Coverage-aware proxy placement for dynamic content management over the Internet

Srivatsan Varadarajan, Raja Harinath, Jaideep Srivastava and Zhi-Li Zhang
Department of Computer Science, University of Minnesota
200 Union street, EE/CS Building, room 4-192, Minneapolis, MN 55455
{varadara,harinath,srivasta,zhzhang}@cs.umn.edu

Abstract— **In an effort to differentiate service quality, service providers have resorted to employing Content Distribution Networks (CDNs) over the Internet. CDNs deploy geographically distributed *proxy servers* which manage content on behalf of the service provider's servers for better performance and enhanced availability.**

In this paper we explore the *proxy placement problem for content distribution* over the Internet. Its goal is to strategically place a number of proxies in the network to optimize certain criteria which improve performance of proxies. We motivate the various necessary factors and constraints that need to be taken into account for a *good* placement of proxies over the Internet which reflect real world scenario more accurately and which we claim hitherto has not been completely addressed. We introduce a novel concept of *host coverage* characterizing every *Autonomous Systems (AS)* and use this stable, coarse grained measure as a long-term estimate of the load being serviced by the proxy system. We then pose an optimal formulation of the proxy placement problem taking into consideration all the relevant factors. We propose a couple of proxy placement algorithms that solve the above problem and analyze their behavior. Finally we present the performance of those algorithms against the optimal solution and other schemes proposed in literature. We also study the stability of the proposed algorithms through a variety of experiments.

Keyword: **Proxy Placement, Coverage, Internet, Content Distribution Network (CDN)**

I. Introduction

There has been an explosive growth in the services offered by different enterprises and commercial organizations over the Internet in the past few years. This has led to the emergence of a wide variety of applications with differing quality of service (QoS) needs. In an effort to differentiate their service quality, service providers have resorted to employing the services of Content Distribution Networks (CDNs) over the Internet. CDNs are geographically distributed *proxy servers* which manage content on behalf of the service provider's servers for better performance and enhanced availability. These proxies serve as *proactive client caches* reducing service response time by being closer to the client and balancing the server workload. There has been an increasing trend for content distribution to be out-sourced to distribution networks like Akamai, on behalf of traditional server hosting companies. The task of such a service is to redirect a user request in a manner that improves service time for different clients. If done well, this can lead to cost reduction, by requiring fewer proxies, and improved client response time through better coverage - both of which are beneficial from the service provider's viewpoint. In this paper we study the *proxy placement problem for content distribution*, whose goal is to strategically place a number of proxies in the network to optimize certain criteria. Broadly speaking, an effective and realistic proxy placement should consider (i) constraints imposed by Internet routing policies (ii) the ability of proxies to handle dynamically changing content and (iii) the demand or load to be serviced by the proxies.

We handle the placement of proxies at the level of *Autonomous Systems (AS)*, just as in [1], [2]. The reasoning being that though there are many ASs on the Internet, they are far less than the actual number of hosts and hence makes the problem more tractable. But *physical connectivity* between ASs does not necessarily imply *logical connectivity*: adjacent ASs do not always route traffic between each other. This is due to constraints imposed by routing policies in BGP, an inter-domain routing protocol on the Internet. Most work on proxy placement [1], [3]–[6] neglect this constraint on connectivity. The work in [2] is a notable exception to this. Though in that work they consider content on all the proxies as identical, which is easier to analyze but undesirable, since it increases storage overheads and limits the use of smart *content/cache servicing* schemes based on the current *request load* once the placement is done. It was observed in [5] that while typical tree based algorithms using mechanisms of hierarchical placement performed better than random placement, they do not work as well as algorithms for general graphs. Thus, in this formulation we characterize the cost of communication structure amongst proxies, but we do not restrict them to predefined specific topologies or structures (as in [1]). In the context of web servers, [5] observed that placement algorithms which incorporate client distance and request rate in their decisions perform 2-5 times better than workload-oblivious algorithms. A typical approach to capture demand is to use some request trace data e.g. web logs. We believe that this information is ephemeral and too detailed to be used as a basis for a long-term placement problem. The challenge then is to be able to get a longer-term load measure in the Internet, wherein workload information is always likely to be imperfect. We introduce a novel concept of *host coverage* which is a coarse indicator of load associated with an AS and provide techniques to infer it. For experimental validation of the coverage metric through an Internet study, intuition behind the proposed proxy placement model and more experimental results, readers are directed to the longer version of this paper in [7].

This paper is organized as follows: Section II introduces Autonomous Systems, and AS graph construction. It also explain how to infer host coverage information from the AS graph. Section III formulates the Optimal Proxy Placement problem which takes into consideration all the various constraints and factors listed before and their tradeoffs. Section IV presents a couple of heuristics to solve this problem, and analyzes them. Section V describes simulations evaluating the heuristics, comparing them with the optimal solution and other simple simple algorithms, and their sensitivity to small perturbations of input data.

Proceedings of the 23 rd International Conference on Distributed Computing Systems Workshops (ICDCSW'03)
0-7695-1921-0/03 $17.00 © 2003 IEEE

II. Inferring AS-related Data

In this section we provide mechanisms to extract the data required for the proxy placement problem formulation. We show how the AS graph is inferred and distance between every pair of ASs are computed (§II-A). We then address the issue of inferring host coverage (§II-B).

