]. Significant research has gone into optimizing cache performance [**CDN**$^+$**96, WAS**$^+$**96, NLA**], co-operation among several caches [**CDN**$^+$**96, KS97, MLB95, GRC97**], and cache hierarchies [**Squ, CDN**$^+$**96, NL**]. Web servers are also replicated to achieve load-balancing.

An important method that is gaining momentum is caching inside the network [**ZFJ97, DHS93, HM97**]. Danzig et al. [**DHS93**] 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 [**ZFJ97**]. Heddaya and Mirdad [**HM97**] 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.

In this paper, we concentrate 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 snaps 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. Such a model requires little or no change to the existing infrastructure, except the introduction of caches in the network at appropriate locations. Existing or upcoming products [**Cis97**] can help with such a cache deployment. TERCs are easier to manage than replicated web servers since they are oblivious. 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.

An important issue in en-route caching is determining the appropriate locations for the caches. In this paper, we study the problem of optimally locating en-route 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. 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. Observing the importance of the cache's hit rate, and motivated by proxy caching studies [**NLA, Use97, WAS⁺96, DFKM97**] that report a fairly steady hit rate, we fix the cache hit rate for all TERCs in the network, and use this parameter to represent the performance of the caching algorithm.

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 [**GJ79**]. We thus solve here the en-route cache location problem for special types of graphs namely line graphs and rings. We observe that a small number of TERCs are sufficient to reduce the network traffic significantly. We compare our optimal algorithm against a greedy algorithm and report on their relative performance. Similar results were reported for general networks [**KRS98**].

In Section 2, we present our model in detail, and describe the computational complexity of the problem. We present our algorithms for a homogeneous line in Section 3, and for general line in Section 4. Section 5 discuss the cache placement in rings. We conclude our results in Section 6.

1.1. Related work. Transparent en-route caches were suggested for load balancing by Heddaya and Mirdad [**HM97**]. They point to the use of run-time code generation techniques to dynamically download high performance packet filters to the kernel [**EK96**]. Zhang et al. [**ZFJ97**] also advocated the use of en-route caches, but do not address the placement problem. Transparent caches could be implemented with packet-filter techniques used by firewalls or by new active network techniques [**TSS⁺97**]. Cisco's CacheEngine [**Cis97**] (currently designed for Intranets) is an example of an enabling product.

Zhang et al. [**ZFJ97**] have recently 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, and for server push models [**Bes95**].

Our dynamic programming solution is similar to the one used by Hassin and Tamir [**Tam96**] for solving the p-median problem, and by Narahari et al. [**NSS95**] for solving the erasure node placement in DQDB LANs. Our homogeneous solution are in the flavor of the one used by Garrett and Li [**GL91**] for erasure node placement in DQDB LANs.

2. 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 [**BLFF96, FGM⁺97**]. The

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

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 concern here is to find good locations for the caches. To achieve this, we abstract away the exact behavior of the caches, and replace it with a simple, yet meaningful, parameter, the *cache hit rate*. The cache hit rate, p, is the fraction of data that can be served from the cache's local memory. A higher hit rate implies a lower load on the network, and much work has been done to improve cache hit rates [**WAS+96, NLA, Use97**]. It has been shown [**Bes96**] that amongst all pages on a server only a small fraction are very popular, and our measurements [**KRS98**] 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 is not found in one of the caches, it will not be found in any other cache.

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 [**Mel96**], 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. Our objective is to minimize the total network flow; i.e., the sum of the web flows taken over all the links. This may eliminate the need to increase the capacity of network links. It is equivalent to minimizing the average delay for user traffic through our network and thus also optimize the performance gain to the entire user population.

