Constrained Mirror Placement on the Internet

Eric Cronin, Sugih Jamin, Cheng Jin, Anthony R. Kurc, Danny Raz, *Member, IEEE*, and
Yuval Shavitt, *Senior Member, IEEE*

***Abstract*—Web content providers and content distribution network (CDN) operators often set up mirrors of popular content to improve performance. Due to the scale and decentralized administration of the Internet, companies have a limited number of sites (relative to the size of the Internet) where they can place mirrors. We formalize the mirror placement problem as a case of constrained mirror placement, where mirrors can only be placed on a preselected set of candidates. We study performance improvement in terms of client round-trip time (RTT) and server load when clients are clustered by the autonomous systems (AS) in which they reside. Our results show that, regardless of the mirror placement algorithm used, for only a surprisingly small range of values is increasing the number of mirror sites (under the constraint) effective in reducing client to server RTT and server load. In this range, we show that greedy placement performs the best.**

***Index Terms*—Constrained mirror placement, Internet experiments, performance analysis, placement algorithms.**

I. Introduction

THERE ARE a growing number of frequently accessed Web sites that employ mirror servers to increase the reliability and performance of their services. Mirror servers, or simply "mirrors," replicate the whole content or the most popular content of a web server, or "server." A client requesting the server's content is then redirected to one of the mirrors (we consider co-located mirrors to be a single mirror). Since each mirror sees only a portion of the total requests, clients can be served faster; furthermore, if clients are redirected to mirrors closer to them than the server, download times can be reduced (a more formal argument will be presented in Section III-A).

At first glance, Web caches appear to serve the same purpose as mirrors. We differentiate mirrors from caches in that client access to a mirror never results in a "miss." A client is redirected to a mirror only when the mirror has the requested content. Accesses to Web caches, on the other hand, can result in cache misses. In addition, mirrors can also serve some forms of dynamic content and content customized for each client.

To keep mirrors' content consistent, synchronization between mirrors and the main server is required whenever the main server's content changes. Various algorithms to keep Web caches consistent have been proposed in the literature and may be applicable to mirrors. We classify these algorithms into two categories: those based on time-to-live [1], and those based on server invalidation [2]. Without going into the details of the algorithms, we note that the cost of keeping mirrors consistent, in terms of the amount of traffic seen at the server (in the case of [1]) or the total amount of traffic seen on the network (in the case of [2]), increases linearly with the number of mirrors. Thus, even if one assumes that larger number of mirrors provide further reduction in server load or client download time, simply increasing the number of mirrors with impunity will result in higher consistency cost. Certainly, one would be willing to pay the cost associated with a large number of mirrors if it would be outweighed by the reduction in the overall system cost. We show in this paper, however, assuming that clients access the mirror which lowers their download time the most, increasing the number of mirrors beyond a certain value does not significantly reduce server load nor client download time. Obviously, we are not considering the case where there is a mirror on every client host or local area network (LAN).

Given a finite number of mirrors, we are interested in where to place them to maximize performance. A content distribution network (CDN), for instance, may have a large number of machines scattered around the Internet capable of hosting mirrors. A content provider with a busy Web server can rent resources on these machines to host their mirrors. The question is then: on which subset of the candidate machines should a content provider put mirrors of its content? Ideally, a mirror can be placed where there is a cluster of clients interested in the content of the server [3]. We only consider a model in which there is a fixed number of candidate sites where mirrors can be placed. We call this the constrained mirror placement (CMP) problem. We discuss some of the current works in the area of mirror placement in Section II. We then give a formal definition of the CMP problem in Section III and look at various mirror placement algorithms and heuristics. We describe our simulation and Internet experiments in Section IV and our results in Section V. We conclude and discuss future works in Section VI.

II. Related Work

There have been some recent works on mirror performance and closest server selection. Myers *et al.* [4] measured nine clients scattered throughout the United States retrieving

Manuscript received May 1, 2001; revised February 1, 2002. This work was supported in part by the National Science Foundation (NSF) under Grant ANI-9876541 and under a Grant from the United States—Israel Binational Science Foundastion (BSF), Jerusalem, Israel, and in part by MCI Worldcom, in part by Lucent Bell-Labs, in part by Fujitsu Laboratories America, and in part by equipment grants from Sun Microsystems Inc. and Compaq Corporation. The work of S. Jamin was supported by the NSF CAREER Award ANI-9734145, the Presidential Early Career Award for Scientists and Engineers (PECASE) 1999, and by the Alfred P. Sloan Research Fellowship 2001. This paper was presented at the IEEE INFOCOM 2001, Anchorage, AK.

E. Cronin, S. Jamin, C. Jin, and A. R. Kurc are with the Electrical Engineering and Computer Science Department, University of Michigan, Ann Arbor, MI 48109 USA (e-mail: ecronin@eecs.umich.edu; jamin@eecs.umich.edu; chengjin@eecs.umich.edu; tkurc@eecs.umich.edu).

D. Raz is with the Computer Science Department, Technion–Israel Institute of Technology, Haifa, Israel and also with Bell Labs, Lucent Technologies, Holmdel, NJ 07733 USA.

Y. Shavitt is with the Department of Electrical Engineering-Systems, Tel-Aviv University, Tel-Aviv, Israel and with Bell Labs, Lucent Technologies, Holmdel, NJ 07733 USA (e-mail: shavitt@eng.tau.ac.il).

Publisher Item Identifier 10.1109/JSAC.2002.802066.

documents from 47 Web servers, which mirrored three different Web sites. Fei *et al.* [5] present a server selection technique that can be employed by clients on end hosts. The technique involves periodic measurements from clients to all of the mirrors of a server. Seshan *et al.* [6] proposed a server selection scheme based on shared passive end-to-end performance measurements collected from clients in the same network. There are also related works that focus on maintaining consistency among cache servers, which can be applicable in keeping mirrors consistent [2], [1]. These works studied different scalable Web cache consistency approaches and showed various overhead of keeping caches consistent.

Jamin *et al.* [7] used two graph theoretic algorithms, k-HST [8] and $\min K$-center [9], to determine the number and the placement of *instrumentation boxes* for the purpose of network measurement. The authors demonstrated the usefulness of a network distance service by showing that the distance map computed can be used to redirect clients to the closest (in latency) of three server mirrors. While they used closest mirror selection as a motivating problem, the three mirrors they consider are manually placed at arbitrarily selected locations. In this paper, we take a closer look at mirror placement on the Internet under a more realistic setting where the number of mirrors is small, but generally larger than three, and the placement is restricted to a given set of hosts. Krishnamurthy and Wang [3] proposed a scheme to group nearby Web clients into clusters and evaluated it to be highly effective. They further proposed and evaluated schemes for proxy placement where a proxy is placed inside each such client cluster. In parallel to an earlier version of our work [10], Qiu *et al.* [11] studied placing M replicas on N client clusters to maximize performance. Various placement schemes were proposed and evaluated against a "super-optimal" algorithm, which provided the performance lower bound for the optimal placement. The placement algorithms were evaluated on artificially generated topologies as well as the Internet autonomous systems (AS) topology. The authors concluded that a greedy algorithm based on client cluster provided performance that was close to the optimal solution. The authors focused on finding the best placement algorithm/heuristic given a certain constraint, while our paper focuses on the performance limitations of all placement algorithms under the constrained setting. We show that even the best placement gives almost no performance improvements after mirrors are placed on 20% of candidate sites.

III. Constrained Mirror Placement

We model the Internet as a graph, $G = (V, E)$, where V is the set of nodes and $E \subseteq V \times V$ the set of links. We define $\mathcal{H} \subseteq V$ to be the set of candidate hosts where mirrors can be placed, $\mathcal{M} \subseteq \mathcal{H}$ the set of mirrors of a particular server and $\mathcal{B} \subseteq V$ the server's clients. The objective of the CMP problem is to place the set of mirrors on the set of candidate hosts such that some optimization condition $\mathcal{OC}(\mathcal{M}, p)$ (defined in Section III-A) is satisfied for the client set. How well the optimization condition is satisfied depends on the sizes and topological placements of the candidate hosts and client sets. We denote the sizes of the candidate host, mirror, and client sets as $|\mathcal{H}|$, $|\mathcal{M}|$, and $|\mathcal{B}|$, and their topological placements as $\mathcal{P}(\mathcal{H})$, $\mathcal{P}(\mathcal{M})$, and $\mathcal{P}(\mathcal{B})$, respectively. We use the notation $\mathcal{H}$, $\mathcal{M}$, and $\mathcal{B}$ to denote a specific size and placement of the sets. The constrained mirror placement problem can now be formally stated.

Definition 1: Given a graph G, a set of candidate hosts, $\mathcal{H}$, a positive integer k, and an optimization condition $\mathcal{OC}(\mathcal{M}, p)$, the CMP problem is to find a set of mirrors, $\mathcal{M}$, of size k such that $\mathcal{OC}(\mathcal{M}, p)$ is minimized.

We include $\mathcal{M}$ as part of the notation for $\mathcal{OC}(\mathcal{M}, p)$ to emphasize that we are studying the effect of changing $\mathcal{M}$ on the performance of CMP. Specifically, we study the effect of changing $|\mathcal{M}|$ and $\mathcal{P}(\mathcal{M})$ while holding $|\mathcal{H}|$ constant, with $\mathcal{H} \cap \mathcal{B} = \emptyset$, and $\mathcal{H} \cup \mathcal{B} \subseteq V$. We experiment with $\mathcal{P}(\mathcal{H})$ uniformly distributed and on nodes with the highest outdegrees (outgoing links). We also experiment with both uniformly distributed and trace-based $\mathcal{P}(\mathcal{B})$. A major difference between our formulation of the problem and the one in [11] is that they assume mirrors can be placed within client clusters, i.e., $\mathcal{H} \subseteq \mathcal{B}$. We do not consider it realistic for a CDN to always be able to place mirrors inside client clusters.