A. AS Graph Inference and Distance matrix computation

The Internet is a collection of ASs each having different authority and routing policies depending on the organization they belong to. An AS typically has detailed information about its internal topology and limited connectivity information about other ASs. BGP is a *distance vector* protocol that constructs paths by successfully propagating route announcement/withdrawal updates between adjacent BGP speaking routers (internally maintained in a *BGP routing table*), through *peering sessions*. The update announcement or acceptance between neighboring ASs are constrained by *selective export or import policies*, depending on different commercial contractual business agreements amongst the Internet service Providers (ISP) managing the corresponding ASs [8], [9].

Lot of work has been done on inferring AS-level Internet topology from BGP routing table information and through other mechanisms like intelligent probing etc. ([10], [11]). In there they list out mechanisms in which the *AS graph*, which is a logical map of the Internet, is constructed. For the purpose of this work we assume the Internet AS topology information is known *a priori* and is collected through one of the these mechanisms.

Physical connectivity between AS domains do not imply reachability – if Node A has neighbor B, which in turn has neighbor C, then it is not necessarily true that A can reach C through B. For a proxy placement, one needs to contend with these policies and the relationships between ASs [9]. As a consequence of this the AS graph collected before is a *directed graphs*. To handle this problem, the authors in [9] infer such AS relationships and then use these relationships to construct the logical Internet graph in [12]. Finally in [2], they use a *Modified Dijkstra algorithm* to find the minimum AS hop count distance from every AS to a proxy placed in a particular AS A i.e *shortest distance* from every AS to a proxy in AS A. We can then extend this algorithm in a straight forward manner to run n (= total number of ASs) times or use a *similarly modified (all pair shortest path) Warshall-Floyd algorithm* to construct the complete distance matrix. This distance matrix is *not symmetric* in nature as also *does not satisfy the triangle inequality* because, any such correlations are effectively broken by different BGP related policies. In effect *distance matrix is non-metric*. The effect of BGP policies is, however, limited to the distance matrix computation (every entry in it correspond to the shortest distance between the corresponding ASs) with no other implications from the point of view of this paper.

B. Host Coverage Inference

It has been shown that placement algorithms which incorporate workload/demand information in their decisions outperform those that are oblivious to such information [5]. In the context of proxy web servers or DNS servers, the typical approach taken in estimating client demand is to cluster different ASs that are topologically close together using BGP information. Then by analyzing web server logs or traces, extracting the client IPs and associating them with the corresponding cluster, aggregated demands of the clusters (of ASs) are made ([1], [13]). While this approach has its merits, from the point of view of a content service provider (who wants to establish such a proxy service), there are some issues and problems to contend with: 1) A new service provider does not have access to client request traces [2]. 2) The client request traces so logged is very specific information relating to client interests and could be a transient, short-term phenomenon useful for caching/request routing schemes but not very relevant for a permanent proxy placement scheme.

To address these issues we take the following approach. We define a more *long term coarse load measure* called *host coverage*, which is indicative of a potential load request pattern. We determine host coverage based on the *IP Address Space distribution* amongst the different ASs. Formally we define the load α_i of AS i as the fraction of IP Address Space originating from an AS with respect to overall routable IP space.

$$\alpha_i = \frac{\text{IP address space originating from } AS_i}{\text{Overall rout-able IP Space}}$$

Algorithm 1 *computeHostCoverage*

Input: BGP routing table entries $R(i) = (pref, len, num)$, where $pref$ is the prefix, len is the mask length and and num is the originating AS: the last AS number in the AS list

Output: $\alpha_j, \forall 1 \leq j \leq n$ where n is the number of ASs in R

Define: $R(i) \preceq R(j)$ iff $R(i).pref$ is a prefix of $R(j).pref$ or $R(i).pref = R(j).pref$ and $R(i).len \leq R(j).len$.

1. Construct the *Prefix Tree* T such that each node of the tree contains an entry of R:
   - 1.1 Create a dummy root with entry $(0.0.0.0/128.0.0.0, 1, 0)$
   - 1.2 Construct T so that for every subtree T' rooted at $R(i)$, every child $R(j) \in T'$ satisfies $R(i) \preceq R(j)$,
2. Process the tree T in post-order:
   - 2.1 For a leaf node with entry $R(i)$, set $\alpha_{R(i).num} = \alpha_{R(i).num} + 2^{32-R(i).len}$
   - 2.2 For an interior node with entry $R(i)$, set $\alpha_{R(i).num} = \alpha_{R(i).num} + 2^{32-R(i).len} - \sum_{\forall j, R(j) \text{is child of } R(i)} \alpha_{R(j).num}$
   - 2.3 For the root of T, set $\alpha_0 = 2^{32} - \sum_{\forall j, R(j) \text{is child of root}} \alpha_{R(j).num}$
3. Set $\forall 1 \leq i \leq n, \alpha_i = \frac{\alpha_i}{2^{32} - \alpha_0}$

