# ProxyTeller: A Proxy Placement Tool for Content Delivery under Performance Constraints

Peter Triantafillou
*Department of Computer Engineering and Informatics, University of Patras*
*peter@ceid.upatras.gr*

Ioannis Aekaterinidis
*Department of Electronic and Computer Engineering, Technical University of Crete*
*jaikater@softnet.tuc.gr*

## Abstract

*The efficient delivery of web content has been identified as a key issue of research for some time. Forward (or reverse) proxies, which are positioned along the request route from the users' browsers to the origin content servers, maintain a cache with copies of content from their origin servers. The strategic placement of proxies across the backbone ISP or the Content Delivery Network can drastically improve the performance of the system (in terms of network bandwidth savings, origin server load, and user-request latency).*

*The ultimate goal of this work is to develop a tool that decides on the position and the number of proxies required in order to achieve given performance improvements (expressed in terms of network bandwidth, origin server load, and user-latency). We believe such a tool will be very helpful to ISPs/CDNs, content providers, and end-users and is, thus, very much lacking.*

## 1. Introduction

Currently there are billions of data objects in the web. Typically, a fraction of these objects will be requested more frequently. A web proxy server is a computer system that stands between web origin servers and (the browsers of) a community of users/clients. It provides a cache that is shared among its client community, storing web objects, so that later requests from these clients for the cached objects can be served from the cache rather than from the remote origin server. Since some objects are requested more frequently, the web traffic is reduced considerably, the origin server is substantially off-loaded, and the user-observed latencies may be reduced [17].

Proxies are categorized into forward or reverse proxies (the latter are also called *surrogate servers*). The former can be found in ISP network backbone nodes, intercepting all web traffic. The latter are found in CDN servers being contacted only for content requests belonging to origin servers, which have specific business agreements with the CDN. Cached content in reverse proxies is under the explicit control of the content provider. Despite these differences, both proxy types perform fundamentally the same caching tasks. Thus, in both types of environments, appropriately placed proxies within a network can prove very beneficial for the systems' efficiency. Therefore, the problem of appropriately placing proxies within a given network constitutes a fundamental issue for the efficient delivery of content.

### 1.1 Motivation, goals, and related work

The central aim of this research is to ultimately develop a tool, coined *ProxyTeller*, for the efficient placement of proxies. The problem ProxyTeller solves accepts as inputs the network topology over which proxies will be placed, the desired sizes of their caches, the characteristics of the workload of the community served by a proxy (i.e., representative workload traces) and the performance improvement that is desired. The latter can involve one of the metrics of network bandwidth, user-request latency, and hit ratio and is specified in terms of the percentage improvement that is desired to be achieved, compared to the case where no proxies are deployed. The output of ProxyTeller is the position and the number of proxies required in order to achieve the stated performance goals.

It should be clear that to meet our challenge we have to be able to:

(a) define the performance improvement that is the result of introducing a proxy at a specific position within the network, and

(b) employ efficient proxy placement algorithms and measure their performance impact.

During the past several years a great wealth of knowledge

on efficient web proxy caching ([6],[22],[1],[15],[2],[19], [18]) has been accumulated. This knowledge is in terms of components, which replacement functions must include and on auxiliary algorithms and resources. Briefly, the accumulated knowledge suggests employing a cache replacement function that takes into account the objects' popularities, their sizes, and their remote-access costs, favoring small, remote, and popular objects. In addition, auxiliary caches and cache admission control can help further by eliminating unfortunate, eager replacements and maintaining key statistics (e.g., for estimating popularities). In order to be able to build powerful web proxies and understand their performance, one must be able to appreciate the true impact and significance of these contributions and how they can be integrated. We undertook a study for this task. The presentation of the details of the study and our findings are beyond the scope of this paper. We will overview the results, presenting a chart for the problem solution space identifying which algorithm performs better under different circumstances (i.e., cache sizes, access skew distributions, etc) and by how much. The findings of this study play a fundamental role within ProxyTeller, since on the one hand it 'advises' the tool with respect to the best algorithm to use and, on the other, it quantifies the performance improvement with respect to the performance metrics of interest.

Another key issue is that of the suitable placement of server replicas in nodal points of the network. The server replica placement problem can also be viewed as the *facility location problem (FLP)* [7], whose objective is to find both the number of 'facilities' and their location in the network, which results in the minimization of a cost function appropriately defined. Given that FLP is NP-Hard, a number of heuristic algorithms have been developed for the placement of full-server replicas in a graph topology [16] or for the placement of proxies in tree topologies [14]. The related problem of content placement, (i.e., placing object replicas across a network) has also been studied recently [21],[12], examining issues such as the minimization of Autonomous System hops [21] for satisfying user requests, or taking into account bandwidth constraints in WAN settings [12].

We concentrate on the efficient placement of forward or reverse proxy caches. Our approach is different than the ones in [21],[12]: Instead of studying content placement, we study the placement of proxy caches, with given sizes. The functionality of each proxy cache is independent; it can use its own replacement algorithm and mechanisms, utilizing the results of our first study, effectively harnessing the wealth of knowledge accumulated to decide which objects should be locally cached/replicated.