A. Optimization Condition $\mathcal{OC}(\mathcal{M}, p)$

We identify two goals commonly associated with placing mirrors on the Internet: reducing client download time and alleviating server load. In the previous section, we mentioned the cost of keeping mirrors consistent as a limiting factor in deploying a large number of mirrors. We will show that even discounting consistency cost, increasing the number of mirrors beyond a certain number does not significantly reduce client download time, or assuming that clients access mirrors with the lowest client server round-trip time (RTT), distribute server load. Without loss of generality, we assume zero cost to keep mirrors consistent for the remainder of this paper. With zero consistency cost, we can treat the server itself as simply one of the mirrors. Assuming one can add a mirror with no cost, we ask, "By how much adding one more mirror reduces client download time and alleviates load at existing mirrors (including the server)?" Client download time can be affected by factors such as load at mirrors, bottleneck bandwidth, network latency (in terms of RTT), etc.

We focus primarily on the network latency factor and consider reducing RTT as our *sole* optimization condition, $\mathcal{OC}(\mathcal{M}, p)$. From a theoretic standpoint, network latency is the most difficult factor to improve since it is limited by the speed of light. A heavily loaded mirror can always be better provisioned to meet the load requirements, e.g., by forming a server cluster [12] (content providers and CDNs have incentives to ensure that there is enough provisioning), and bottleneck bandwidth may be upgraded[1]; however, we cannot "upgrade" latency—in the same sense that we do for server load and bandwidth—by simply "adding hardware." From a practical standpoint, transmission control protocol (TCP), the underlying transport protocol for Web download, has well-known biases against connections with long RTTs [13]. Routers drop packets when there is network congestion. Upon detection of network

[1] There are certain financial constraints associated with such upgrades, but no inherent technical constraints.

congestion, TCP backs off its transmission window size and slowly increases the window again based on successfully acknowledged transmissions. Connections with longer RTTs, thus, experience longer congestion recovery periods and lower throughput [14]. In this paper, we study the use of the maximum, 95 percentile, and mean client–mirror distance[2] in the optimization conditions, denoted as $\mathcal{OC}(\mathcal{M}, 1)$, $\mathcal{OC}(\mathcal{M}, 0.95)$, and $\mathcal{OC}(\mathcal{M}, \mu)$, respectively. We do not factor in the time it takes for a client to find its closest mirror because it can be amortized over the number of client-to-server requests, and any mechanism that improves this transaction can be equally applied to any redirection scheme.

In order to direct clients to the closest mirrors, we need to know the distances between each client and all of the mirrors. If the network topology is known, the closest mirror to any client can be identified, for example, by computing the shortest path from the client to all mirrors, using Dijkstra's shortest path algorithm. When network topology is not known, such as in the case of the Internet, client redirection can be done randomly. Jamin *et al.* [7] showed that closest mirror selection using a distance map[3] invariably gives better performance than random selection. In comparing various mirror placement algorithms, we require that the distances between candidate sites and clients are known (in the case of $\min K$-center, all pairs of distances must be known), and the closest mirror to a client can be deterministically computed. Obviously, Internet topology is *not* known *a priori*. In order to apply the placement algorithms we, thus, need to first construct a virtual topology of the Internet. In Section IV-B2, we present a methodology to construct a virtual topology of the Internet.

B. Mirror Placement Algorithms and Heuristics

We now present three graph-theoretic algorithms and two heuristics that we use in placing mirrors. We look at placement algorithms that can optimize for all three of our performance metrics such as cost-adjustable set cover and ℓ-greedy, as well as $\min K$-center that optimizes exclusively for $\mathcal{OC}(\mathcal{M}, 1)$. We also look at two heuristics that do not require topological knowledge of the network. In the subsequent discussion, in accordance to the terminologies used in the literature, we use the term "center" instead of "mirror."

1) $\mathrm{Min}\, K$-Center: $\mathrm{Min}\, K$-center is a graph-theoretic algorithm that finds a set of center nodes to minimize the maximum distance between a node and its closest center. Given this definition, the $\min K$-center problem is relevant only in the case of optimization condition $\mathcal{OC}(\mathcal{M}, 1)$. The $\min K$-center problem is known to be NP-complete [15]; however, a 2-approximate algorithm exists [9]. With the 2-approximate algorithm, the maximum distance between a node and its nearest center is no worse than twice the maximum in the optimal case. For ease of reference, we include here our summary of the 2-approximate algorithm presented in [7].

```
Algorithm 1 (2-approximate min $K$-center [9])

1. Construct $G_1^2, G_2^2, \ldots, G_m^2$
2. Compute $M_i$ for each $G_i^2$
3. Find smallest $i$ such that $|M_i| \leq K$, say $j$
4. $M_j$ is the set of $K$ centers
```

Fig. 1. Two-approximate algorithm for the $\min K$ -center problem.

The algorithm receives as input a graph $G = (V, E)$ where V is the set of nodes, $E = V \times V$, and the cost of an edge $e = (v_1, v_2) \in E$, $c(e)$, is the cost of the shortest path between v_1 and v_2. All the graph edges are arranged in nondecreasing order by cost, c: $c(e_1) \leq c(e_2) \leq \cdots \leq c(e_m)$, let $G_i = (V, E_i)$, where $E_i = \{e_1, e_2, \ldots, e_i\}$. A *square graph* of G_i, G_i^2, is the graph containing V and edges (u, v) wherever there is a path between u and v in G_i of at most two hops, and $u \neq v$. An *independent set* of a graph $G = (V, E)$ is a subset $V' \subseteq V$ such that, for all $u, v \in V'$, the edge (u, v) is *not* in E. An independent set of G_i^2 is, thus, a set of nodes in G_i that are at least three hops apart. We also define a *maximal* independent set M as an independent set V' such that all nodes in $V - V'$ are at most one hop away from nodes in V'.

The outline of the $\min K$-center algorithm from [9] is shown in Fig. 1. The basic observation is that the cost of the optimal solution to the K-center problem is the cost of e_i, where i is the smallest index such that G_i has a dominating set[4] of size at most K. This is true since the set of center nodes is a dominating set, and if G_i has a dominating set of size K, then choosing this set to be the centers guarantees that the distance from a node to the nearest center is bounded by e_i. The second observation is that a star topology in G_i transfers into a clique (full mesh) in G_i^2. Thus, a maximal independent set of size K in G_i^2 implies that there exists a set of K stars in G, such that the cost of each edge in it is bounded by $2e_i$: the smaller the i, the larger the K. The solution to the $\min K$-center problem is the G_i^2 with K stars. Note that this approximation does not always yield a unique solution.

We have to make further approximations in applying the $\min K$-center algorithm to the CMP problem. In the construction of the $\min K$-center algorithm above, any node in G may be selected to act as a "center." In CMP, only nodes in $\mathcal{H}$ can host mirrors. Thus, to apply the $\min K$-center algorithm, we first run the algorithm on G with $V = \mathcal{H} \cup \mathcal{B}$. Should a node in $\mathcal{B}$ be selected as a center, we substitute it with a node in $\mathcal{H}$ that is closest to it. Recall that we assume $\mathcal{H} \cap \mathcal{B} = \emptyset$.

2) ℓ-Greedy: This algorithm places mirrors on the network iteratively in a greedy fashion. First it exhaustively checks each node in $\mathcal{H}$ to determine the node that best satisfies the optimization condition (see Section III-A) for a given $\mathcal{B}$. For $\ell = 0$, after assigning the first mirror to this node, the algorithm greedily looks for the best location for the next mirror, etc., until all $|\mathcal{M}|$ mirrors are placed. For any other ℓ value, the algorithm allows for ℓ steps backtracking: it checks all the possible combinations of removing ℓ of the already placed mirrors and replacing them

[2] We use the term "client–mirror distance" to mean the distance between client and the closest mirror.

[3] By "distance map" we mean a virtual topology of the Internet constructed by tracing paths on the Internet. An architecture to build such a distance map was proposed in [7].

[4] A dominating set is a set of D nodes such that every $v \in V$ is either in D or has a neighbor in D.

```
Algorithm 2 ($\ell$-Greedy [16])

if ($|\mathcal{M}| \le \ell$)
    Choose among all sets $\mathcal{M}'$ with $|\mathcal{M}'| = |\mathcal{M}|$
      the set $\mathcal{M}''$ with minimal $\mathcal{OC}(\mathcal{M}'', p)$
    return set $\mathcal{M}''$
end
Set $\mathcal{M}'$ to be an arbitrary set of size $\ell$
while ($|\mathcal{M}'| < |\mathcal{M}|$)
    Among all sets $X$ of $\ell$ elements in $\mathcal{M}'$
      and among all sets $Y$ of $\ell + 1$ elements
        in $V - \mathcal{M}' + X$, choose the sets $X$, $Y$
        with minimal $\mathcal{OC}(\mathcal{M}' - X + Y, p)$
    $\mathcal{M}' = \mathcal{M}' - X + Y$
end
return set $\mathcal{M}'$
```

Fig. 2. Algorithm *ℓ-greedy*.

```
Algorithm 3 (gsc($U$, $S$, $cost(S)$))

$C = \emptyset, X = \emptyset$
$\forall s \in S$
    $\alpha(s) = \frac{cost(s)}{|s|}$
While ( $C \neq U$ )
    select $s$ such that $\alpha(s)$ is minimized.
    $C = C \cup s, X = X + s, S = S - s$
    $\forall s \in S$
        $\alpha(s) = \frac{cost(s)}{|s \cap C|}$
$X$ is the set cover
```

Fig. 3. Greedy set-cover algorithm.

with $\ell + 1$ new mirrors. That is, ℓ of the already placed mirrors can be moved around to optimize the gain. Fig. 2 summarizes the algorithm.

3) Cost-Adjustable Set Cover: The setup of the set-cover problem is as follows: given several subsets, each covering a different set of elements, find the minimum number of these subsets such that all elements are included or "covered." Since a set's members may overlap with other sets' members, some elements may be covered by several sets. In our case, the elements are the clients of a Web server. This problem is NP-complete [15]. Vazirani [9] gives a greedy approximate solution to the minimum set-cover problem. We now describe this greedy algorithm and explain how we apply it to the CMP problem.