Algorithm 1 *computeHostCoverage* illustrates the method through which α_i can be computed. Each node routes to a given IP address using the BGP entry which has the longest prefix match, and for all practical purpose that address can be associated with the originating AS of that BGP entry. Since IP addresses are not assigned in a hierarchical manner, any originating AS an be found for multiple prefixes in the BGP table. We construct the prefix tree T wherein every node is a specific (*prefix*, *mask length*, *originating AS*) entry from the BGP table. We add a dummy root which spans the full 2^{32} IP address space. We process the tree bottom up so that we do not double count any specific IP address to be associated with an AS. For any node in T and its associated AS, the IP addresses specifically originating from it are the complete space it spans (based on the mask length), excluding all the IP spaces originating from its children. Trivially the dummy route node (with a dummy AS 0) finally holds the complete set of IP addresses which cannot be routed using the given BGP table information, termed *black holes* [14] plus the unused/unassigned IP address spaces. Finally we normalize each $\alpha_i, \forall 1 \leq i \leq n$ with respect to the total routable IP address space, where n is the total number of ASs.

Proceedings of the 23 rd International Conference on Distributed Computing Systems Workshops (ICDCSW'03)
0-7695-1921-0/03 $17.00 © 2003 IEEE

easily modify this algorithm to exclude reserved IP address subspaces.

In reality, a single IP address may represent whole other intranets hidden behind it using techniques like NAT (Network Address Translator) [15]. Some IP addresses might be used only temporarily or not assigned. Also we do not discriminate between routers and end hosts in an AS. Since we are looking at only a *coarse measure* of the load, we believe using the IP address space in this manner is still reasonable. Further we design heuristics (in section IV) such that the actual placement is *insensitive* to perturbations of the load.

III. Optimal Proxy Placement

Setting up a proxy involves two components: *proxy placement*, and *proxy service region distribution*. Proxy placement deals with placing the different proxies in different ASs. Service region distribution determines set of ASes that is served by a given proxy. *The optimization output variables are* x, y represent the two components listed here. In this formulation we not only account for costs arising because of services provided directly from proxies to different ASs, but also costs between different proxies as this would have implications for content cache management once proxies are set up. We first state the problem informally.

Vble.	Definition
$\mathcal{N}$	Set of ASs, $\mathcal{N} = \{1, 2, \ldots, n\}$
$\mathcal{P}$	Set of placed Proxies, $\mathcal{P} \subseteq \mathcal{N}, \lvert\mathcal{P}\rvert = m$
$\mathcal{D}$	$(d_{i,j})$, $1 \leq i, j \leq n$; $d_{i,j}$ = distance from AS i to j
α	Normalized *host coverage* Vector, 1 x n matrix, $\alpha = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}, \sum_{i=1}^{n} \alpha_i = 1$
x	n x m matrix, $x = (x_{i,j})$, $i \in \mathcal{N}$, $j \in \mathcal{P}$, where $x_{i,j} = 1$ if proxy j is placed in AS i, else 0. Note that $\mathcal{P} = \{i, \exists j$ such that $x_{i,j} = 1\}$
y	n x m matrix,$y = (y_{i,j})$, $i \in \mathcal{N}$, $j \in \mathcal{P}$, where $y_{i,j} = 1$ if AS i is serviced by proxy j, else 0.
$W_{\mathcal{N},\mathcal{P}}$	Work done for requests originating from ASs to corresponding proxies
$W_{\mathcal{P},\mathcal{P}}$	Work done amongst proxies to service different requests
λ	$0 \leq \lambda \leq 1$: probability of requests from ASs directly serviced by the corr. proxies w/o any redirections.
$W_{n,m}$	Overall work done in n ASs using m proxies

TABLE I
Variables and Definitions

Optimal Proxy Placement Formulation (OPPF):

Objective Reduce the overall work done by the set of m proxies to service requests from all the different n ASs.

Considerations

- Load as a result of hosts covered by different ASs
- Work done between a proxy and an AS for every request originating from the AS
- Work done amongst the different proxies. This is the cost for content management.

Constraints

placement: At most one proxy placed in any AS
service: Each AS serviced by exactly one proxy

Output Place at most m proxies in n potential AS sites and assign every AS to be serviced by one of the proxies.

Assumption All hosts are equally likely to originate requests and ASs are only as important as their host coverage.

The *Optimal Proxy Placement Formulation (OPPF)* is a mixed-integer programming problem with a non-linear objective function. The variables used and their definitions are provided in table I.