2.1. 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 shortest distance between nodes v_i and v_j. 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. We denote by K the set of at most k nodes where the caches are to be put, and the hit rate by p. The *cost* cost $c_{s,c}$ of demand $f_{s,c}$ using a cache in location v_k is

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

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 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 [**NL**]. 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 2.1. **The general k-cache location problem.**

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

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

As pointed out in Section 1, 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 the ones 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. In other words,

	line	*Bounded degree tree*	*Tree*	*General graph*
One server	Polynomial	Polynomial	Polynomial	$\boldsymbol{NP}$-hard
m servers	Polynomial	$\boldsymbol{NP}$-hard	$\boldsymbol{NP}$-hard	$\boldsymbol{NP}$-hard

TABLE 1. The hardness of the k-cache and k-TERC location problems.

PROBLEM 2.2. **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 2.1), except that the minimization in the objective function is over the set $\{v_k \in (K \cup \{v_s\}) \cap \{path(v_c, v_s)\}$.

Note that for the TERC location problem (unlike for the general k-cache location problem), the optimal set K will not depend on p, since all we need for a solution to be optimal is that the (weighted) average distance from a client to the nearest TERC on the route to the server is minimal. Therefore, without loss of generality, for the TERC location problem, we can assume that $p = 1$. Notice that the $p = 1$ means that all the requests are delivered by the cache, therefore there is no meaning to discuss dependency among the caches in this case. This means that the optimal locations for the independence case with $p = 1$, is exactly the optimal location for the fully dependence caches. Another interesting observation is that the k-TERC location problem is a special case of the general k-cache location problem since by choosing $p = \frac{1}{n}$ the caches are forced to be on the path from the client to the server.

2.2. 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 1 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 when $p = 1$ is NP-hard. In fact, it is not too difficult to show that this case is equivalent to the well known p-median problem [**KH79, GJ79**]. 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 correspond to caches that are related to a specific link. The proof is obtained via a reduction from the multicuts problem [**CF98**] 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. The details of all these reductions are out of the scope of this paper.

3. Homogeneous Line

3.1. 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

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

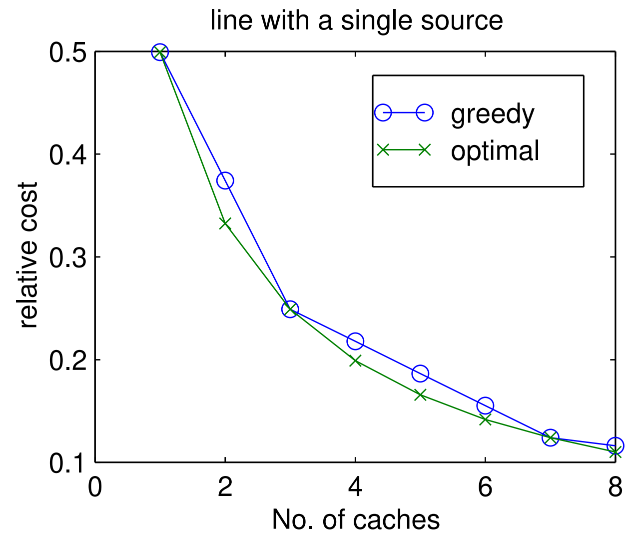

FIGURE 1. A line network with a single source and homogeneous client population. We compare the greedy algorithm with the optimal placement.

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 [**KRS98**].

3.2. 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 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 start by analyzing the location of single interface cache and then analyze the multiple interface cache. We also analyze the independent caches here, as it is more general, and use the results for $p = 1$, for the fully dependent case.

3.2.1. *Single Interface Cache.* Suppose we have n nodes numbered $1, 2, \dots, n$. Let the number of available caches, k, be one. Since the traffic is symmetrical for the two directions we need to improve only an arbitrary selected side. Thus in the rest of this case analysis we assume a unidirectional flow in the direction of node 1.