In the greedy set-cover algorithm, each set s is associated with a "cost-effectiveness," $\alpha(s)$, which is the average cost of the set. The greedy algorithm selects the most cost-effective set iteratively until all elements are covered. The "cost-effectiveness" of a set is computed by dividing the cost associated with the set $cost(s)$ by the number of not-already-covered members of the set. Recall that a set's members may overlap with other sets' members and, therefore, some elements may be covered by several sets. Fig. 3 shows the outline of the greedy set-cover algorithm. Given a universe, U, of elements and a collection of subsets, S, the algorithm finds the subsets X to cover U by selecting the set with the minimum $\alpha(s)$ at each iteration. In the description of the algorithm, we use "$+$" to indicate the addition of an element (the second operand) to an existing set, "$-$" the deletion of an element (the second operand) from an existing set, and "$\cup$" the union of two sets.

```
Algorithm 4 (cost-adjustable gsc($\mathcal{H}$, $\mathcal{B}$, $\kappa$))
$\mathcal{S} = \emptyset$
$\forall h \in \mathcal{H}$
    sort $\mathcal{B}$ based on client-candidate distance
    $s = \emptyset$
    $\forall b \in \mathcal{B}$
        $s = s + b, \mathcal{S} = \mathcal{S} + s$
$\forall s \in \mathcal{S}$
    $cost(s) = \mathcal{OC}(\mathcal{M}, p) + \kappa$
$\mathcal{M} = $ gsc($\mathcal{B}, \mathcal{S}, cost(\mathcal{S})$)
```

Fig. 4. Cost-adjustable greedy set-cover algorithm.

Ideally, we would apply the greedy set cover to the client sets and obtain a set cover which is a collection of subsets, and we can place a mirror in or near each such subset. However, under constrained mirror placement, we cannot directly apply the greedy set-cover algorithm for two reasons. First, we do not have the collection of subsets, S. Second, the greedy set-cover algorithm only produces the "minimum" set cover with a fixed number of mirrors, without allowing us the same flexibility of placing a variable number of mirrors as in the $\min K$-center or ℓ-greedy algorithm.

In order to apply the greedy set cover problem to constrained mirror placement, we need to define the collection, S, of subsets of clients, and the cost, $cost(s)$, associated with each subset. We obtain the subsets of clients by constructing sets of clients centered at each candidate site. For each candidate site, we order the clients based on their distances from the candidate site. We then add one client at a time to form a separate subset. One can think of these subsets as concentric circles, each with one more client added to the immediately preceding subset. The cost of each such subset is simply the performance metric $\mathcal{OC}(\mathcal{M}, p)$ of the subset. The use of $\mathcal{OC}(\mathcal{M}, p)$ as the cost of a subset is consistent with the objective of cost-effectiveness since we want to minimize the performance metric for as many clients as possible. A small performance metric along with a large set cardinality makes a set cost-effective.

In the context of CMP, we need to examine the effect of different numbers of mirrors. Since the greedy algorithm can only return the "minimum" number of subsets, we need to introduce a parameter to the algorithm that allows us to tune the algorithm to produce the placement of any desired number of mirrors. We observe that the greedy algorithm uses the cost-effectiveness of each set to decide which sets are included in the set cover; therefore, we can vary the cost-effectiveness of each set to favor set covers of smaller or larger sizes than the "minimum," hence producing the placement of a larger or smaller number of mirrors. We introduce an additive parameter κ to the set cost, $cost(s)$, which we can use to vary the set costs. By increasing κ, we make larger client sets more attractive since the cost-effectiveness, which is $cost(s)$ divided by the number of elements in s, increases more for client sets of smaller sizes. By the same argument, we can make smaller client sets more attractive by decreasing κ. We call this variant of the set cover problem the cost-adjustable set cover problem. We show the outline of the cost-adjustable set cover in Fig. 4.

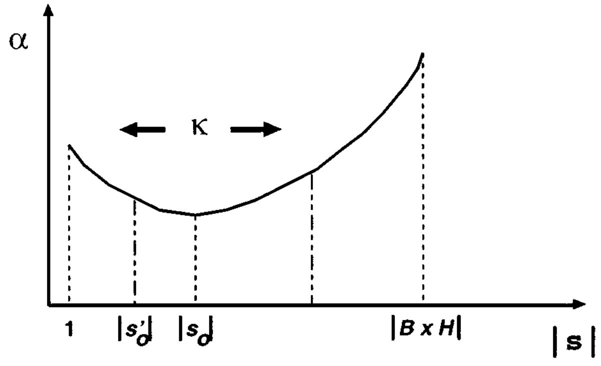


Fig. 5. Ideal cost-effectiveness versus set size.

We now discuss how κ affects the overall performance of the cost-adjustable greedy set-cover algorithm. In particular, we study the relationship between κ and α, the cost-effectiveness, and show how the relationship affects the solution to the cost-adjustable greedy algorithm. Fig. 5 shows a sample curve of cost-effectiveness, α, versus the client subset size, $|s|$. We note that the shape of such a curve will generally be concave because α is the sum of two monotonic functions of different trends. The first function, $\kappa/|s|$, is a monotonically decreasing function with respect to $|s|$, and the second function, $\mathcal{OC}(\mathcal{M}, p)/|s|$, is a monotonically nondecreasing function with respect to $|s|$ for all $\mathcal{OC}(\mathcal{M}, p)$s because marginal increase (when a new client is included) in $\mathcal{OC}(\mathcal{M}, p)$ is nonnegative. The sum of the two functions may not be monotonic with respect to $|s|$ at all times, depending on the magnitude of κ. By increasing κ, the first function $\kappa/|s|$ will increase smaller sets' αs more than larger sets, thus, making larger sets more attractive to the greedy algorithm, and decreasing κ makes smaller sets more attractive.

The parameter κ introduces concave instances in the "α versus $|s|$" curve, which make subsets of rather different sizes appear to have identical or similar costs. A horizontal line cutting across the curve would intersect the "α versus $|s|$" curve at two different points, which correspond to two subsets of different sizes. This does not present a problem since neither intersection would have the minimum α. Thus, we use this parameter κ to "force" the algorithm to produce the set with size k. If we get a smaller-than-expected set cover, we increase κ, and in case of a large set cover (with respect to k), we decrease κ. We stop the algorithm when we get the desired set-cover size, or if getting the exact cover size is not possible, the largest set-cover size that is smaller than the desired value.

4) Heuristics: In addition to the above placement algorithms, we also look at the following heuristics that do not require knowledge of the network topology.

- Transit Node: The outdegree of a node is the number of other nodes it is connected to. Assuming that nodes with the highest outdegrees can reach more nodes with lower latencies, we place mirrors on candidate hosts in descending order of outdegree. We call this the *transit node* heuristic under the assumption that nodes in the core of the Internet that act as transit points will have the highest outdegrees.
- Random Placement: Under random placement, each candidate host has a uniform probability of hosting a mirror.

C. Performance Analysis

In this section, we present an analysis of the performance of unconstrained mirror placement to illustrate what could be expected of mirror placement in the ideal setting. In particular, the analysis shows that, under optimal mirror placement, there is a diminishing return in reducing client–mirror distance[5] with respect to the number of mirrors, which agrees with our intuition. The analysis also shows that the ratio of expected maximum client–mirror distance between optimal and random placement increases logarithmically; however, under random placement, most clients are still close enough to their closest mirrors, and only a small portion of the clients are actually very "far" from their closest mirrors as the number of mirrors increases. This further illustrates the diminishing return in using the optimal placement as the number of mirrors increases.

To abstract the unconstrained mirror placement problem, we can picture the network as a continuous plane on which clients can be uniformly spread over the infinitely many points ($\aleph$). We want to place a given number of mirrors such that the maximum distance of any client to its closest mirror is minimized. This measure of quality translates into finding a placement such that the radius of the largest circle one can draw in the plane that does not include any mirror is minimized.

Solving this problem analytically is cumbersome, instead we study the same problem in one dimension only. We can transform the problem into one dimension by distributing the clients uniformly on the segment (0, 1) and placing mirrors on the same segment. Clearly, the optimal allocation of mirrors given the maximum distance criterion is to separate the mirrors by the same distance apart. Thus, if one needs to place $n-1$ mirrors, the optimal location is at locations i/n, $1 \leq i \leq n-1$, and the maximum distance from any client to its closest mirror is $1/n$.[6] It is clear that there is diminishing return in client–mirror distance as the number of mirrors increases. We can also see that each mirror site will have approximately the same number of clients if each client is directed to its closest mirror.

The optimal placement could be difficult to achieve in real life. Hence, we would like to quantify how good random placement is compared with the optimal placement in terms of the expected maximum client–mirror distance. Under random placement, $n-1$ points (mirrors) are uniformly distributed in the interval (0, 1). Now, let $Y_{(n)}$ be the random variable representing the longest segment length, $f_{Y_{(n)}}$ the density function, and $F_{Y_{(n)}}$ the cumulative distribution function. Using known results from order statistics [17, Sec. 5.4], we have

$$\Pr\left\{Y_{(n)} > y\right\} = \sum_{1-iy>0,\, i\geq 1} (-1)^{i-1}\binom{n}{i}(1-iy)^{n-1}. \quad (1)$$

[5] We use the term client–mirror distance to mean the distance between client and the closest mirror.

[6] The actual optimal locations for n mirrors should be at $(1/2n)+(i/n)$, but the importance of this boundary condition diminishes with n. For ease of analysis, we consider only the limit case with n going to infinity.

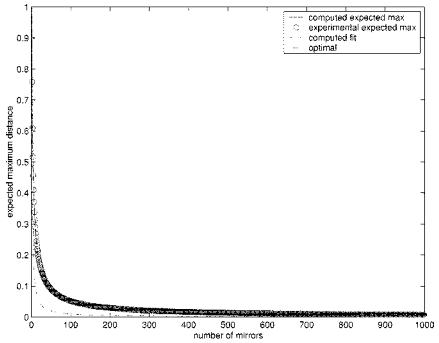


Fig. 6. Expected maximum segment length on the unit interval.