For obvious reasons, studying proxy (in essence, partial replicas) placement as opposed to full-server replica placement, is of vital importance for both ISPs/CDNs, content providers, and consumers, since the capital investment need for proxy caches is far smaller than that needed for full-server replicas. Using the results from our study on proxy-cache performance, we undertook a second study to define the performance impact of adapting the existing server-replica placement algorithms for proxy cache placement.

Combining the results from the above studies, we implemented a novel tool, ProxyTeller. We believe a tool such as ProxyTeller will be very useful, and in particular:

- to ISPs/CDNs, in order to optimally decide on required capital investments, weighing them against expected performance improvements,
- to content providers, which will be able to estimate their origin server off-load possible (as this is depicted in cache hit ratios) from a potential contract with a CDN and the expected performance improvements seen by the consumers of their content, and
- to end-users, being able to understand the performance of possible applications they build and/or run, accessing such content.

Thus, the significance of this work lies mainly in its ability to answer placement questions of high interest to all parties involved in a content delivery setting.

A demo of ProxyTeller is available through the internet [24]. Our aim is to make the source and executable code, as well as detailed results from the two constituent studies, freely available to the community.

## 2. The performance impact of proxies

As mentioned above, we first undertook a simulation-based performance study to quantify and qualify the performance effect of proxy deployment, which drastically depends on cache replacement algorithms. Precisely, we studied the impact of several key components and mechanisms of a cache replacement scheme. Details of the study setup, the results obtained, and their explicit explanation is definitely outside the scope of this paper. For the purposes of this work, we simply mention the key characteristics of this study, which are the following:

- The study used synthetic workloads, using tools such as SURGE [4]. We also used some traces for validation purposes. The generated workloads issue requests for objects following a Zipf distribution with parameter ? varying from 0.6 to 1, as has been found by related work [4],[5],[3].
- We quantified the performance improvement of well-known algorithms, such as LRU and the GDS algorithm [6] (we tested the GDS(1) which tries to maximize hit ratio and the GDS(lat) which tries to minimize overall latency).
- We modeled the 'conventional wisdom' contributed to

by several researchers by the employment and performance testing of the Cost Size Popularity (*CSP*) algorithm, which assigns a caching gain value to each object based on a mathematical expression involving the size, the popularity (which can be estimated with well-known techniques found in the literature [20]), and the communication cost to retrieve the object (objects with smaller caching gain are more likely candidates for eviction). We also studied CSP-related algorithms, which (de-) emphasized the importance of some terms, such as the CP-SQRT(S), the CP-LOG(S), and the CP algorithm that gradually degrade the influence of the object's size in the caching gain and the CS algorithm which does not take into account the popularity of objects.

- We measured the impact of utilizing mechanisms such as admission controllers in the cache and auxiliary caches for maintaining key statistics.
- In addition to the Zipf distribution parameter ?, we tested the algorithms' performance varying the size of the proxy cache.
- Finally, for modeling request-latency, from the different components that impact it [9],[10], we concentrated solely on data transfer latency from origin servers to proxies, (i.e., assuming TCP-connection caching and fast DNS resolution). We used an M/M/1-based analytical model for estimating latency as opposed to related research, which uses either trace-data or simple average-network times. We feel that the problem of estimating latency savings due to proxy deployment is an open problem. Despite its shortcomings, our model allows for the latency estimator to account for key requirements in the context of cache replacement, namely that (i) it be parametric and sensitive to the load on the internet, (ii) it reflect the cost differences when fetching objects of different sizes, and (iii) reflect the fact that different replacement algorithms have different impact on internet load, due to the different hit ratios they achieve. Our conclusions regarding latency are in concert with those for bandwidth savings, which were explicitly measured. However, we stress that the results which refer to latency can be replaced with any other results from other researchers, as better answers to the latency modeling problem emerge, or even downgraded (especially given the fact that most agree that the latency benefits are significantly inferior to those one would hope for [8],[13]).

The following are the performance metrics used in the study:

i. The hit ratio, defined as the percentage of the requests that hit the cache compared to all requests.
ii. The network bandwidth requirements (web traffic), which is defined to be the amount of data retrieved (in Kbytes) over the web during the duration of playing the trace file.
iii. The client response time (latency) measured in milliseconds. Since we are concentrating on proxy server design issues, it does not include the network latency incurred for the (internal) communication between the client's browser and the proxy.

## 2.1 Problem solution-space chart and discussion

**2.1.1 Main results concerning the replacement algorithms.** We have found that in general GDS(1) outperforms CSP and LRU in terms of hit ratio except in skewed workloads (?=1.0) and small cache size (1%), with our results comparing GDS and LRU being similar to those reported by the GDS authors in [6]. However, our results also show that GDS(1) and GDS(lat) perform poorly in terms of the latency and network bandwidth metrics. More precisely, the performance of GDS(1) based on latency, for all cache sizes is getting worse as ? grows. The performance of GDS(lat) based on latency is worse than of GDS(1) for all workloads and cache sizes, except for the case of skewed workload and large cache size. Similar behavior holds when considering network bandwidth requirements, with GDS(1) being the worst even though GDS(1) enjoys higher hit ratios, because it prefers to store small objects and fetch large ones from the web, which yields in higher network congestion.