The gain in traffic that results from placing a cache at the incoming interface to node t is given by

$$\mathcal{G} = \sum_{i=1}^{t} \sum_{j=t+1}^{n} j - t = \frac{1}{2}\left(t^3 - (2n+1)t^2 + n(n+1)t\right) \tag{6}$$

To find the optimal location, we can take the derivative $\mathcal{G}$

$$\frac{d\mathcal{G}}{dt} = 3t^2 + 2(2n+1)t + n(n+1) \tag{7}$$

the derivative zeroes for

$$t^{opt} = \frac{1}{3}\left((2n+1) - \sqrt{(n+1)^2 - n}\right) \tag{8}$$

which implies that

$$\frac{n}{3} < t^{opt} < \frac{n}{3} + \frac{1}{3} \tag{9}$$

Since we can place caches only at node location the optimal location is at

$$t^{opt} = \frac{n}{3} \tag{10}$$

For $k > 1$ this method becomes more and more cumbersome. To overcome this difficulty we assume that the line in continuous [**GL91**]. In each point along the line from 0 to 1 there exists a node that is a source and a destination. For $k = 1$, the cost of the flow in the line when a single cache is put at point t is given by

$$\mathcal{F} = \int_0^t \int_r^t (s-r)\,ds\,dr + \int_0^t \int_t^1 (pt + (1-p)s - r)\,ds\,dr + \int_t^1 \int_r^1 (s-r)\,ds\,dr \tag{11}$$

where p is the average hit rate for a data item in the cache. The optimum is achieved at $t^{opt} = 1/3$ regardless of the value of p which corresponds to the discrete result in (10). Figure 2 depicts the flow in the line with optimal cache location as a function of the hit ratio, p. For an achievable hit ration of 40%, we see that 20% of the total flow can be saved.

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:

$$\begin{aligned} \mathcal{F} &= \int_0^{t_1}\int_r^{t_1} (s-r)\,ds\,dr + \int_{t_1}^{t_2}\int_r^{t_2} (s-r)\,ds\,dr + \int_{t_2}^{1}\int_r^{1} (s-r)\,ds\,dr + \int_0^{t_1}\int_{t_1}^{t_2} (pt_1 + (1-p)s - r)\,ds\,dr \\ &\quad + \int_{t_1}^{t_2}\int_{t_2}^{1} (pt_2 + (1-p)s - r)\,ds\,dr + \int_0^{t_1}\int_{t_2}^{1} (pt_1 + (1-p)pt_2 + (1-p)^2 s - r)\,ds\,dr \\ &= \frac{1}{6}\left(1 + 3p^2 t_1(1-t_2)^2 - 3p(t_1(1-t_1)^2 + t_2(1-t_2)^2)\right) \end{aligned} \tag{12}$$

Taking the derivative of $\mathcal{F}$ w.r.t. t_2 and setting to zero, gives an optimum at

$$t_2^{opt} = pt_1 + \frac{1}{3}(1 - pt_1) \tag{13}$$

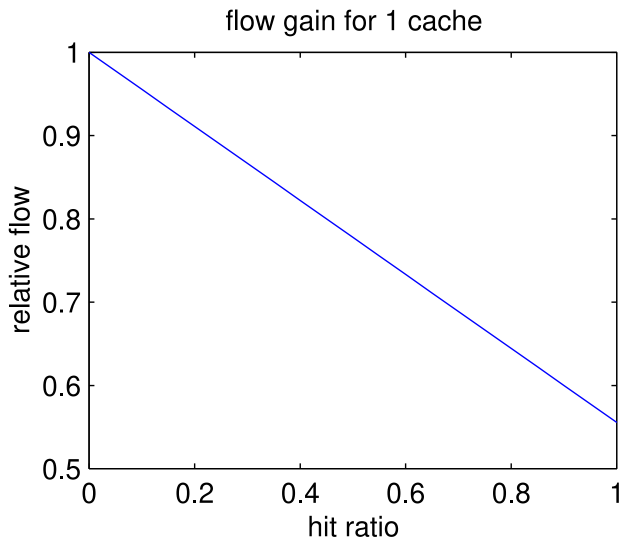

FIGURE 2. The relative flow in a line network with one cache placed at optimal location as a function of p.