Minimize

$$W_{n,m} = W_{\mathcal{N},\mathcal{P}} + (1 - \lambda).W_{\mathcal{P},\mathcal{P}} \tag{1}$$

where

$$W_{\mathcal{N},\mathcal{P}} = \sum_{i \in \mathcal{N}} \alpha_i \sum_{j \in \mathcal{P}} y_{i,j} \sum_{k \in \mathcal{N}} x_{k,j}.d_{k,i}$$

$$W_{\mathcal{P},\mathcal{P}} = \frac{1}{|\mathcal{P}|^2}. \sum_{a \in \mathcal{P}} \sum_{b \in \mathcal{N}} x_{b,a} \sum_{c \in \mathcal{P}} \sum_{d \in \mathcal{P}} x_{d,c}.d_{b,d} \tag{2}$$

Subject to

$$x_{i,j},\ y_{i,j} \ \in \ \{0,1\},\ \forall i \in \mathcal{N},\ \forall j \in \mathcal{P} \tag{3}$$

$$\sum_{i \in \mathcal{N}} x_{i,j} \ \leq \ 1,\ \forall j \in \mathcal{P} \tag{4}$$

$$\sum_{j \in \mathcal{P}} x_{i,j} \ \leq \ 1,\ \forall i \in \mathcal{N} \tag{5}$$

$$\sum_{j \in \mathcal{P}} y_{i,j} \ = \ 1,\ \forall i \in \mathcal{N} \tag{6}$$

The objective is to minimize *the expected work done by the proxy setup* in serving a request. The two components are the work done by proxies servicing the requests from different ASs $W_{\mathcal{N},\mathcal{P}}$ and work done amongst the different proxies $W_{\mathcal{P},\mathcal{P}}$. $W_{\mathcal{N},\mathcal{P}}$ indicates the *average* distance traveled by requests from the assigned proxies to different ASs, normalized by the host coverage α. $W_{\mathcal{P},\mathcal{P}}$ indicates the average work done amongst other proxies to be able to service content, when proxies in charge of different ASs cannot directly service the request. We represent this work as the *average distance between all pair of proxies*. Thus distance between $Proxy_a$ and $Proxy_c$ placed in AS_b and AS_d respectively is given by $x_{b,a}.x_{d,c}.d_{b,d}$. λ, an input parameter to the formulation, indicates the overall probability of content being serviced by proxies directly without any redirections amongst proxies. $\lambda \to 1$ implies highly *available* proxies that have almost all the content and for all practical purpose can be considered as independent *server replicas*. We do not have consider costs amongst the different proxies in that case. The work $W_{\mathcal{N},\mathcal{P}}$ is always done, but with probability $(1-\lambda)$ there is an additional work done of $W_{\mathcal{P},\mathcal{P}}$. An important point is that the objective function assigns *weights* to different ASs proportional to the *host coverage* α_i. If the proxy service provider has additional information so that certain ASs are more important than others, then the objective function could be easily changed for suitable other weights; e.g., if only a subset of ASs are to be considered for proxy placement, the α_i's can be *renormalized* for those ASs. Another way to chose these weights is to use geographical information of ASs, to determine the focus of the service and the likely demand for the service.

Constraint 5 is the *placement constraint* while constraint 6 is the *service constraint*. Notice that in this formulation, *multiple ASs can be serviced by a single proxy* and the *optimization could result in less than* m *proxies being placed if adding more proxies does not help in decreasing the overall cost*.

This problem is essentially a variant of the *K-Median Problem* and has been shown to be *NP-hard* [16]. The metric versions of this problem, wherein the costs are non-negative, symmetric and satisfies the triangle inequality, has been studied in detail and a slew of constant-factor (with reference to optimal values) polynomial time approximations have been proposed. It is important to note that the distance metric in our case is *non-metric*. Thus the proxy placement problem, which we have posed here, is a *non-metric uncapacitated K*

problem and is NP Hard. To the best of our knowledge, we do not know of existing graph theoretic solutions that are directly applicable to this problem.

IV. Heuristics

In this section we propose two heuristics: *greedy exchange* (§IV-A) and *greedy coverage* (§IV-B). We then analyze the properties of the heuristics (§IV-C).

A. Greedy Exchange

Algorithm 2 *greedy-exchangeCompute*

Input : $\mathcal{D}$, α, $\mathcal{N}$ with$|\mathcal{N}| = n$, m, λ, μ
Output : $\mathcal{P}$ with $|\mathcal{P}| \leq m$.

0. $\mathcal{P} \leftarrow \phi$, $i \leftarrow 1$
1. Compute C_1 using the first proxy in AS j for which C_i^j is minimum, $\forall j \in \mathcal{N}$.
   Set $\mathcal{P} \leftarrow \mathcal{P} \cup j$ and the x, y appropriately.
2. while $i < m$ do
   - 2.1 Compute C_{i+1} using equation (8)
   - 2.2 $B_{i+1} \leftarrow C_{i+1} - C_i$
   - 2.3 if $B_{i+1} \leq 0$ then break out of loop: *no benefit adding any more.*
   - 2.4 Pick the cost effective AS j which gave the minimum in the above computed C_{i+1}
   - 2.5 $\mathcal{P} \leftarrow \mathcal{P} \cup j$
   - 2.6 Update x and y variables appropriately.
   - 2.7 $i \leftarrow |\mathcal{P}|$
3. Let $\mathcal{P} = \{p_1, p_2, \cdots, p_{|\mathcal{P}|-\mu}, p_{|\mathcal{P}|-\mu+1}, \cdots, p_{|\mathcal{P}|}\}$ be the order in which the ASs were added into the proxy list in steps 1 and 2.
4. Considering the last μ ASs added to the proxy list
   - for each $p \in \{p_{|\mathcal{P}|-\mu}, p_{|\mathcal{P}|-\mu+1}, \cdots, p_{|\mathcal{P}|}\} \subseteq \mathcal{P}$
     - for each $e \in \mathcal{N} - \mathcal{P}$
       - if swapping p and e, reduces overall cost, then swap them: $\mathcal{P} = \mathcal{P} - \{p\} \cup \{e\}$.