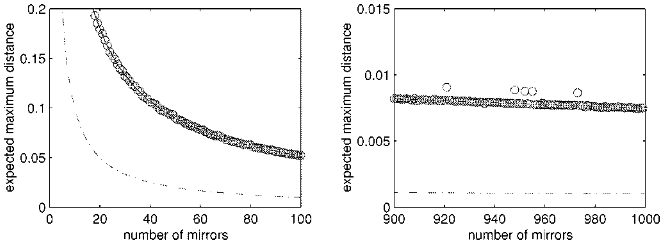


Fig. 7. Expected maximum segment length on the unit interval (details).

The expected value of the maximum segment between two neighboring points is, thus, given by

$$\begin{aligned}E\left[Y_{(n)}\right] &\triangleq \int_0^1 f_{Y_{(n)}}(y)y\,dy\\ &= \int_0^1 \frac{d}{dy}\left(F_{Y_{(n)}}(y)y\right)dy - \int_0^1 F_{Y_{(n)}}(y)\,dy\\ &= F_{Y_{(n)}}(y)y\Big|_0^1 - \int_0^1 \Pr\left\{Y_{(n)} \le y\right\}dy\\ &= 1 - \int_0^1 \left[1 - \Pr\left\{Y_{(n)} > y\right\}\right]dy\\ &= \int_0^1 \Pr\left\{Y_{(n)} > y\right\}dy\\ &= \sum_{1-iy>0,\, i\ge 1} (-1)^i \binom{n}{i} \frac{(1-iy)^n}{in}\Bigg|_0^1\\ &= \sum_{i=1}^{n} (-1)^{i+1}\binom{n}{i}\frac{1}{in}.\end{aligned}$$

Fig. 6 depicts the computed expected maximum segment length together with numerical simulation results. Each point in the simulation represents the mean of 1000 experiments; in each, $n-1$ points are randomly placed on the unit interval with uniform probability and the maximum segment is computed. The confidence interval is negligible in most cases. It is clear that the simulation and the numerical calculation are almost identical. The detailed enlargement in Fig. 7 (also in Fig. 8) shows that some outliers are observable in different scales. There is a clear knee around $n = 60$ after which the return from adding additional mirrors diminishes. Comparing the segment length to the optimal length shows that for a large range, $n < 150$, the difference is substantial.

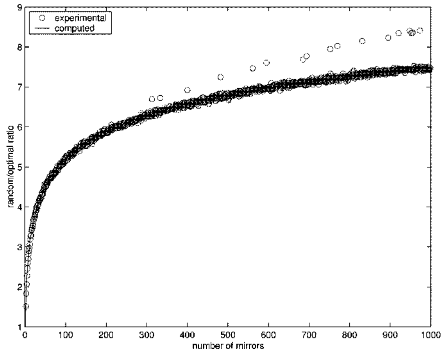


Fig. 8. Ratio of the random placement over the optimal placement.

Fig. 8 shows the ratio of expected maximum segment length between the random placement and the optimal for both the simulated data and the calculated data. Surprisingly, it seems that the ratio increases logarithmically with the number of mirrors (we saw before that the absolute difference diminishes). To check this we fitted the exponent of the ratio with the best (mean square) linear function of the form $\alpha + \beta n$. The resulting fitted curve is $2.675 + 1.78n$. Plotting the fit for the expected maximum length in Fig. 6, $\ln(2.675 + 1.78n)/n$, we could not distinguish it from the calculated one in all but the microscopic scale.

One might be tempted to discount random placement algorithm based on the above result. However, we show next that random placement under the unconstrained regime studied here is really not all that bad by examining what portion of the client population is within a "good distance" from its closest mirror given random placement. In the optimal placement, a client is at most $1/2n$ away from its closest mirror. In the case of random placement, we are interested in computing the portion of clients that are farther away from their closest mirror by more than a factor of t from optimal, i.e., by more than $t/2n$. This is done by looking at the probability that, for a random point, no mirror is placed at a segment of length t/n around it (a one dimensional ball of radius $t/2n$), which is given by

$$\Pr\{\text{distance} > t/2n\} = (1 - t/n)^n. \tag{2}$$

As n grows, we can write

$$\lim_{n\to\infty} \Pr\{\text{distance} > t/2n\} = \lim_{n\to\infty} (1 - t/n)^n = e^{-t}. \tag{3}$$

Thus, as the number of mirrors grows, a fixed portion of the clients are away by a certain stretch from optimal. Specifically, $1/e$ of the clients are at distance farther than the worst case of the optimal distance. Fig. 9 shows the result of an experiment we conducted to test the above analysis. As one can see, the probability converges to e^{-t} for n values well below 100 (the limit values are plotted in Fig. 9 as small symbols at $n = 900$).

The above analysis shows that, under optimal placement, the reduction in client–mirror distance has diminishing return with a well-defined "knee" as the number of mirrors increases. When clients are uniformly distributed, the optimal placement can achieve good load balancing while directing clients to the closest mirrors. The analysis also shows that even though the optimal placement increasingly outperforms random placement

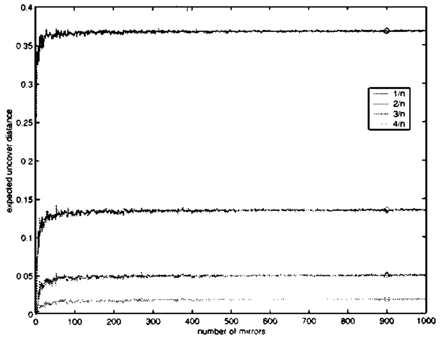


Fig. 9. Probability a client under random placement is farther than a stretch t of the distance bound in the optimal placement.

in terms of expected maximum client–mirror distance as the number of mirrors increases, the worst case maximum distances occur very rarely under random placement. We will refer to these results in interpreting our empirical data in Section V-A.

IV. Performance Evaluation

Our goal in conducting performance evaluation is to study the effect of changing $|\mathcal{M}|$ and $\mathcal{P}(\mathcal{M})$ on the optimization condition $\mathcal{OC}(\mathcal{M}, p)$. For our performance evaluation, we conduct both simulations on random topologies and experiments using Internet AS topology. For each set of experiments, we vary either $|\mathcal{M}|$ or $\mathcal{P}(\mathcal{M})$ while holding all the other variables constant. We now describe our simulation setup and scenarios, followed by a description of our Internet experiment setup.

A. Simulation Setup

The most accurate form of simulation for CMP would be on the host level, where individual servers and clients are represented as nodes in a network. There are two main difficulties with this approach: the computational overhead of large network and the representation of host-level topology capturing properties of the Internet. We are not aware of any current topology generator that generates a network of a reasonable (thousands of nodes) size with Internet like characteristics. Instead of simulating at the host level, we decide to simulate at the AS level, which has several advantages. Instead of simulating hundreds of thousands of hosts, we could represent the AS-level topology with just a few thousands nodes, thus, cutting down the computation cost and make the simulation feasible. Second, since an AS is under a single administration, a caching solution can be employed within the AS to avoid redundant client requests. In the best case, only a single cache server needs to make request on behalf of all clients inside the AS [3], effectively merging all the real clients into a single caching client. There are some issues that should be examined with using the AS-level topologies. Many ASs on the Internet are diverse in geographic location and span different continents, while others are small and confined to a single geographic location. From mirrors' standpoint, there are more potential clients inside large ASs, so representing a large AS as a single node masks many potential clients. However, we have already stated that having a cache proxy would essentially reduce redundant client requests within an AS into a single unique request from the cache proxy, effectively making each AS look like a single client node. From clients' standpoint, mirrors servers inside large ASs are preferable to mirrors inside smaller ASs due to better connectivity and bandwidth provisioning inside large ASs. By modeling each AS as a node actually makes mirrors in different ASs appear equally good, therefore, offering more available mirrors to choose from for each client than in reality. In other words, clients on the Internet are likely to select the closest mirrors from fewer candidates sites than in our simulations. Overall, using the AS-level topology model is a good abstraction for client requests while representing a more optimistic mirroring setup than in the Internet.

The AS topologies used in our simulations are generated using the Inet topology generator [18]. For this study, we generate several random topologies with 3037 nodes each.[7] Each generated network is a connected graph on a plane, with each node representing an AS; a link between two nodes represents AS connectivity, and its Euclidean distance the latency between the two connected nodes.

In each simulation, we first select 50 nodes to act as candidate hosts. We experiment with two candidate host selection methods. We simulate brute-force candidate placement with uniform random selection, whereby each node has an equal probability of being selected. We also simulate a more intelligent placement on nodes with largest outdegrees, which are generally perceived to be close to the core of the Internet. After the candidate hosts are selected, we randomly, with uniform probability, select 1000 of the remaining nodes to act as clients. For each mirror placement algorithm, we compute $\mathcal{OC}(\mathcal{M}, 1)$, $\mathcal{OC}(\mathcal{M}, 0.95)$ and $\mathcal{OC}(\mathcal{M}, \mu)$. We compute each $\mathcal{OC}(\mathcal{M}, p)$ for $|\mathcal{M}|$ ranging from 2 to 50. Client redirection is done by either redirection to the mirror with the smallest latency or randomly. We present the simulation results in Section V.

B. Internet Experiments

In addition to studying CMP on generated topologies, we also evaluate it with trace-based experiments on the Internet. In particular, we study the effect of optimizing the number and placement of mirrors on client download time when CMP is applied to clients extracted from several Web server logs. We conducted two experiments, the first one in December 1999 and the second in November 2000. The first experiment is based on Bell Labs Web server log, and the second experiment is based on Web server logs from nonprofit organizations Amnesty International and the Apache Software Foundation, and commercial businesses Sun Microsystems and Marimba. The use of these sites give our experiments a variety of client bases.

1) Candidate Host Set: We do not have access to 50 machines distributed across the Internet that can act as candidate hosts. Given our optimization condition of minimizing the latency experienced by clients, we observe that for the purpose of performance evaluation, CMP can be emulated on the Internet as long as we can determine the RTTs between $|\mathcal{H}|$ sites on the Internet and our client sets. We use multiple

[7] This was the size of the Internet in November 1997; our Internet experiments from December 1999 and November 2000 (not included in this paper) indicate that observations made in this paper also apply to larger networks.