A key result is that as the cache size grows and for Zipf distributions with parameter ? varying from 0.6 to 0.8, LRU performs equally-well or better than the other algorithms. The importance of this finding stems from the fact that disk-based caches are prevalent these days and that these ?-values are the most representative according to some studies [5]).

Through our experiments we found that having an admission control policy improves the performance and combining it with the auxiliary cache that acts as an additional filter can further improve performance. We examined how the size component in the caching gain function affects the performance and we concluded that it is worthwhile to include it because otherwise the hit ratio decreases, yielding poor performance. Finally, we observed that the popularity term in the caching gain function plays an important positive role in the proxy's performance.

**2.1.2 Major conclusions.** Throughout our study we observed that there is a disparity between hit ratio results and the results concerning our two other metrics (latency and bandwidth requirements). As you can see in Table 1, GDS(1) is the preferred policy among the others when considering hit ratio, while in other cases where the performance metric is latency or bandwidth requirements, GDS(1) is not as good as we would expect.

We also see that the performance results regarding the latency metric follow the same trends as the results regarding the bandwidth requirements metric.

We also noted that the disparity between the hit ratio and latency results is smaller than that between the hit ratio and the network bandwidth requirements results. This is due to the fact that the improvement in the hit ratio comes partly from expelling larger objects from the cache. However, fetching these larger objects from the web impacts adversely network bandwidth requirements in a direct way. On the other hand, latency is only partially adversely affected (since latency is also affected by other factors, such as the number of hops, link bandwidths, etc.).

Due to lack of space (our extended results quantifying the performance improvements are available online [25]), we present in Table 1 the chart for the problem's solution space. We should note that this table which shows the preferred algorithm (as well as the next best one in the case where the one is marginally better than the other) under various environment characteristics (e.g. workload skewness) can be later updated as other researchers may develop new replacement algorithms with better performance, or develop better latency models/results.

**Table 1. Preferred algorithm based on hit ratio, latency, and bandwidth requirements**

| Cache Size | ? Value | | | | | | ? Value | | | | | | ? Value | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Parameter of the Zipf Distribution | | | | | | Parameter of the Zipf Distribution | | | | | | Parameter of the Zipf Distribution | | | | | |
| % of Max. Required Space | Best Algorithm based on **Hit Ratio** | | | | | | Best Algorithm based on **Latency** | | | | | | Best Algorithm based on **Bandwidth** | | | | | |
| | 0.50 | 0.60 | 0.70 | 0.80 | 0.90 | 1.00 | 0.50 | 0.60 | 0.70 | 0.80 | 0.90 | 1.00 | 0.50 | 0.60 | 0.70 | 0.80 | 0.90 | 1.00 |
| **1%** | ▲ | ▲ | ▲ | ▲■ | ■ | ■▲ | ▲ | ▲● | ■ | ■▲ | ■ | ■ | ✕ | ●✕ | ✕ | ■✕ | ■✕ | ■✕ |
| **5%** | ▲ | ▲ | ▲ | ▲ | ■✕ | ■ | ▲✕ | ▲✕ | ▲ | ■✕ | ■ | ■✕ | ✕ | ● | ✕ | ✕ | ■✕ | ■ |
| **10%** | ▲✕ | ▲ | ▲ | ▲ | ✕ | ■ | ● | ● | ●▲ | ■● | ■● | ■● | ✕ | ● | ● | ● | ■● | ● |
| **15%** | ▲● | ✕ | ▲ | ▲✕ | ● | ✕ | ● | ✕● | ● | ■✕ | ■● | ● | ● | ✕ | ● | ● | ✕ | ■ |
| **20%** | ● | ●✕ | ▲● | ▲✕ | ✕ | ✕ | ▲✕✕ | | ● | ✕ | ✕ | ●✕ | ●✕ | ✕ | ● | ● | ✕ | ●✕ |
| **25%** | ● | ✕ | ✕ | ● | ✕ | ▲✕ | ▲●● | | ● | ✕ | ●✕ | ✕ | ●✕ | ● | ● | ● | ● | ● |
| **30%** | ▲✕ | ▲✕ | ▲✕ | ▲● | ▲●▲✕ | | ▲✕▲● | | ●✕ | ▲● | ● | ●✕ | ●✕ | ● | ● | ● | ✕● | ● |

**LRU:** ● **GDS(1):** ▲ **GDS(lat):** ✕ **CSP:** ■

## 3. Placement of proxies

### 3.1 Study setup

We model the network under which the proxy placement takes place as a graph with nodes and edges connecting them. This graph could represent the backbone network of a large ISP. Each one of the nodes covers a concrete geographic region with thousands of users connected to these nodes. The nodes are connected with weighted edges. These weights represent the time (in msec) that is required in order to transmit a packet of information from one node to another. Therefore, the edge weights show how loaded the links that connect each node are. This weight depends on the geographical distance of nodes, the delay on the routers, the delay due to packet processing, etc.