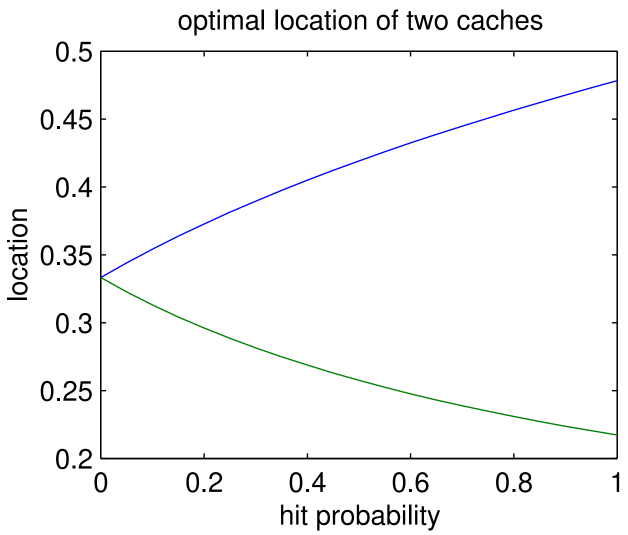

FIGURE 3. The optimal location of two caches in a homogeneous line as a function of p.

Substituting the term for t_2^{opt} from eq. 13 in the term for the flow (eq. 12), deriving by t_1 and comparing to zero yields the optimum for t_1:

$$t_1^{opt} = \frac{18 - 4p^2 - 3\sqrt{9 + 12p - 16p^2 + 4p^3}}{27 - 4p^3} \tag{14}$$

The optimal location for the two caches as a function of the hit ratio is depicted in figure 3. When $p = 1$ the caches are placed at locations 5/23 and 11/23. Figure 4 depicts the flow in the line with optimal cache location as a function of the hit ratio, p. As can be seen for an achievable hit ratio of 40%, 30% of the total flow can be saved.

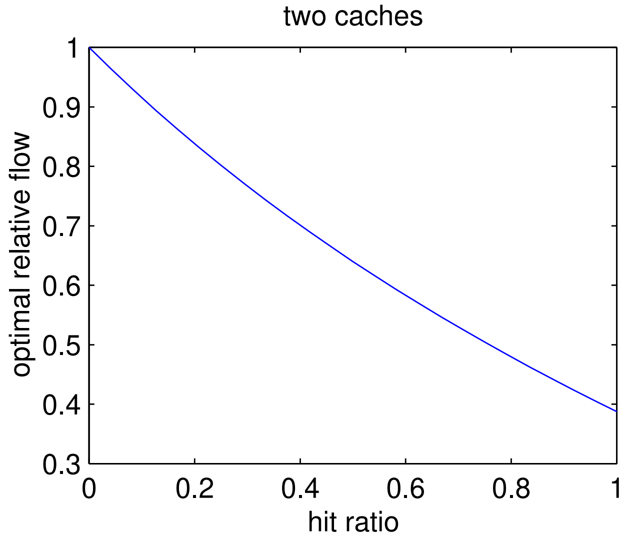

FIGURE 4. The relative flow in a line network with two caches as a function of p.

Equation 12 assumes that the probability to find an item in a cache is independent. However some studies [**DHS93**] suggest that this is not the case. The claim is that a meaningful volume of WWW traffic is cachable, but that the same popular pages are demanded everywhere and thus are expected to be found in all caches while other pages are not to be found anywhere. To reflect this we change eq. 12 to be

$$\begin{aligned}\mathcal{F} &= \int_0^{t_1}\int_r^{t_1}(s-r)\,ds\,dr+\int_{t_1}^{t_2}\int_r^{t_2}(s-r)\,ds\,dr+\int_{t_2}^{1}\int_r^{1}(s-r)\,ds\,dr+\int_0^{t_1}\int_{t_1}^{t_2}(pt_1+(1-p)s-r)\,ds\,dr\\ &+\int_{t_1}^{t_2}\int_{t_2}^{1}(pt_2+(1-p)s-r)\,ds\,dr+\int_0^{t_1}\int_{t_2}^{1}(pt_1+(1-p)s-r)\,ds\,dr \qquad (15)\\ &= \frac{1}{6}\left(1-3p(t_1^3-2t_1^2+t_1t_2(2-t_2)+t_2(1-t_2)^2)\right)\end{aligned}$$