The greedy algorithm starts by placing one proxy, and then iteratively places one additional proxy at a time until all m proxies have been placed. Let the current set of proxies already placed be $\mathcal{P}$ with $|\mathcal{P}| = i$ at the beginning of an iteration (i.e i proxies have been placed). Define the cost of iteration $i+1$ as

$$C_{i+1}^j = W_{\mathcal{N},\mathcal{P}\cup\{j\}} + (1-\lambda).W_{\mathcal{P}\cup\{|\},\mathcal{P}\cup\{|\}} \quad (7)$$

$$C_{i+1} = min\{C_{i+1}^j, \forall j \in \mathcal{N} - \mathcal{P}\} \quad (8)$$

where W is as specified in equation 2 of section III, except that it uses the distance matrix $\mathcal{D}$ instead. At each iteration, we compute the x, y variables needed for cost computation as follows:

- For the x, when AS j is added to to $\mathcal{P}$ in i^{th} iteration, set $x_{j,i} = 1$.
- For the y variable, we assign it so that $\forall a \in \mathcal{N}$ find the *closest proxy* $k \in \mathcal{P}$ with $x_{l,k} = 1$ (placed in AS l), such that $d_{l,a}$ is minimum.

Now C_{i+1}^j is the cost of the placement of choosing j as the new proxy and C_{i+1} searches this over all unchosen proxy i.e $\forall j \in \mathcal{N} - \mathcal{P}$ and picks the cheapest one.This cost represents the cheapest cost of placing one more proxy. Let also define *benefit* of placing this additional proxy as $B_{i+1} = C_{i+1} - C_i$.

The greedy exchange process is listed in algorithm 2. The steps 1, 2 indicates the greedy process, while step 3, 4 is the exchange process. The greedy process is simply to keep adding proxies one at a time until we reach m or there is no more benefit in adding one. Then is the exchange process, we look at the last μ ASs added to the proxy list, where $0 \leq \mu < |\mathcal{P}|$. If by exchanging one of those ASs with other ASs which are not selected, we get a lower cost, we then exchange. Higher values for μ increases the running time of the algorithm, but gives better placement. Typically small, constant values of μ, say 2 or 3 or 4, gives close to optimal value for most of the experiments. Since we are dealing with an offline algorithm where running time is not very critical we do not show the results of those experiment in this paper.

B. Greedy Coverage

Algorithm 3 *greedy-coverageCompute*

Input : $\mathcal{D}$, α, $\mathcal{N}$ with$|\mathcal{N}| = n$, m, λ
Output : $\mathcal{P}$ with $|\mathcal{P}| \leq m$.

0. $zone \leftarrow \phi$, $i \leftarrow 1$, $done \leftarrow false$
1. while $i \leq m$ and $done = false$ do
   - 1.1 $zone_i \leftarrow \phi$, $load_i \leftarrow 0$
   - 1.2 while $load_i \leq \frac{1}{m}$ and $done = false$ do
     - 1.2.1 find an AS $j \in \mathcal{N} - zone$ such that it is the *minimum distance* tosome AS $k \in zone_i$
       If $\mathcal{N} - zone = \phi$ set $done = true$, then break out of the loop.
       If $zone_i = \phi$, pick any AS $j \in \mathcal{N} - zone$
     - 1.2.2 $zone_i \leftarrow zone_i \cup j$
     - 1.2.3 $zone \leftarrow zone \cup j$
     - 1.2.4 $load_i \leftarrow load_i + \alpha_j$
     - 1.2.5 $numProxies \leftarrow i$
   - 1.3 $i \leftarrow i + 1$
2. while $i \leq numProxies$ do
   - 2.1 do step 1 of algorithm 2 using the complete AS set as $zone_i$ so as to find (by setting x variable appropriately) a proxy among $zone_i$ ASs with cheapest cost and y variable so that all $j \in zone_i$ are serviced by that proxy.

In the previous *greedy exchange* approach, we consider both the dimensions of host coverage and distances (between AS and proxy as well as amongst proxies) *simultaneously* (Algorithm 2). In the *greedy coverage* approach, we deal with the host coverage dimension first and then deal with the distances. We first *partition* the n ASs into m zones in such a manner so as to *equalize* the total host coverage of each zone. In each zone, the way we pick ASs is similar to any spanning tree algorithm. Next we place a proxy in each of the m zones so that the cost is minimized in each of the zone. The cost is computed as before, except that we consider each proxy to be closest to ASs in its own zone. This process is listed in algorithm 3. Note that $\sum_{i=1}^{n} \alpha_i = 1$. We call this approach *greedy coverage*.