TABLE I
LOCATIONS OF THE 89 TRACEROUTE GATEWAYS (MTGs)

Location	Number of MTGs	Percentage
North America	58	65.2
Western Europe	15	16.8
Rest of Europe	6	6.7
Australia	6	6.7
Israel	1	1.1
Korea	1	1.1
Mexico	1	1.1
S. Africa	1	1.1

TABLE II
HOST STATISTICS FROM INTERNET EXPERIMENT

First Experiment				
Web Log	Time Period	Unique IPs	Live IPs	Measured IPs, $\lvert\mathcal{B}\rvert$
Bell Labs	11/7-11/14/98	10,115	4,980	3,130

Second Experiment				
Web Log	Time Period	Unique IPs	Reachable IPs	Measured IPs, $\lvert\mathcal{B}\rvert$
Apache	10/1-11/30/97	274,843	51,227	25,635
Sun	10/1-10/31/97	219,528	39,304	19,914
Marimba	12/1-12/31/97	26,566	4,078	2,173
Amnesty	10/1-11/30/97	5,436	1,268	681

traceroute gateways (MTGs) [19] to serve as our candidate host sites. Traceroute gateways are Web servers made available to the public for measurement purposes by volunteers around the world. Given a host name or address, a traceroute gateway runs `traceroute` to that host and reports the result back to the user. In our experiments, we only need the RTTs between MTGs and clients, so using the MTGs is sufficient.

Table I lists the geographical locations of the 89 traceroute gateways used in the first experiment. The table reflects a reasonable diversity of the geographic locations of the traceroute gateways. Unfortunately, by the time we conducted our second experiment, only 50 of the 89 traceroute gateways were still operational.

In our experiments, we apply the *Transit* heuristic by placing mirrors on candidate hosts in descending order of outdegrees. Since we do not know the outdegrees of the traceroute gateways, we associate with each traceroute gateway the outdegree of the AS in which it is located. We first map the IP address of a traceroute gateway to its AS using a tool called `prtraceroute`, which is part of the routing arbiter project toolkit (www.irrd.net). Then, to determine the ASs outdegree, we use the AS summary information available at National Laboratory for Applied Network Research (NLANR) (moat.nlanr.net/AS/), which lists the outdegree of each AS. If the destination traceroute gateway's AS has a single connection to the rest of the Internet, we assign it the outdegree of its closest upstream AS with outdegree larger than one. The motivation here is to differentiate singly connected ASs that have well-connected parents and those that have poorly connected parents. Our intuition is that CMP can perform better by selecting well-connected degree-one ASs to host mirrors than fringe ASs that have higher degrees. On the other hand, the performance may be worse if the latencies between the singly connected ASs and their well-connected parents are large. From examining the data, we find that the latter case is not likely. However, we are not able to quantify the probability of it occuring given the limited granularity of the data.

2) Client Set: The client sets in our experiments are the IP addresses extracted from Web server logs of Bell Labs, Amnesty International, Apache Software Foundation, Sun Microsystems, and Marimba Inc. All of the logs except the Amnesty International log store the client addresses as dotted decimal IPs. The Amnesty International log stores the clients' fully qualified domain names (FQDN) instead. Since we determine unique clients by comparing IP addresses, it is necessary to perform DNS lookup on the Amnesty International clients to resolve their IP addresses. Only 2% of the domain name system (DNS) lookups failed. We present the number of unique clients extracted from all of the logs in Table II.

Due to the dynamic nature of the Internet, some IP addresses in the log file may no longer exist. Furthermore, dial-up connections with short lifetimes also prevent clients from being reached by MTGs *post facto*. Unreachable clients cause `traceroute` to wait until a timeout occurs, which could take up to 450 s (5 s/probe × 3 probes/hop × 30 hops). Not only do unreachable IPs greatly lengthen the experiment, but they also place extra strain on the MTGs with prolonged connections. Since we do not have control over the MTGs and cannot change their `traceroute`'s timeout behavior, we attempt to send only "live" IPs to the MTGs.

In our first experiment with the Bell Labs clients, we use a "TCP probe" to test for "live" IPs. With the "TCP probe," we open a TCP connection to port 80 (HTTP) of each IP address, and look for either a TCP SYN-ACK or RST reply from the IP. Either of these replies indicate the IP is "live," regardless of whether anything is listening on port 80 to accept the connection. If we receive no reply to the connection attempt, then the host is not reachable by the probe. We do this probe from two different sites (one in Michigan and the other in California), and eliminate IPs that are not reachable by at least one of the two probes. Even after this check for live IPs, only 63% of the IPs which passed our test were able to be reached by the MTGs.

For our second experiment, we take a different approach to determining "live" IPs. In the year between experiments, many networks have become much more suspicious of unsolicited traffic. To minimize the likelihood that our measurements are misconceived as malicious, we add another layer of filtering to our list of IPs. We use a standard "ICMP echo request" as the first check on each IP. Many security conscious networks block these requests, and we choose to interpret a failed request as a sign that they are unwilling to be measured. To each host, we send three packets spaced five seconds apart, with a five second timeout on each packet. If any of the probes successfully completes, we consider the host measurable. Otherwise, we remove the host from our list of unique IPs. We then send these IPs to MTGs to `traceroute`; despite being reachable to our ICMP probes, only about 50% of these IPs are reachable by the MTGs. To prevent these unreachable IPs from holding up the MTGs, we install an additional filtering step. Right before sending an IP to an MTG, we first sent another ICMP echo packet to the

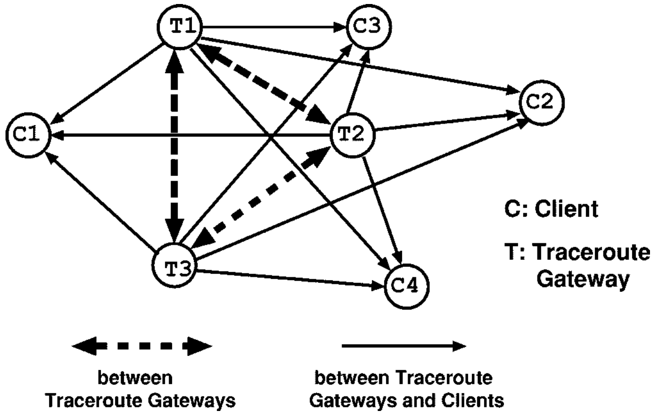

T1
C3
C2
C1
T2
T3
C4
C: Client
T: Traceroute Gateway
between Traceroute Gateways
between Traceroute Gateways and Clients

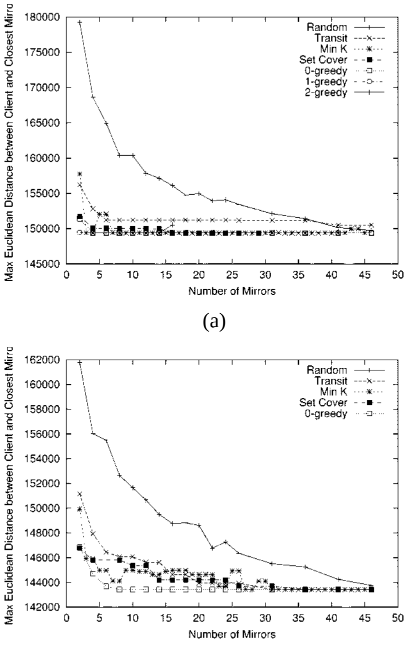
Fig. 10. Experiment setup.

host to test its reachability from our local machine. If the host is reachable, we then forward the IP to the MTGs. If an IP is not reachable, it is placed on a retry list to be tried again later; if a retried IP is still not reachable, it is permanently removed. The statistics on hosts that MTGs were able to `traceroute` are presented in Table II. We consider an IP address MTG-reachable when a `traceroute` from *any* MTG is successful.

The virtual network on which we conduct our CMP experiments, thus, consists of 89 (later 50) traceroute gateways as our candidate hosts, and the IP addresses found in the Web server logs as our clients. The "edges" of these virtual network consist of RTT measurements from each traceroute gateway to all of the other traceroute gateways and to all of the clients. For illustrative purposes, Fig. 10 shows a sample virtual network consisting of three traceroute gateways and four clients. The traceroute gateways measure RTTs between each other and RTTs to the four clients. The RTT measurements between traceroute gateways are bidirectional, while those between traceroute gateways and clients are unidirectional, as indicated in the figure.

V. Experiment Results

Recall from Section IV-A that in all of our simulations we use a network of 3037 nodes, of which $|\mathcal{H}| = 50$ are selected as candidate hosts. The choice of which host is selected to be a candidate host is determined either randomly with uniform probability for all nodes, or by the node outdegree. The client set consists of 1000 nodes randomly selected, with uniform probability, from the remaining nodes. Recall also that we define three optimization conditions: $\mathcal{OC}(\mathcal{M}, 1)$, $\mathcal{OC}(\mathcal{M}, 0.95)$, and $\mathcal{OC}(\mathcal{M}, \mu)$. For each optimization condition, we run a set of simulations. In each set of simulation, we first pick $|\mathcal{M}|$, the number of mirrors. For the given number of mirrors, we run one simulation for each mirror placement algorithm $\mathcal{P}(\mathcal{M})$: $\min K$-center, 0-greedy, 1-greedy, 2-greedy, cost-adjustable set cover, and Transit. Since random placement of mirrors gives different results based on the sites selected, for random placement, we run ten simulations for a given mirror set size and compute the mean of the observed $\mathcal{OC}(\mathcal{M}, p)$. Then, we repeat all simulations for the next $|\mathcal{M}|$. In our simulations, we experiment with $|\mathcal{M}|$ ranging from 2 to 50, stepping by 2 up to 26, and stepping by 5 afterwards. We then repeat each set of simulations on ten different Inet generated networks of 3037 nodes each. We do the above on 50 randomly selected candidate hosts first. We then repeat everything on 50 candidate hosts selected based on decreasing outdegrees, except that we do not simulate the 1-greedy and 2-greedy algorithms as they do not show marked improvement over the 0-greedy case in the former scenarios. Hence in total we ran 7350 simulations on randomly selected candidate hosts, and 6630 simulations on candidate selection based on outdegree.