Nodes also have weight. The node weight represents the quantity of information that all users connected to this node request from the web origin server. Suppose, therefore, that there is an origin server connected to the network and that a trace file, which represents the expected workload of the origin server, is available; in order to calculate the node weights we follow this process: initially, users are grouped into clusters based on their geographical position. That means that users found to be geographically close belong to the same cluster. The clustering is based on user IP address (which exists in the trace file) and relative information from BGP Routing Tables [11]. The N larger clusters are randomly assigned to N nodes (where N is the total number of nodes). Then we calculate the amount of information (in MByte or Gbyte), which each cluster requests from the web server and this number constitutes the weight of each node.

The proxy placement algorithm will use edge and node

weights, the network topology information and the performance tables of Section 2 (like Table 1), in order to decide where and how many proxy servers to place.

**3.1.1 Proxy server performance estimation.** Apart from node weights, edge weights, and the network topology information, the performance impact of proxy servers plays an important role. The Byte Hit Ratio achieved by the selected cache algorithm, as it depends on the characteristics of the request stream and the available storage size will be used to measure the impact/benefits of placing a proxy at a specific location.

The estimation of the BHR (Byte Hit Ratio = 1 - Byte Miss Ratio (BMR)) value of a proxy server that is going to be placed in the network is based on the performance tables, which were derived through the experiments presented in Section 2 (i.e., we measured the proxy server performance under different sizes of available cache space, different values of the parameter ? of the zipf popularity distribution, and different replacement algorithms (LRU, CSP, GDS(1), and GDS (lat)). Thus, for each combination of these values, we know which the preferred replacement algorithm is, and its performance expressed in BHR.

The methodology therefore for assigning the right BHR to each proxy server is the following: initially for each cluster of users we calculate the characteristics of their request stream, which are the value of the parameter ? and the available cache size. The latter is expressed as the percentage of the maximum disk space required for storing all requested objects. (In other words if we consider that a proxy server, k, is deployed with 300 GB cache size and the aggregated size of all requested objects from the cluster that is connected to proxy k is 1000 GB, then the percentage is 30%). Finally, we consult the performance tables so as to find the BHR of the proxy server as well as the algorithm that is going to be used in order to yield in the best possible overall performance.

**3.1.2 Modeling of the total proxy placement cost.** Consider that $n\_wt_i$ is the weight of node i, $e\_wt_{ij}$ is the edge weight between the nodes i and j, d(i,j) is the minimum distance between the nodes i and j, N is the number of nodes in the graph, and Cost (G) is the total cost for a proxy placement in a network modeled by a graph G (we also refer to it as the total graph cost). The minimum distance d(i,j) is defined to be the minimum sum of weights along the path from i to j, and is computed after examining all the possible paths. Note that d(i,i) = 0.

We assume the existence of a collaborative environment under which proxy servers are placed. Thus, the objects that are not served by a proxy server can be served by a nearby proxy with a specific probability which is based on the BMR of the proxy that generates a miss. For example, consider the simple case where there are 2 proxy servers, between a web server and a node. Each proxy server has stored a quantity of information D. A percentage (we call it OL, OverLapping) of the D is common between the proxy servers. Therefore, the overlapped quantity of information is equal to OL*D. The remaining information (which is not common between the proxy servers) can produce a hit to the user requests, which have already generated a miss at the other proxy server. Thus, the quantity of information that was not served by the first proxy server (which is $BMR_2*n\_wt_1$) is served by the second proxy server with probability $OL*BHR_3$.

The algorithm for computing the graph cost is shown in table 2.

Having, therefore, in mind the cost calculation method, the placement algorithm tries to place the proxy servers in those nodes that will lead to the minimum total proxy placement cost.

We also observe that the distance between two nodes d(i,j) can model the transmission delay of a data packet from one node to the other or the number of hops in the path from i to j. In the first case, the graph cost reflects the time needed to serve all user requests (as they appear in the workload). In the second case, it reflects the bandwidth required for the given workload.

**Table 2. Graph cost calculation algorithm**

1. For each node beginning from i = 1 to N.
2. Find k, the closest proxy server to i.
3. Cost(G) = $n\_wt_i$ * d (i,k)
4. Find j, the closest proxy server to k.
5. Cost (G) = Cost (G) + $n\_wt_i$ * d (k,j) * $BMR_k$, where $BMR_k=1-BHR_k*OL$
6. k = j
7. Go to step 4 if k is not the server
8. i= i+1
9. Go to step 1 if i does not equal N
10. End of process.

**3.1.3 Web proxy cache placement algorithms.** We examined the performance of the well-known Greedy, HotSpot, and Random algorithms. In the study that follows we do not present the results for the tree-based algorithm [14], because its performance was found, as expected, to be comparatively poor when it is applied in a graph and not in a tree topology.

*The greedy algorithm*

The Greedy algorithm works as follows: Suppose that we want to place M proxies among N potential nodes. We choose one node each time. In the first iteration all the nodes are evaluated to determine which is more appropriate

for placing a replica. The total cost (section 3.1.2) of the system is calculated considering that all the requests from users are directed to the node that we select each time. In the end of the iteration we will select the node that will give us the smaller total cost. In the second iteration we try to select the second node that will store a replica (given that the first node is already selected) and finally we select the node with the smaller total cost. The total cost is calculated considering that the users' requests are directed to the node with replica that is closer to them, that is to say it leads to the smaller cost. This process is repeated until M nodes are found [16].