C. Analysis of Heuristics

The greedy-exchange heuristic is iterative. Proxy service providers can add proxies iteratively based on the growth in their demand. Of course the *greedy-exchange heuristics is not guaranteed to be optimal* — we **cannot say** that for $\mathcal{P} \subseteq \mathcal{N}$ being the proxy set computed through the greedy-exchange process and *any* $\mathcal{P}_1 \subseteq \mathcal{N}$ such that $|\mathcal{P}_1| = |\mathcal{P}|$,

$$W_{\mathcal{N},\mathcal{P}} + (1-\lambda).W_{\mathcal{P},\mathcal{P}} \leq W_{\mathcal{N},\mathcal{P}_1} + (1-\lambda).W_{\mathcal{P}_1,\mathcal{P}_1}$$

What we hope to do is by controlling the parameters (with small values) of Greedy Exchange (μ), we get close to the optimal. There is a nice property of the greedy-exchange heuristic, summarized in lemma 4.1. Basically the greedy exchange heuristic exhibits a property of *local optimum cost value AS as a place for a proxy in the current iteration than from other unselected ASs for large* μ. Obviously, large μ increases the time complexity of the algorithm.

Lemma 4.1: If $\mathcal{P} \subseteq \mathcal{N}, |\mathcal{P}| = m$ is the set of proxies computed by greedy-exchange algorithm 2 and for any other set $\mathcal{P}_1 \subseteq \mathcal{N}, |\mathcal{P}_1| = m$ such that $|\mathcal{P} \cap \mathcal{P}_1| = \mu = m - 1$, then

$$W_{\mathcal{N},\mathcal{P}} + (1-\lambda).W_{\mathcal{P},\mathcal{P}} \leq W_{\mathcal{N},\mathcal{P}_1} + (1-\lambda).W_{\mathcal{P}_1,\mathcal{P}_1}$$

Proof: The proof is straight forward. Since $\mu = m - 1$, then step 4 of algorithm 2, simply loops through all $p \in \mathcal{P}$ and $e \in \mathcal{N} - \mathcal{P}$ and swaps if there is an unselected proxy which will result in a *lower* cost. Thus for any other proxy set $\mathcal{P}_1$ which has $m-1$ common elements with $\mathcal{P}$, it can never have that *additional* proxy e found in it and not in $\mathcal{P}$ which will result in a lower cost (else it would have been exchanged). ■

An important point to note is that by Lemma 4.1, when $m = 1$ i.e., only one proxy to be placed, the greedy-exchange heuristic is optimal. By following the algorithm 3, it is also clear that the greedy coverage is also optimal when $m = 1$.

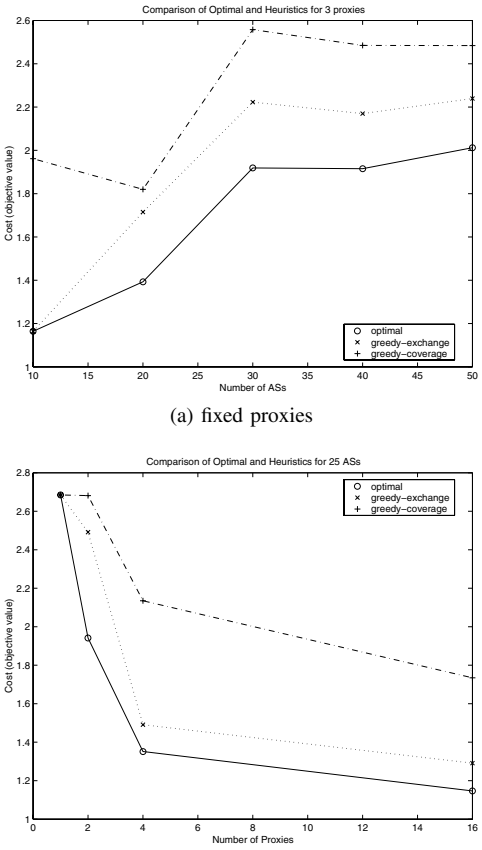
V. Experimental Evaluation

In this section we evaluate our proposed heuristic algorithms: *greedy exchange* and *greedy coverage*. All the experiments were run on a SGI Origin 3800 using a single 500 MHz R14000 processor and 1 GB memory. All the code has been written in Matlab Version 6.0. In the discussion below, the term *costs* refers to the overall work done by the proxies.

A. Optimal vs Heuristics comparison

In this set of simulations, we compare the *optimal* algorithm with *greedy-exchange* and *greedy-coverage*. We conducted two batch of experiments: *Experiment 1*: Effect on cost for a fixed number of proxies placed in a increasing number of ASs (1(a)) and *Experiment 2*: Effect on cost for increasing number of proxies placed in a fixed number of ASs (1(b)). The optimal solution is computationally expensive and impractical to compute for large number of ASs. For example placing 5 proxies in a 500-node topology takes roughly 13 hours to solve. Also, for large number of ASs and/or proxies time is an issue, as is the convergence to the optimal solution. But, in an experimental setup, the optimum cost or *best (minimum)* cost serves as a baseline for comparing the heuristics against. So we generate *synthetic small topologies* having small number of ASs to analyze the behavior of greedy algorithms.