Note the the change is only in the last element where flows that traverse both caches are considered. Solving as before we get that the optimal locations for the caches are at $5/23$ and $11/23$ regardless of p. Note that t_2 location is at $t_1+(1-t_1)/3$, i.e., at one third of the way between t_1 and 1.

3.2.2. *Multiple Interface Cache.* Due to symmetry, when only a single cache is available its position is obviously in the line center. As with the single interface cache, when $k>1$ we solve for the continuous line rather than for 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:

$$\begin{aligned}\mathcal{F} &= \int_0^{t_1}\int_r^{t_1}2(s-r)\,ds\,dr+\int_{t_1}^{t_2}\int_r^{t_2}2(s-r)\,ds\,dr+\int_{t_2}^{1}\int_r^{1}2(s-r)\,ds\,dr\\ &+\int_0^{t_1}\int_{t_1}^{t_2}(p+2(1-p))(s-r)\,ds\,dr+\int_{t_1}^{t_2}\int_{t_2}^{1}(p+2(1-p))(s-r)\,ds\,dr\\ &+\int_0^{t_1}\int_{t_2}^{1}pt_1+(1-p)pt_2+(1-p)^2s-r+s-(pt_2+p(1-p)t_1+(1-p)^2r)\,ds\,dr \qquad (16)\\ &= \frac{1}{3}+\frac{1}{2}\left(p^2\,t_1\,(1+t_1-t_2)\,(1-t_2)-p\,((1-t_1)\,t_1+(1-t_2)\,t_2)\right)\end{aligned}$$

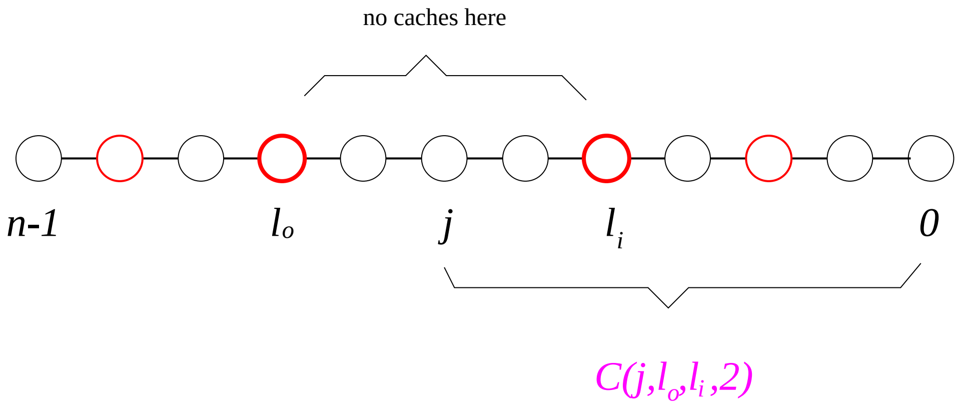

FIGURE 5. The definition of $C(j,\ l_o,\ l_i,\ k')$.

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

$$t_2^{opt} = \frac{1 + pt_1(2 + t_1)}{2(1 + pt_1)} \tag{17}$$

substituting t_2^{opt} in $\mathcal{F}$ and deriving by t_1 gives the optimum

$$t_1^{opt} = \frac{\sqrt{1 + 3p} - 1}{3p} \tag{18}$$

when the hit ratio is equal to 1 the caches optimal position is at 1/3 and 2/3.