(a)

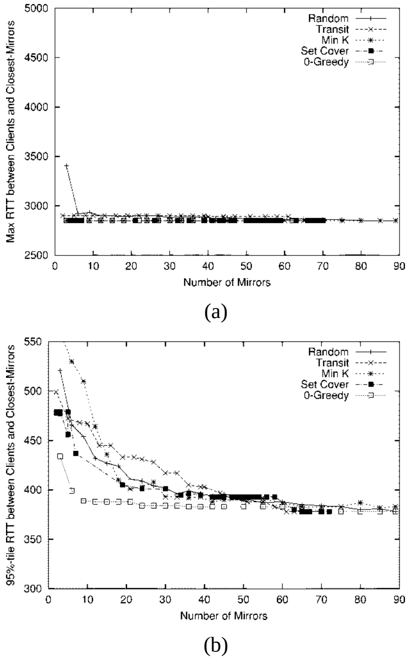
(b)

Fig. 11. Minimizing maximum RTTs between clients and closest mirrors. (a) Random candidate placements. (b) Outdegree-based candidate placements.

For the first Internet-based experiment, we repeat the above scenario with the 89 traceroute gateways acting as candidate hosts. Mirror set sizes range from 3 to 89, stepping by 3 up to 45, and stepping by 5 afterwards. Since there is only one virtual network, we do not repeat the set of simulations ten times; we do, however, still repeat the experiment ten times for each mirror set size when the mirror placement algorithm used is random placement. This means we run 1014 experiments on the virtual network. In the second experiment, the mirror set sizes range from 2 to 50 with an increment of 2. Again, we perform ten experiments for random mirror placement as in the first experiment. Both sets of experiments show qualitatively similar results, hence, for ease of exposition, we discuss only the results of the first experiment in the rest of this paper. Results from the second experiment are presented in the Appendix.

A. Optimization Condition $\mathcal{OC}(\mathcal{M}, p)$

We first consider the optimization condition $\mathcal{OC}(\mathcal{M}, p)$. Figs. 11(a), (b), and 12(a) show the maximum values of the client–mirror RTTs for $\mathcal{OC}(\mathcal{M}, 1)$. The x axis of each figure lists the number of mirrors, and the y axis the maximum value of the RTTs between clients and their closest mirrors. The x axes for the simulation results range from 0 to 50, while those for Internet experiments range from 0 to 90. The y axis in the various figures have different ranges. In the simulation results, the "distance" between two nodes is the Euclidean



(a)

(b)

Fig. 12. Internet experiments based on Bell Labs' clients. (a) Minimizing maximum RTTs between clients and closest mirrors. (b) Minimizing 95 percentile RTTs between clients and closest mirrors.

distance between them on the simulated plane. In the Internet experiments, distance is in milliseconds. The numbers for all placement algorithms, except for random placement, are averaged over simulations on ten random topologies to obtain the mean, the maximum, and minimum. For clarity, we only show the mean values in the figures, but we note that the maximum and minimum values are typically within 20% of the mean. Recall that for each of the ten random topologies, we simulate random placement of n mirrors ten times. From these ten random placements, we obtain the mean client–mirror RTTs. For random placement, the figures show the means of these mean values over the ten random topologies.

From these figures, we observe that optimizing $\mathcal{OC}(\mathcal{M}, 1)$ yields very little improvement as the number of mirrors increases, both in simulations and actual Internet experiments. The results from our remaining Internet experiments are presented in the Appendix and are consistent with the observations here. Under constrained mirror placement, the distance between clients and mirrors cannot be improved incrementally much beyond the first ten mirrors because mirrors cannot be placed progressively closer to the clients. Both candidate site placement and mirror placement can contribute to this problem. First, the optimal mirror placement is very "location-sensitive" in that it has very specific requirements on where the candidate sites should be, i.e., separated by an equal distance. Also, the optimal solutions for different mirror value n have very little overlap so it is impossible for all $O(n^2)$ optimal locations (at {1/2}, {1/3, 2/3}, {1/4, 1/2, 3/4}, … in the case of the mirror placement on the [0, 1] line segment studied in Section III-C) to be occupied if only n candidate sites are selected. Second, adding more mirrors cannot improve the minimum distance between a client and its closest candidate site (therefore the client's closest mirror) further, once the candidate site is selected for mirror placement. This problem can be exacerbated when the number of candidate sites is small relative to the client population.

Recall that solution to the $\min K$-center problem is applicable only in the case of optimization condition $\mathcal{OC}(\mathcal{M}, 1)$. Hence, for optimization conditions $\mathcal{OC}(\mathcal{M}, \mu)$ and $\mathcal{OC}(\mathcal{M}, 0.95)$, we consider only the ℓ-greedy, in particular 0-greedy, and the cost-adjustable set-cover algorithms. Fig. 12(b) shows the Internet experiment result for all placement algorithms when $\mathcal{OC}(\mathcal{M}, 0.95)$ is used. We observe that 0-greedy outperforms the cost-adjustable set cover even though both can optimize for $\mathcal{OC}(\mathcal{M}, 0.95)$. Our explanation is that the two greedy algorithms are actually quite similar except that cost-adjustable set cover optimizes for the average of the $\mathcal{OC}(\mathcal{M}, p)$ by dividing the $\mathcal{OC}(\mathcal{M}, p)$ of a client set by the cardinality of the set. The division of the actual optimization condition causes cost-adjustable set cover to optimize for a different objective even though it is qualitatively consistent with the optimization condition, $\mathcal{OC}(\mathcal{M}, p)$. Hence, we will only consider 0-greedy algorithm for optimization conditions, $\mathcal{OC}(\mathcal{M}, 0.95)$ and $\mathcal{OC}(\mathcal{M}, \mu)$.

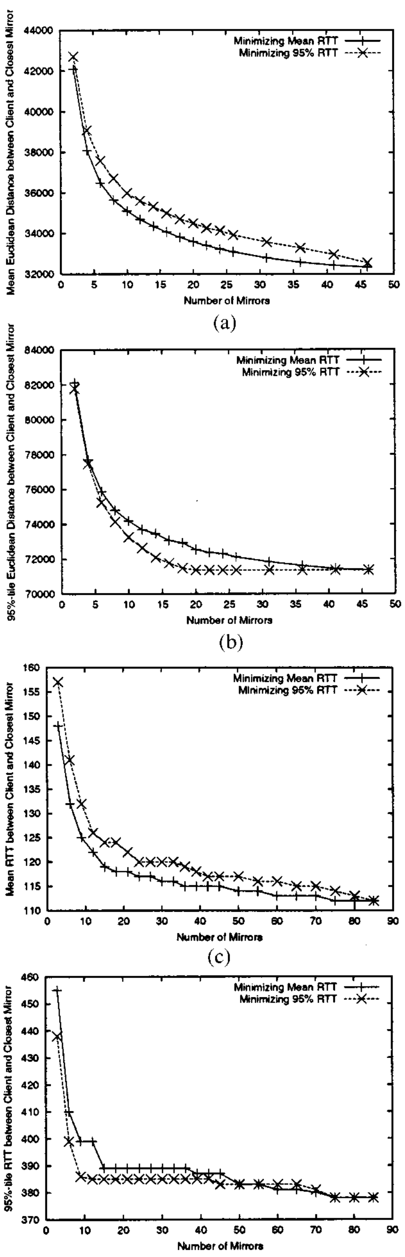
Fig. 13 shows the mean and 95 percentile of client–mirror distances when candidate sites are selected based on outdegrees, and mirror placement is by the 0-greedy algorithm. Both the 95 percentile and mean client–mirror graphs show diminishing return and a well-defined "knee," which confirms the theoretical analysis and our intuition. We observe very similar performance between the two curves, reflecting $\mathcal{OC}(\mathcal{M}, 0.95)$ and $\mathcal{OC}(\mathcal{M}, \mu)$ optimization conditions, and attribute this to the potentially long, but nonetheless not heavy tail of the client–mirror RTT distribution in our setups (which means that the 95 percentile is not that far from the mean). In the remainder of this paper, we use $\mathcal{OC}(\mathcal{M}, 0.95)$ as our optimization condition.

B. Effect of $|\mathcal{M}|$ and $\mathcal{P}(\mathcal{M})$ on $\mathcal{OC}(\mathcal{M}, p)$

Figs. 12(b) as well as 14(a), and (b) show the observed 95 percentile RTTs between clients and their closest mirrors when $\mathcal{OC}(\mathcal{M}, 0.95)$ is used. Note that in most cases, especially when the 0-greedy algorithm for mirror placement is used, there is little improvement in 95 percentile RTT beyond ten mirrors.

One important observation with regard to $\mathcal{P}(\mathcal{M})$ is that placement is very important when the number of mirrors is small. In all cases, when $|\mathcal{M}|$ is small, there is a significant difference in observed latency between using the greedy placement algorithm and random placement. When $\mathcal{P}(\mathcal{H})$ is uniform, nonrandom $\mathcal{P}(\mathcal{M})$ outperforms random placement. Even when $\mathcal{P}(\mathcal{H})$ is nonrandom, as in the case of outdegree-based candidate selection, using greedy placement improves $\mathcal{OC}(\mathcal{M}, 0.95)$ by 10% to 20% as shown in Fig. 14 (note the difference in y-axis ranges).