The distance matrix $\mathcal{D}$ was generated with each entry in the matrix having AS hop count values randomly generated values between 1 and 6, with the diagonal entries as 0 (distance of AS to itself). These numbers were chosen because they are representative of the top 100 ASs, as indicated in [5]. The α's were generated uniformly in the range $[0.0\text{-}1.0]$ and *renormalized* so that they sum up-to 1.0. For the *constrained non-linear optimization* we use the Sequential Quadratic Programming (SQP) method for solving it. Since variables are restricted to *discrete values*, we use a *branch and bound* technique on top of the SQP method. All experiments were run for a wide range of parameters and multiple runs. We show the results in figure 1 for $m = 3$, $\mu = 2$ and $n = 10, 20, 30, 40, 50$ in experiment 1 and for $n = 25$, $m = 1, 2, 4, 8, 16$ and $\mu = m - 1$ in experiment 2. As seen from the graphs, both greedy-exchange and greedy-coverage perform close to optimal and in-fact the graph behavior of these algorithms is similar to the optimal. As expected we see the cost increasing when we try to place 3 proxies in an increasing number of ASs (see figure 1(a)). Figure 1(b) shows exhibits some interesting patterns. a) As shown by lemma 4.1 in §IV-C both the greedy heuristics give the same results as optimum when $m = 1$. b) As expected



(a) fixed proxies

(b) fixed ASes

Fig. 1. Comparison of optimal and heuristics

for a fixed 25 number of ASs, when we keep increasing the number of proxies, the costs drop. But after certain number of proxies, the costs (including optimal) *plateau* and then start increasing as we increase the number of proxies. This is due to the fact that after some *threshold* number of proxies, the inter-proxy cost start dominating over the proxy-AS cost and it does not make sense to increase the number of proxies beyond that threshold. Thus, both the greedy heuristics proposed here have been constructed so as to add proxies in an iterative manner and stop when the inter-proxy costs start to dominate over the proxy-AS cost.

B. Comparison of proposed heuristics against other schemes

For the following experiments we chose BRITE [17] to generate the AS topology over other generators like GT-ITM and INET, due to its flexibility in changing a wide range of parameters, ability to interface with other tools and for its ability to support a lot of the models. We use the Waxman model and a *flat AS* representation for topology generation in BRITE. In this set of experiments we chose $\lambda = 0.70$ and we vary number of ASs $n = 500, 1000, 2000$ and for each we always place $m = 10$ proxies. The distance matrix $\mathcal{D}$ is computed for each of the them running *All pair shortest path Warshall* algorithm on the topology generated. For all degree 1 nodes α was generated in the range $[0.5\text{-}1.0]$, while all the remaining nodes with higher degrees have theirs in the range $[0.0\text{-}0.5]$. Results are shown in figure 2.

We compare *greedy-exchange* and *greedy-coverage* against *random* and *highest-degree-first* algorithm. Random algorithm is load oblivious. It's m proxies are chosen randomly and each AS is then assigned the closest proxy near it. Highes

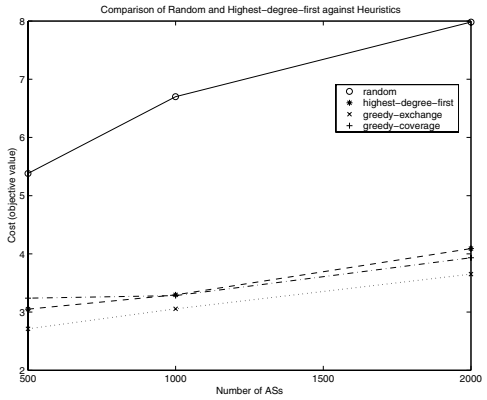


Fig. 2. Comparison of Heuristics

first algorithm chooses the proxies based on the degree of the different ASs. The AS with the highest degree is chosen first, the AS with the next highest degree is chosen next and so on. Again, once all the proxies are placed, each AS is serviced by the closest proxy. These algorithms have been considered as reference algorithms in [2], [5]. They again serve as baseline heuristics for comparison against and the results are shown in figure 2. It can be seen that greedy-exchange, greedy-coverage and highest-degree-first algorithms, all *outperform* random one, reasoning being that random algorithm does not take into account any form of load consideration. *greedy-exchange* performs the *best*. The performance of greedy-coverage and highest-degree-first are very *similar*. As mentioned in §II-B, an AS degree captures *weakly* the notion of host coverage. In effect, when greedy-coverage attempts to equally distribute the load among m zones first and then places a proxy in each zone, it *mimics* what an AS degree does. It has the same effect as balancing the overall sum hosts covered by an AS, when a proxy is placed there which is the notion captured by the degree of an AS. Also both greedy-coverage and highest-degree-first, solves the host coverage (load) dimension first and then the proxy to AS distances as a secondary problem. Also both do not solve for costs associated with inter proxy distances. greedy-exchange solves simultaneously all the costs involved in 1)host coverage 2)proxy-as distance and 3)proxy-proxy distances taking into consideration all the tradeoffs involved. Thus it outperforms all of them.

C. Sensitivity study through perturbations of input data

In the following experiments we try to analyze the behavior of the greedy algorithms, when the input data is *erroneous*. The idea is that we place the proxies with existing data using the heuristics and compute the cost for it. Then we perturb the input data and use this to compute the cost as "what should have been" with *real/actual data*. The error percentages are computed with this actual data as reference. We study the impact on the heuristics due to 1) *erroneous coverage* and 2) *erroneous topology* i.e. error in distance matrix. Due to space constraints, we do not show the performance graphs here but it can be found in [7]. Instead we just summarize the results.