As with unidirectional caches, when the hit probability is assumed to be fully correlated the cache position is independent of p, and in this case it is in 1/3 and 2/3.

4. The General Line

In this section we give an optimal solution to the general cache location problem 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 2. Under this assumption the optimal location does not depends on the hit ratio, thus, for convenience, we assume a hit ratio of one.

4.1. Multiple interface cache. 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 5 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 $FR(i) = FC(i, n-1, 0)$.

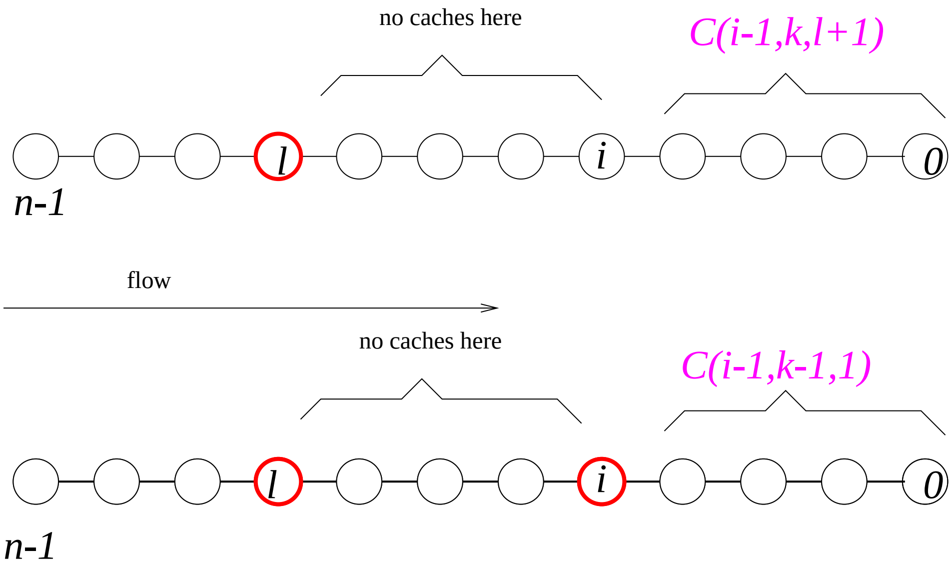

FIGURE 6. A depiction of the dynamic programming optimization for $C(i,\ k,\ l)$ in the single interface cache case.

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 4.1.

$$\begin{aligned} C(j,\ l_o,\ l_i,\ k') \quad &= \min\{ \quad C(j-1,\ j,\ l_i,\ k'-1) + FC(j,\ j,\ l_i), \\ & \qquad\qquad C(j-1,\ l_o,\ l_i,\ k') + FC(j,\ l_o,\ l_i) \ \} \end{aligned} \tag{19}$$

Proof: The optimal placement of k caches in the segment $[0, j]$, can either put a cach at the jth location and $k-1$ caches in the sement $[0, j-1]$, or put all k caches in the sement $[0, j-1]$. 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 algorithms is $O(n^3)$ to compute the base case, and $O(n^3 \cdot k)$ to compute $C(j,\ l_o,\ l_i,\ k')$.

4.2. Single interface cache. The optimal location of k unidirectional (or single interface) caches can be found by solving separately the two directions and looking for the optimal combination. More formally, let $C^r(k')$ be the overall flow in the optimal solution when k' caches are placed for the flow in the right direction, and let $C^l(k')$ be the overall flow in the optimal solution for the left direction. The optimal location of k unidirectional caches is given by:

$$C^{opt}(k) = \min_{i=0}^{k} C^r(i) + C^l(k-i)$$

In the single interface case the distance for the next cache downstream does not effect the overall cost in the segment $[0, j]$. Therefore we define $C^r(j,\ k',\ l)$ be the overall flow in the optimal placement of k' single-interface caches in the segment of $[0, j]$ where the next upstream cache is at location l. Then $C^r(j,\ k',\ l)$, for $j > 1$, is given by the expression (see Figure 6):

$$C^r(j,\ k',\ l) = \min\{C^r(j-1,\ k',\ l),\ C^r(j-1.\ k'-1,\ j)\} + AF^r(i,\ l)$$