We conclude that increasing the number of mirrors beyond a small portion of the candidate sites (ten, in our examples) does not necessarily improve client to closest mirror latency. Furthermore, careful placement of mirrors on a small number of candidate sites can provide the same performance gain as placing mirrors on all candidate sites. These results suggest that candidate site placement can be just as important and possibly more important than mirror placement itself. We note that, in practice,



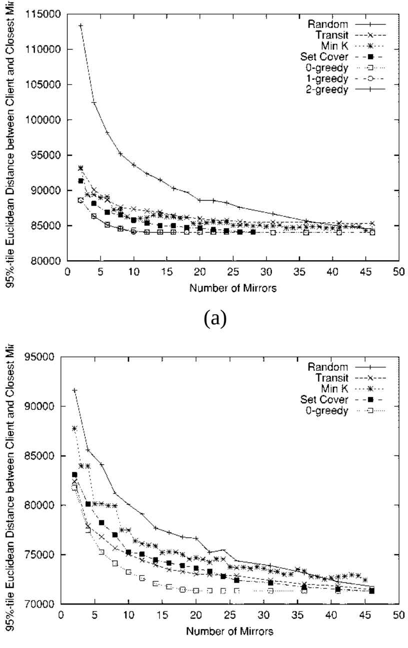
(a)

(b)

(c)

(d)

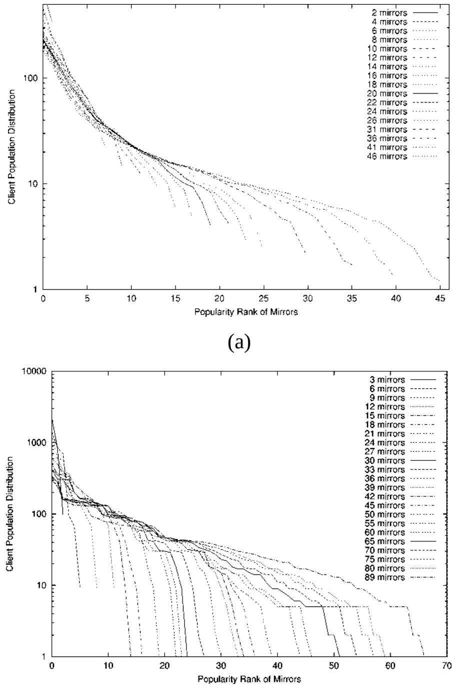
Fig. 13. Mean and 95 percentile RTTs. (a) Mean RTTs with outdegree-based candidate placement. (b) 95 percentile RTTs with outdegree-based candidate placement. (c) Mean RTTs on Bell Labs experiments. (d) 95 percentile RTTs on Bell Labs experiments.



(a)

(b)

Fig. 14. Minimizing the 95 percentile RTTs between clients and mirrors. (a) Random candidate placements. (b) Outdegree-based candidate placements.



(a)

(b)

Fig. 15. Client population distribution under 95 percentile RTT optimization. (a) Simulation results. (b) Bell Labs experiment.

candidate sites are often determined by administrative and financial constraints rather than technical ones.

C. Mirror Load Distribution

We now show that using $\mathcal{OC}(\mathcal{M},\ 0.95)$ as the optimization condition, mirror load distribution is not improved much even with larger numbers of mirrors. Fig. 15 plots client distribution among mirrors when the number of mirrors is increased from 2 to 50 (3 to 89, in the Internet experiment). The x axis is the popularity rank of each mirror, and the y axis is the number of clients redirected to a particular mirror, with the most popular one getting the most redirections. Each curve in the graphs represents a specific mirror set size. For the simulations, the candidate sets are chosen based on decreasing outdegrees. In all cases, the optimization condition is $\mathcal{OC}(\mathcal{M},\ 0.95)$, and the mirror placement algorithm is 0-greedy. In the simulation, only a small number of clients (less than 1% of mirrors) get redistributed with each additional mirror once the number of mirrors is above 15. Client redistribution is also infrequent in our Internet experiments.

Again, we point to our analysis in Section III-C, where we showed that the optimal placement produces good load

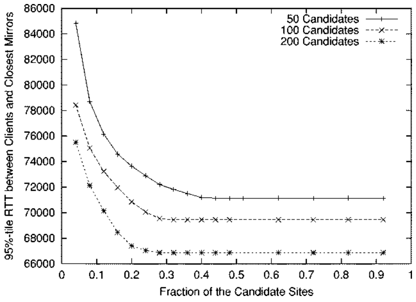


Fig. 16. 95 percentile RTT optimization with different numbers of candidates in simulation.

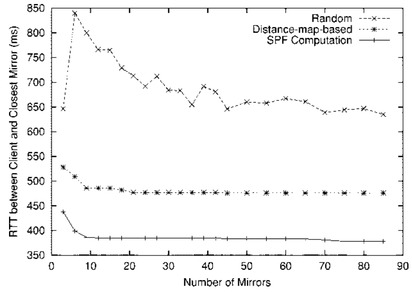


Fig. 17. 95 percentile RTT optimization under different redirection schemes in Bell Labs experiment.

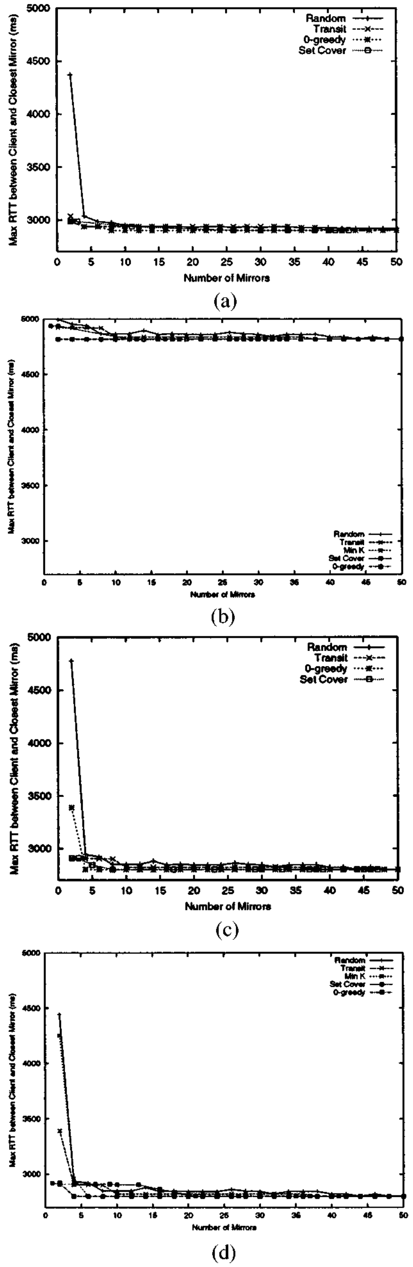
balancing among mirrors as the number of mirrors increases. We have already shown that it is difficult to reproduce the ideal setting when mirror placement is constrained so perhaps it is not surprising that the ability to load-balance is also lost.

D. Effect of $|\mathcal{H}|$ on $\mathcal{OC}(\mathcal{M}, p)$

Thus far in simulation, we have shown that given 50 candidate sites, we observe rapid diminishing return as more mirrors are placed on candidate sites. It is important to validate this result when more candidate sites are available. We repeat the simulations outlined in Section IV-A except now instead of selecting only 50 candidate sites, we select 100 and 200 candidates sites based on outdegree. We perform the 95 percentile optimization on RTTs between clients and closest mirrors using the 0-greedy algorithm. Fig. 16 shows the results of having 100 and 200 candidate sites along with the case of 50 candidate sites. The x axis is the percentage of the candidate sites with mirrors placed, and the y axis is the 95 percentile RTT between clients and closest servers. Again, we average our results over the ten Inet topologies as stated before. We make two observations: as expected, having more candidates improves performance—the 95 percentile RTT can be further improved with additional candidate sites; the rapid diminishing return is observed under all three scenarios. We believe our conclusion that a careful placement algorithm is required to place mirrors on a small fraction of candidate sites, holds for larger numbers of candidate sites.

E. Effect of Redirection Methods

Up to now, we have assumed that client–mirror distances can be directly measured. In this section, we consider the case where only ten of the highest outdegree traceroute gateways are able to perform `traceroute`. In this situation, distances among the other traceroute gateways and from a traceroute gateway, other than these ten, to a client must be estimated by doing triangulation on the distances measured by these ten traceroute gateways only. This simulates the case where the underlying network topology is not known (such is the case with the Internet), and a "distance map" of the underlying topology must be estimated by placing measurement boxes on the network. It was shown by simulations in [7] that when the underlying network topology is not known, nearest mirror redirection using some form of distance map outperforms random redirection. We now show that similar results can also be observed on the Internet. Fig. 17 shows the 95 percentile of client–mirror RTTs under $\mathcal{OC}(\mathcal{M}, 0.95)$ when distances are known, with random redirection, and with redirection using a distance map. The results were obtained from Internet-based experiments, when mirrors are placed using the 0-greedy algorithm.



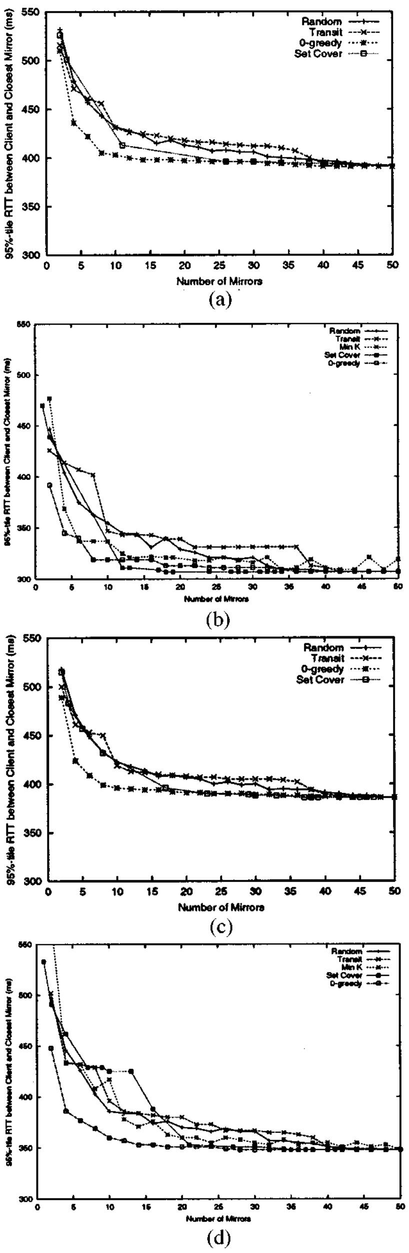
(a)

(b)

(c)

(d)

Fig. 18. Minimizing maximum RTTs between clients and closest mirrors. (a) Apache. (b) Amnesty International. (c) Sun. (d) Marimba.