As expected, perturbations of coverage data affects greedy-coverage more than greedy-exchange and perturbations of distance data affects greedy-exchange more than greedy-coverage. Interestingly, greedy-coverage (which mimics performance of highest-degree-first algorithm) has really close to 0% error for perturbations of distances. In both the cases (perturbations upto 10% of host coverage as well as upto 10% of the edges by 50% of the distance value) greedy-exchange and greedy-coverage algorithm do very well, with less than 1.4% error or deviation from actual. Hence these algorithms are quite stable with reference to small errors.

VI. Conclusions

In this paper we address the issue of placing proxies so as to enable content delivery over the Internet. We proposed a *novel* idea of *host coverage*, a coarse measure of load and provided techniques to compute it. Then we formulated the problem which takes into consideration all the factors that affect the proxy placement decisions, namely load, AS-proxy distances which affects the response time of servicing requests originating from an AS and also inter-proxy distances which enables a more resilient proxy service on failure and a more flexible setup to enable moving content between proxies. Next we proposed two heuristics *greedy-exchange* and *greedy-coverage* and their analysis. Finally from the various performance results of the experiments, it has been shown that the both the greedy-exchange and greedy-coverage algorithms are *insensitive* to *minor perturbations* of the input data and are hence quite *stable*. Further these algorithms model *close to optimal* behavior and in-fact greedy-exchange outperforms both random and highest-degree-first heuristic, while greedy-coverage performs much better than random but is similar to highest-degree-first.

References

[1] P. Barford, J.-Y. Cai, and J. Gast, "Cache Placement Methods Based on Client Demand Clustering." 16th IEEE Annual Computer Communications Workshop, October 14-17 2001.

[2] K. Kamath, H. Bassali, R. Hosamani, and L.Gao, "Policy-Aware Algorithms for Proxy Placement in the Internet," in *ITCOM*, 2001.

[3] M. R. Korupolu, C. G. Plaxton, and R. Rajaraman, "Placement algorithms for hierarchical cooperative caching," in *SODA: ACM-SIAM Symposium on Discrete Algorithms (A Conference on Theoretical and Experimental Analysis of Discrete Algorithms)*, 1999. [Online]. Available: citeseer.nj.nec.com/article/korupolu99placement.html

[4] S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, and L. Zhang, "On the placement of internet instrumentation," in *INFOCOM (1)*, 2000, pp. 295–304. [Online]. Available: citeseer.nj.nec.com/jamin00placement.html

[5] L. Qiu, V. Padmanaban, and G. M. Voelker, "On the Placement of Web Server Replicas." IEEE INFOCOMM, 2001.

[6] S. Jamin, C. Jin, A. R. Kurc, D. Raz, and Y. Shavitt, "Constrained mirror placement on the internet," in *INFOCOM*, 2001, pp. 31–40. [Online]. Available: citeseer.nj.nec.com/jamin01constrained.html

[7] S. Varadarajan, R. Harinath, J. Srivastava, and Z.-L. Zhang, "Towards Proxy Placement in Content Distribution Networks," Dept of Computer Science, University of Minnesota, Twin Cities, Tech. Rep. TR02-035, April 2002.

[8] A. Broido and kc claffy, "Internet Topology: connectivity of IP graphs: http://www.caida.org/outreach/papers/topologylocal/index.xml." Cooperative Association for Internet Data Analysis (CAIDA).

[9] L. Gao, "On inferring autonomous system relationships in the internet," in *IEEE Global Internet Symposium*, November, 2000. [Online]. Available: citeseer.nj.nec.com/gao00inferring.html

[10] H. Chang, S. Jamin, and W. Willinger, "Inferring AS-level Internet Topology from Router-Level Path Traces." Proc. of SPIE ITCom, August 2001, pp. 19–24.

[11] L. Subramanian, S. Agarwal, J. Rexford, and R. Katz, "Characterizing the internet hierarchy from multiple vantage points," 2001. [Online]. Available: citeseer.nj.nec.com/subramanian02characterizing.html

[12] Z. Ge, D. Figueiredo, S. Jaiwal, and L.Gao, "On the Hierarchical Structure of the Logical Internet Graph," in *ITCOM*, 2001.

[13] B. Krishnamurthy and J. Wang, "On network-aware clustering of web clients," in *SIGCOMM*, 2000, pp. 97–110. [Online]. Available: citeseer.nj.nec.com/302864.html

[14] S. Halabi and D. Mc.Pherson, *Interent Routing Architectures*, 2nd ed. Cisco Press, 2000.

[15] P. Srisuresh and K. Egevang, "Traditional IP Network Address Translator (Traditional NAT)," RFC 3022, http://www.isi.edu/in-notes/rfc3022.txt, January 2001.

[16] "A Compendium of NP optimization problems: http://www.nada.kth.se/~viggo/problemlist/compendium.html."

[17] "Boston University Representative Internet Topology Generator (BRITE): http://www.cs.bu.edu/brite/."