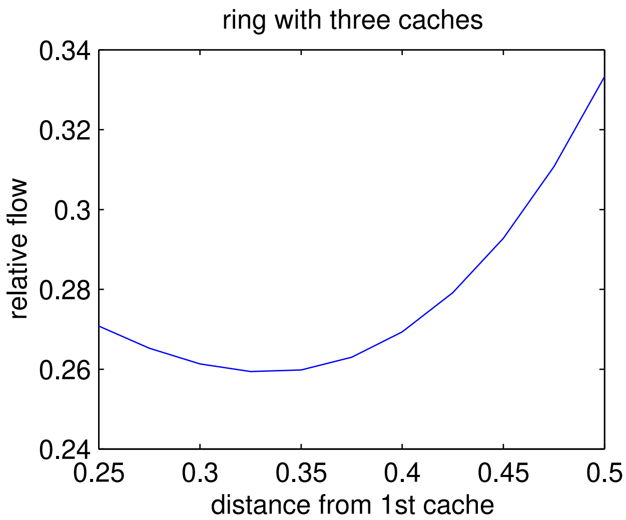

FIGURE 7. The relative flow in a ring network with 3 caches as a function of the distance of two caches from the third. (assuming the distances of the two caches from the first are the same.)

where $AF^r(j,\ l)$ is the adjusted flow through link $(j-1,\ j)$ given that the closest upstream cache is at location l. the algorithm, and the proof of its correctness are very similar to the multiple interface case.

The complexity of computing AF^r is $O(n^2)$. The complexity of computing C^r is $O(n^2 \cdot k)$. The overall complexity of computing the optimal solution for the single-interface cache location problem is, thus, $O(n^2 \cdot k)$, since the final stage of comparing the k possible combination is negligible compared with calculating the directional costs.

5. 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 7 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 colocated. 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, p, and whether the hit probability in one cache is independent or fully correlated, the caches should be still 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. 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 20.

$$\begin{aligned} \mathcal{F} \quad = \quad & \int_0^x \int_s^x t - s\, dt\, ds + \int_0^x \int_x^1 t - x\, dt\, ds + \int_0^x \int_0^s t\, dt\, ds + \\ & \int_x^1 \int_x^s t - x\, dt\, ds + \int_x^1 \int_s^1 t - s\, dt\, ds + \int_x^1 \int_0^x t\, dt\, ds \;=\; \frac{2 - 3\,x + 3\,x^2}{6} \end{aligned} \tag{20}$$

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

$$\begin{aligned} \mathcal{F} &= \int_0^x \int_0^t t - s \, ds \, dt + \int_0^x \int_t^1 t \, ds \, dt + \\ &\int_x^y \int_x^t t - s \, ds \, dt + \int_x^y \int_t^{1+x} t - x \, ds \, dt = \frac{3\,x^2\ (2-y) - 3\,x\ (2-y)\ y + (3-y)\ y^2}{6} \end{aligned} \tag{21}$$

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 the previous section. As we mentioned, the first located cache breaks the ring into a line. Therefore, we can ran the algorithms of Section 4, for any possible first cache location with an additional factor of $O(n)$ to their complexity.

6. 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 the special case of a line network. We have also demonstrated the difficulty of the problem by comparing the optimal solution with a greedy solution.

The factor n difference in the complexity of the solutions for unidirectional and bidirectional general line problems hints that a better solution for the bidirectional cache location may be found. The solution for a general network is $\mathcal{NP}$-hard. However, the good performance of the greedy algorithm on a line suggests that finding good approximations is an achievable open questions.

References

[Bes95] Azer 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, October 1995.