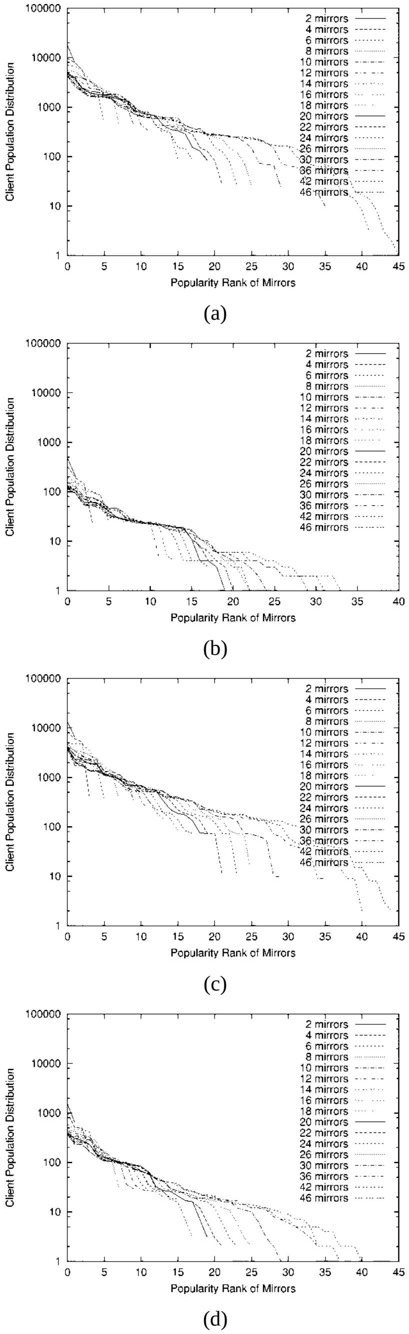
(a)

(b)

(c)

(d)

Fig. 19. Minimizing 95 percentile RTTs between clients and closest mirrors. (a) Apache. (b) Amnesty International. (c) Sun. (d) Marimba.



(a)

(b)

(c)

(d)

Fig. 20. Client population distribution under 95 percentile RTT optimization. (a) Apache. (b) Amnesty International. (c) Sun. (d) Marimba.

VI. Conclusion

In this paper, we take a detailed look at the problem of placing mirrors of Internet content on a restricted set of hosts. We introduce a formal model to study the constrained mirror placement. Using both simulation and real Internet delay data, we examine a number of placement and redirection algorithms for placing various numbers of mirrors and their effects on client response time and mirror load distribution. We determine that the ℓ-greedy algorithm gives the best performance in CMP. We observe that there is a rapidly diminishing return to placing more mirrors in terms of both client latency and server load balancing. We hypothesize that the presence of the locality constraint has eliminated some of the necessary conditions for obtaining the optimal solution and the subsequent performance benefits. Even under the more elaborate placement schemes, simply increasing the number of mirrors yields very little performance improvement beyond that of a relative small number of mirrors.

Appendix

We present in this appendix results from our second Internet experiment described in Section V. Fig. 18 corresponds to Fig. 12(a), Fig. 19 corresponds to Fig. 12(b), and Fig. 20 corresponds to Fig. 15. Please see our discussions of the corresponding figures on how to read and interpret these figures.

ACKNOWLEDGMENT

The authors would like to thank the volunteers who donated their time and resources to host the `traceroute` gateway service and make it available to the public (http://www.tracert.com/cgi-bin/trace.pl). They thank L. Zhang for discussions on the optimization condition, S. Khuller for suggesting that we consider set-cover in the placement problem, and B. Krishnamurthy, Amnesty International, Apache Software Foundation, Marimba Inc., and Sun Microsystems for making the Web server logs available to them.

REFERENCES

[1] C. Gray and D. Cheriton, “Leases: An efficient fault-tolerant mechanism for distributed file cache consistency,” in *Proc. 12th ACM Symp. Operating Systems Principles*, 1989, pp. 202–210.

[2] H. Yu, L. Breslau, and S. Shenker, “A scalable web cache consistency architecture,” in *Proc. ACM SIGCOMM*, Sept. 1999, pp. 163–1674.

[3] B. Krishnamurthy and J. Wang, “On network-aware clustering of web clients,” in *Proc. ACM SIGCOMM 2000*, Aug. 2000, pp. 97–110.

[4] A. Myers, P. Dinda, and H. Zhang, “Performance characteristics of mirror servers on the internet,” in *Proc. IEEE INFOCOM*, Mar. 1999, pp. 304–312.

[5] Z. Fei, S. Bhattacharjee, E. Zegura, and M. Ammar, “A novel server selection technique for improving the response time of a replicated service,” in *Proc. IEEE INFOCOM*, 1998, pp. 783–791.

[6] S. Seshan, M. Stemm, and R. Katz, “Shared passive network performance discovery,” in *Proc. 1st Usenix Symp. Internet Technologies and Systems (USITS ’97)*, Dec. 1997.

[7] S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, and L. Zhang, “On the placement of internet instrumentation,” in *Proc. IEEE INFOCOM*, Mar. 2000, pp. 295–304.

[8] Y. Bartal, “Probabilistic approximation of metric space and its algorithmic applications,” in *Proc. 37th Annual IEEE Symp. Foundations of Computer Science*, Oct. 1996, pp. 184–193.

[9] V. Vazirani, *Approximation Methods*. New York: Springer-Verlag, 2001.

[10] S. Jamin, C. Jin, A. Kurc, D. Raz, and Y. Shavitt, “Constrained mirror placement on the internet,” in *Proc. IEEE INFOCOM*, Apr. 2001, pp. 31–40.

[11] L. Qiu, V. Padmanabhan, and G. Voelker, “On the placement of web server replics,” in *Proc. IEEE INFOCOM*, Apr. 2001, pp. 1587–1596.

[12] V. Cardellini, M. Colajanni, and P. Yu, “Dynamic load balancing on web server systems,” *IEEE Internet Comput.*, pp. 28–39, May–June 1999.

[13] S. Floyd and V. Jacobson, “Random early detection gateways for congestion avoidance,” *ACM/IEEE Trans. Networking*, vol. 1, pp. 397–413, Aug. 1993.

[14] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, “Modeling TCP throughput: A simple model and its empiracl validation,” in *Proc. ACM SIGCOMM*, Sept. 1998, pp. 314–303.

[15] M. Garey and D. Johnson, *Computers and Intractability*. New York: Freeman, 1979.

[16] P. Krishnan, D. Raz, and Y. Shavitt, “The cache location problem,” *ACM/IEEE Trans. Networking*, vol. 8, pp. 568–582, Oct. 2000.

[17] H. David, *Order Statistics*, 2nd ed. New York: Wiley, 1981.

[18] C. Jin, Q. Chen, and S. Jamin, “Inet: Internet topology generator,” Univ. Michigan, Tech. Rep. CSE-TR-433-00, 2000.

[19] Multiple Traceroute Gateways. (1999) TraceRT consulting service. [Online]. Available: http://www.tracert.com

Eric Cronin received the B.S.E. degree in computer engineering, in 2000 and is working toward the Ph.D. degree in the Department of Electrical Engineering and Computer Science (EECS), University of Michigan, Ann Arbor.

His current research interests include network security, Internet topology, and networked games.

Sugih Jamin received the Ph.D. degree in computer science from the University of Southern California, Los Angeles, in 1996.

He is an Assistant Professor in the Department of Electrical Engineering and Computer Science (EECS) at the University of Michigan, Ann Arbor. From 1992 to 1993, he was at Xerox PARC, Palto Alto, CA.

Dr. Jamin is a recipient of the ACM SIGCOMM Best Student Paper Award in 1995, the National Science Foundation Presidential Early Career Award for Scientists and Engineers (PECASE) in 1999, and the Alfred P. Sloan Research Fellowship in 2001. He is currently an Editor for the IEEE/ACM TRANSACTIONS ON NETWORKING.

Cheng Jin received the B.Sc. degree in electrical engineering from Case Western Reserve University, Cleveland, OH, in 1996. He is working toward the Ph.D. dergree in the Department of Electrical Engineering and Computer Science (EECS), University of Michigan, Ann Arbor.

His current area of research includes placement of measurement servers on the Internet. He is a codeveloper of the Inet topology generator.

Anthony R. Kurc received the B.S.E. degree in computer engineering in 2000 and is working toward the Ph.D. degree in the Department of Electrical Engineering and Computer Science (EECS), University of Michigan, Ann Arbor.

His current research interests are building scalable Internet infrastructures and network measurement.

Danny Raz (M’99) received the Ph.D. degree from the Department of Applied Mathematics and Computer Science, Feinberg Graduate School, Weizmann Institute of Science, Rehovot, Israel, in 1995.

He is a faculty member at the Department of Computer Science, Technion–Israel Institute of Technology, Haifa. He was a Postdoctoral Fellow at the International Computer Science Institute, Berkeley, from 1995 to 1997. He was also a Visiting Lecturer at the University of California, Berkeley, CA, during 1996–1997. He has been a Member of the Technical Staff at Bell Labs, Holmdel, NJ, since 1997.

Yuval Shavitt (S’88–M’97–SM’00) received the B.Sc. (*cum laude*), M.Sc., and D.Sc. degrees from the Technion–Israel Institute of Technology, Haifa, in 1986, 1992, and 1996, respectively.

After graduation, he spent a year as a Postdoctoral Fellow in the Computer Science Department at Johns Hopkins University, Baltimore, MD. Since 1997, he has been a Member of the Technical Staff at the Networking Research Lab at Bell Labs, Holmdel, NJ. Since October 2000, he has been a faculty member in the Department of Electrical Engineering at Tel-Aviv University, Tel-Avi, Israel.

Dr. Shavitt served as TPC member for INFOCOM 2000, 2001, and 2002, IWQoS 2001, and ICNP 2001.