*The hot spot algorithm*

The basic idea behind this algorithm is the placement of replicas near communities of users that generate the greatest load. Thus, the N nodes are sorted depending on the amount of traffic that is generated in their neighborhood. The replicas are placed in the nodes whose neighborhood produces the greatest load. The neighborhood of node A is defined to be the circle centered at A with some radius. The radius can vary from 0 to the maximum distance between two nodes [16].

*The random algorithm*

In this algorithm the nodes that will contain the replicas are selected randomly.

## 3.2 Performance study

*Network topology*

For the performance study of the three algorithms (HotSpot, Random and Greedy) the backbone topologies of two major ISPs (AT&T and C&W) in North America were used. We have chosen these topologies, as simple examples, because we were readily able to obtain the monthly average values of edge weights, as these were collected from the websites that offer real time and monthly statistics of their network links.

*Graph cost calculation*

We compute the total cost of a proxy cache placement, following the way described in section 3.1.2. This means that a request, which generated a miss at a proxy server, can be served by the next closest proxy. The percentage of common data between the proxy servers is fixed to 55% [23], while the BHR is the same for all proxies and it is equal to 45%. As it was reported in section 3.1.1 the BHR (with the help of the described methodology) should be different for each node. However, for simplicity in our study we consider it to be constant and equal to 45%.

## 3.3 Performance results

The performance results were similar for both topologies, AT&T and C&W, so for brevity we show briefly only the results of the AT&T topology (Figure 1). It clearly appears that the Greedy algorithm is more efficient, while the Random, as it is expected, is the worst. We can see for example that with the placement of 2 proxies on the AT&T topology (Figure 1) we achieve 26% improvement with Greedy while the corresponding improvement for the HotSpot is around 12% and the Random around 8%. The corresponding percentage for the topology of C&W is for the Greedy 24%, for the HotSpot 5.5% and for the Random 4.5%. We also observed that the same behavior holds, even if we select different nodes of the network as servers. Comparing our results with the performance results of other researchers [16], which studied the same algorithms but for the full-server replica placement problem, we note that despite the quantitative differences in the relative performance of the algorithms, our results identifying the best algorithms to use are the same.

# 4. The proxy placement tool

Based on the above two studies on the placement of proxy servers in the Internet and on the performance results of cache replacement algorithms, we developed ProxyTeller that aims to give a solution to the placement problem using the Greedy placement algorithm. Likely, users will be administrators, who wish to place appropriately proxy servers in nodal points of the Internet. This tool could also be used by bandwidth-selling corporations, for associating envisaged investments with desired performance improvements. It can be used by content providers to estimate the off-loading of their origin servers, as well as the performance seen by their customers/consumers. Finally, it can be used also for testing different placement algorithms and how they interact with replacement algorithms. In the section that follows we present the tool's basic components and functionality.

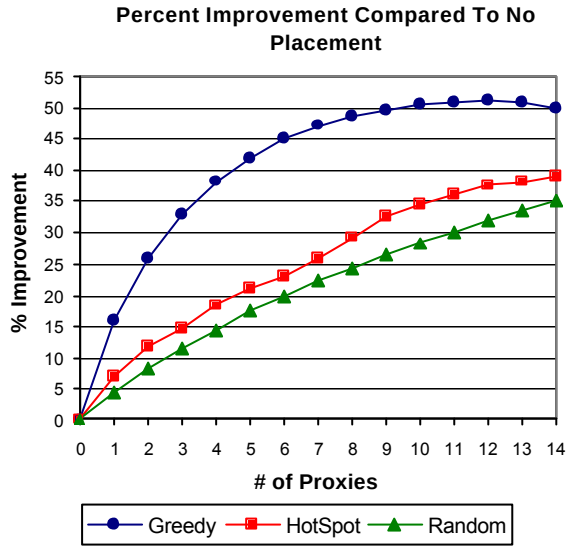


**Figure 1. Performance of placement algorithms after their application in the AT&T network**

## 4.1 Development platform

We used tools for developing interactive applications in the Internet, in order to make our placement tool easily accessible from a large number of users. In fact, a demo for the tool is available [24]. More precisely, we used the PHP language along with the support of the MySQL database management system. For the presentation of the tool and its interaction with the users, we used HTML, while for the controls of data imported by the user we selected javascript. The algorithms of proxy placement were developed with C++, while in certain points it was also essential to use Java.

The system, in which we developed the tool, supports a large number of users registering their data files (i.e. topology description, expected workloads) and their preferences in the database, after having followed certain simple steps in order to become members of the system.