[Bes96] Azer 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, March 1996.

[BLFF96] T. Berners-Lee, R. Fielding, and H. Frystyk. Hypertext Transfer Protocol – HTTP/1.0, May 1996. RFC 1945.

[CDN+96] A. Chankhunthod, P. B. Danzig, C. Neerdaels, M. F. Schwartz, and K. J. Worrell. Hierarchical internet object cache. In *Usenix Technical Conference*, 1996.

[CF98] Gruia Călinescu and Cristina G. Fernandes. Multicuts in bounded-degree trees. submitted, 1998.

[Cis97] Shared network caching and cisco's cache engine. white paper, September 1997. `http://www.cisco.warp/public/751/cache/cds_wp.htm`.

[DFKM97] F. Douglis, A. Feldman, B. Krishnamurthy, and J. Mogul. Rate of change and other metrics: A live study of the world wide web. In *Proceedings of the First Usenix Symposium on Internet Technologies and Systems (USITS)*, December 1997.

[DHS93] Peter B. Danzig, Richard S. Hall, and Michael F. Schwartz. A case for caching file objects inside internetworks. In *SIGCOMM*, pages 239 – 243. ACM, September 1993.

[EK96] Dawson R. Engler and M. Frans Kaashoek. Dpf: Fast, flexible message demultiplexing using dynamic code generation. In *ACM SIGCOMM'96*, August 1996. Stanford University, CA, USA.

[FGM+97] Roy T. Fielding, Jim Gettys, Jeffrey C. Mogul, Henrik Frystyk Nielsen, and Tim Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1, January 1997. RFC 2068.

[GJ79] Michael R. Garey and David S. Johnson. *Computer & Intractability: A Guide to the Theory of NP-Completeness.* W H Freema, November 1979.

[GL91] Mark W. Garrett and San-Qi Li. A study of slot reuse in dual bus multiple access networks. *IEEE Journal on Selected Areas in Communications*, 9(2):248 – 256, February 1991.

[GRC97] 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.

[HM97] Abdelsalam Heddaya and Suliman 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.

[KH79] 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.

[KRS98] P Krishnan, Dan Raz, and Yuval Shavitt. Transparent en-route cache location. 1998. submitted.

[KS97] P. Krishnan and Binay Sugla. Utility of co-operating web proxy caches. In *Proceedings of the 7th International World Wide Web Conference*, Brisbane, Australia, April 1997.

[Mel96] Ingrid Melve. Why internet service providers should integrate web caches into their networks. In *web caching in the internet conference*, sep/oct 1996.

[MLB95] R. Malpani, J. Lorch, and D. Berger. Making world wide web caching servers cooperate. In *Proceedings of the Fourth International World Wide Web Conference*, December 1995.

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

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

[NSS95] Bhagirath Narahari, Sunil Shende, and Rahul Simha. Efficient algorithms for erasure node placement on DQDB networks. In *ICC'95*, pages 935 – 939, June 1995. Seattle, Washington, USA.

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

[Tam96] Arie Tamir. An $O(pn^2)$ algorithm for the p-median and related problems on tree graphs. *Operations Research Letters*, 19:59 – 64, 1996.

[TSS+97] David L. Tennenhouse, Jonathan M. Smith, W. David Sincoskie, David J. Wetherall, and Gary J. Minden. A survey of active network research. *IEEE Communications Magazine*, 35(1):80–86, January 1997.

[Use97] Proceedings of the usenix symposium on internet technologies and systems, December 1997.

[WAS+96] 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.

[ZFJ97] Lixia Zhang, Sally Floyd, and Van Jacobson. Adaptive web caching. In *NLANR Web Cache Workshop*, June 1997. `http://www-nrg.ee.lbl.gov/floyd`.

Bell Laboratories, Lucent Technologies, 101 Crawfords Corner Road, Holmdel, NJ 07733-3030, USA

E-mail address: `{pk, raz, shavitt}@lucent.com`