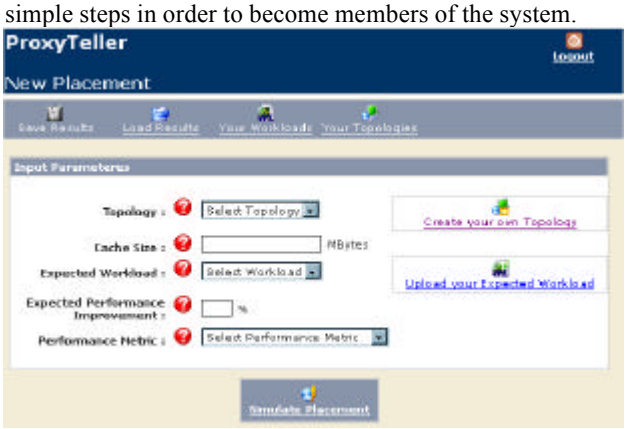


**Figure 2. The ProxyTeller Main Web Page**

## 4.2 A description of ProxyTeller

Initially the user attempts to enter the member section, by following certain simple steps in order to become a member of the system or inserting the username and password. In the member web page (Figure 2), the user must determine certain parameters that are essential for the execution of the algorithms, by filling suitably certain fields. These are:

*Network topology*

The network topology should be described suitably. Consequently, we use a specific format that describes the number of nodes, the number of edges, the edge weights, and the connectivity of each node. The user can select a predefined network topology, such as AT&T and C&W, or creating its own topology using a specific tool developed with Java.

*Expected workload*

As reported above, a trace file, which will contain the request stream, in order to create user clusters and the node weights is essential. Since the random assignment of clusters (and thus node weights, section 3.1) to nodes may lead to erroneous placement decision, the user can alternatively, provide N sub-traces for a N-node network topology, so as to associate each sub-trace to each node that generates the given workload. Also, the characteristics of workload are measured so as to determine the expected proxy server performance. Thus, the user can either import his trace file(s), which he believes it reflects the future demand, or use already existing trace files generated by SURGE.

*Available disk cache size*

The size obviously plays important role in the proxy server performance. Therefore, the user is asked to define the size of the proxy caches that are going to be placed in the network. The cache size is not necessary equal among all proxy caches. Thus the user can choose to place N proxy caches with different cache sizes.

*Desired performance improvement*

In this part the user is required to define the percentage of the desired performance improvement that she wishes to achieve, by placing proxy servers in the network. This percentage describes the reduction of the total cost (section 3.1.2) that is achieved because of the suitable placement, against the cost without any proxies at all in the network. Additionally, the user can define the number of proxies he wishes to place.

*Performance metrics*

The user is asked to determine the performance metric of interest for individual proxy servers. He has three choices: bandwidth requirements, latency, and hit ratio. Depending

on what the user selects, the proxy server will be described by a certain value of Byte Hit Ratio and the suitable replacement algorithm for each proxy server will be proposed (selecting one of LRU, GDS(1), GDS(lat) and CSP, from our results in Section 2).

*How it works*

After specifying these essential parameters the routine, which is responsible for proxy placement, is called. The Greedy placement algorithm (as described in section 3.1.3) is applied until the total cost reaches a value that meets the desired performance improvement or the number of proxies to be placed, specified by the user. More precisely, the tool starts by processing the expected workload in order to compute the node weights for the given topology, as described in section 3. In addition, the request stream from each cluster is being analyzed in order to define the value of ? of the popularity distribution that better describes each stream. This value and the available disk size are then used as an index to the performance table (such as Table 1) so as to specify the replacement algorithm and the performance (expressed with BHR) of the proxy, which may be placed in each node. Thus, at the end of this procedure the tool will be able to know the expected performance of a proxy placed at any node. The next step involves the finding of the minimum distance from one node to the other, for all pairs of nodes. This step is essential in computing the total proxy placement cost 3.1.2. Finally, the Greedy algorithm is being applied by placing more and more proxies, until the desired performance improvement or number of proxies is met.

At the end, the user will see the results of Greedy placement: the network with the proxy servers being placed, the percentage of improvement achieved, the number of proxies placed, and the replacement algorithm that the tool proposes for each proxy server.

Apart from this basic procedure (specifying the input parameters and waiting for the simulation results), we added extra functionality in order to help user organize its runs. Thus, the user can store and load simulation results, descriptions of network topology and expected workloads. All this information is stored in files in a directory designated for each user, while extra information for managing these files is stored in the database.

# 5. ProxyTeller put to use

## 5.1 Basic scenarios

In this section we show indicative scenarios for which ProxyTeller can be called to provide an answer. The scenarios are:

*1. Single web origin server with proxies / full replicas*

Consider a small organization, with a single web origin server, and its need to decide where to place proxy servers within the network to improve the efficiency of content delivery to clients. Supplying the expected workload (i.e., the web servers' trace file for a period of time), and the network topology to ProxyTeller, the tool can compute the best placement based on the expressed needs (i.e., how many proxy servers the organization can afford). Similarly, ProxyTeller can be called to place full replicas, instead of proxies, able to store all objects of the web origin server.

*2. Several web origin servers with proxies / full replicas*

This scenario could emerge when there are many web origin servers storing different content. For example, a large CDN may wish to decide where to place reverse proxies and/or full replicas within its network to improve efficiency.

*3. Incremental improvement: adding surrogate servers and/or full replicas in CDNs*

Within a CDN the goal can be to improve further the overall performance by placing additional surrogate and/or full replica servers.

The results of ProxyTeller can also be used to decide if it is preferable to place surrogate servers vs full replicas.

## 5.2 ProxyTeller results

In this section we present some test case results for the above scenarios. We used the AT&T network backbone topology to test these scenarios, since we were able to obtain real load statistics concerning the edge weights. We asked ProxyTeller to improve the overall performance by 50% and we used as expected workloads the one of the synthetic workloads used in the study we made on proxy replacement algorithms performance. The performance metrics studied include all three key metrics (latency, bandwidth, and hit ratio). Here we report the results of the first two.

**5.2.1 Proxy placement with latency constraints.** First, we looked at latency. The performance constraint was defined to be the minimization of the latency. The disk size was defined to be close to 30% of the maximum required space to store all requested objects (in the case where our goal is to place proxy servers).

For scenario 1, we found that by placing 4 proxies in the network we can achieve a performance improvement up to 56.02%. These four proxies should be placed at nodes 13, 10, 5, and 4. The tool suggested the LRU replacement algorithm for proxy at node 13 while for the rest of the proxies the preferred replacement algorithm is GDS(1).

For the same scenario, we need to place only 2 replicas at nodes 9 and 10, achieving 61.96% improvement. It is clear

that in this case we achieve better improvement but we have to pay the cost to equip the servers with enough space to accommodate the entire web origin server content.

For scenario 2, suppose that there are 3 web origin servers placed at nodes 1, 7 and 17. The ProxyTeller decides to place 3 proxies at nodes 5, 9, and 10, achieving an overall performance improvement of 51.24%. These three proxies should be employed with the GDS(1) replacement algorithm. Deciding to place full server replicas, (that is, scenario 4) we need to place two full replicas in order to achieve 65.17% improvement. We can see for example in this case that it may not be worth placing full replicas because by placing proxies we achieve our goal (50% improvement) and we avoid the cost of full replicas. This cost can be expressed as the storage investment needed to equip each full replica. Suppose for simplicity that each origin server stores an equal amount of data, S. Then each full replica server should be able to store 3*S quantity of data which yields a total capital investment of 2*(3*S). The corresponding investment for the proxy placement is 3*0.3*(3*S) = 0.9*(3*S). Apart from the storage cost, we should also notice the bandwidth required for the propagation of the replica updates in the case of full-replica placement. It is clear that proxy placement should be preferred over full-replica placement in such cases. Considering large-scale systems with hundreds of origins servers, the associated benefits will be much greater.

In scenario 3, the ProxyTeller suggests the placement of 4 surrogate servers at nodes 9, 10, 5, and 6. In this case we achieve an overall improvement of 53.88%. The corresponding results for full replica placement show that we should place 2 replicas at nodes 14 and 10, in order to achieve 56.41% improvement storage.

**5.2.2 Proxy placement with bandwidth constraints.** Next, we looked at the ProxyTeller results for minimizing the bandwidth requirements. We present the results when we are trying to place forward proxies or surrogate servers. Choosing as the appropriate performance metric the BW requirements in these scenarios may lead to a different choice of replacement algorithm for each of them and thus different BHR which in turn leads to different placement decisions.

Having in mind scenario 1, we found that by placing 4 forward proxies in the network we can achieve a performance improvement up to 56.16%. These four should be placed at nodes 13, 12, 16, and 4. The tool suggested the LRU replacement algorithm for proxy at node 13 and 12 and GDS(lat) for the rest.

For scenario 2, we have again 3 web origin servers placed at nodes 1, 7 and 17 (Figure 3, the proxies are in gray nodes and the web origin servers are the black node). The ProxyTeller decides to place 3 proxies at nodes 5, 14, and 10, achieving an overall performance improvement of 51.36%. Proxies 5 and 10 are employed with the GDS(lat) replacement algorithm while proxy 14 with LRU.

In scenario 3, the ProxyTeller suggests the placement of 4 surrogate servers at nodes 9, 10, 5, and 6. In this case we achieve an overall improvement of 54.72%.

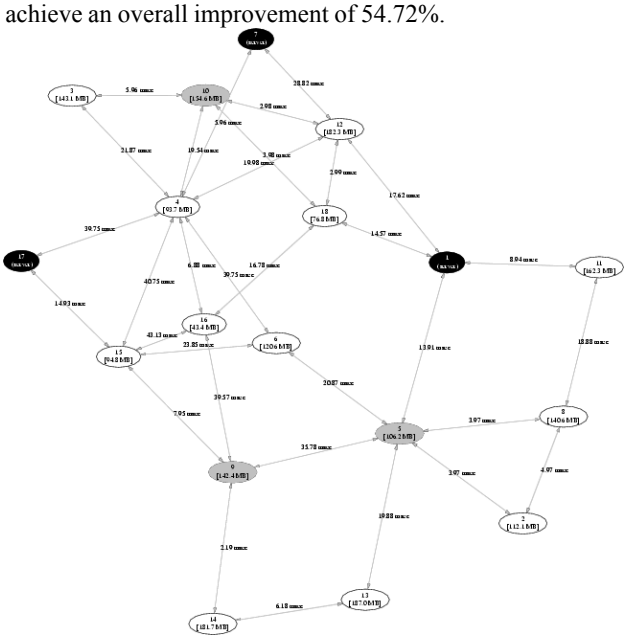

**Figure 3. The solution to scenario 2**

## 6. Concluding remarks

We have presented ProxyTeller, a novel tool deciding the efficient placement of proxies, providing answers to significant questions encountered when considering the efficiency of web content delivery. Specifically, the problem ProxyTeller solves accepts as inputs the network topology over which proxies will be placed, the desired sizes of their caches, the characteristics of the workload of the community served by a proxy (i.e., representative workload traces) and the performance improvement that is desired. The latter can involve one of the metrics of network bandwidth, user-request latency, and hit ratio and is specified in terms of the percentage improvement that is desired to be achieved, compared to the case where no proxies are deployed. The output of ProxyTeller is the position and the number of proxies required in order to achieve the stated performance goals.

ProxyTeller is based on two pillars, namely on:

1. a detailed performance study of the impact of proxy caches, as this depends on the cache replacement algorithms and related mechanisms. We undertook such a study, which culminated in the production of a problem-solution chart, identifying which approach ensures the best performance, under which circumstances (access

skew distribution and cache sizes) and for which performance metric (bandwidth, hit ratio, and latency). These results also quantified the relevant performance of each approach.

2. the adaption of full-server replica placement algorithms into a proxy (partial replica) placement setting and testing their performance, given the results from the previous study.

The tool can be used for both forward and reverse proxies and, we hope, it will be of real use to the organizations maintaining them. We note that our approach with respect to the efficient content delivery problem within the framework of CDNs is different than the one employed by works focusing on content placement. Instead of continuously deciding which objects to replicate and where, we decide how many proxies to place and where, with each proxy employing internally the best algorithms/mechanisms for deciding which objects to have in its cache, given the specific circumstances (cache size, access skews), utilizing thus the wealth of previous research results.

We believe ProxyTeller will be useful, and in particular:

- to ISPs/CDNs, in order to optimally decide on required capital investments, weighing them against expected performance improvements,
- to content providers, which will be able to estimate their origin server off-load possible (as this is depicted in cache hit ratios) from a potential contract with a CDN and the expected performance improvements seen by the consumers of their content, and
- to end-users, being able to understand the performance of possible applications they build and/or run, accessing such content.

Note also that the different parties engaged in an agreement for content delivery care about different metrics. Content providers care mostly about how much their origin servers will be off-loaded and what will be the perceived performance (latency) to their consumers. By using ProxyTeller with these metrics, they will see what benefits they are buying. In contrast, CDNs care mostly about investments on bandwidth and storage. Running ProxyTeller with the bandwidth metric, they will be able to know what bandwidth they can save and with what investment in storage. Overall, we hope, thus, that ProxyTeller is useful for all parties involved.

Thus, the significance of this work lies mainly in its ability to answer placement questions of high interest to all parties involved in a content delivery setting. We have presented some early case studies involving web content delivery and the associated results, indicating the types of answers and the utility of the tool.

As mentioned, we have implemented the tool, making it available through the internet, for use by the community [24]. Our ultimate goal is, through ProxyTeller, to accumulate and combine related research results, which form its basis (i.e., new results on the performance of various mechanisms improving the performance of single proxies, improving/extending the cache replacement problem solution chart, and new results for server/proxy placement), make it available to all, and provide the infrastructure to utilize it in order answer key questions faced in the context of efficient content delivery.

## 7. References


[1] M. Abrams, C. Standridge, G. Abdulla, S. Williams, and E. Fox, "Caching Proxies: Limitations and Potentials", in Proceedings of the 1995 World Wide Web Conference, December 1995.

[2] C. Aggarwal, J. L. Wolf and P. S. Yu, "Caching on the World Wide Web", IEEE Transactions on Knowledge and Data Engineering, vol. 11, no. 1, January/February 1999.

[3] M. Arlitt and C. Williamson, "Web Server Workload Characterization: The Search for Invariants", in the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, May 1996.

[4] P. Barford and M. Grovella, "Generating Representative Web Workloads for Network and Server Evaluation", in the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, 1998.

[5] L Breslau, P. Cao, L. Fan, G. Phillips and S. Shenker, "Web Caching and Zipf-like Distributions: Evidence and Implications", in Proceedings of the IEEE INFOCOM, March 1999.

[6] P. Cao and S. Irani, "Cost-Aware WWW Proxy Caching Algorithms", in Proceedings of the 1997 USENIX Symposium on Internet Technology and Systems, December 1997.

[7] M. Charikar, and S. Guha, "Improved Combinatorial Algorithms for the Facility Location and K-Median Problems", in Proceedings of the 40th Annual IEEE Conference on Foundations of Computer Science, 1999.

[8] A. Feldmann, R. Caceres, F. Douglis, G. Glass, M. Rabinovich, "Performance of Web Proxy Caching in Heterogeneous Bandwidth Environments", in Proceedings of the IEEE INFOCOM, March 1999.

[9] M. A. Habib and M. Abrams, "Analysis of Sources of Latency in Downloading Web Pages", in Proceedings of WebNet 2000, November 2000.

[10] B. Krishnamurthy and C. E. Wills, "Analyzing factors that influence end-to-end Web performance", in Proceedings of 2000 World Wide Web Conference / Computer Networks, May 2000.

[11] B. Krishnamurthy and J. Wang, "On network-aware clustering of web clients", in Proceedings of the ACM SIGCOMM 2000, August 2000.

[12] B. Krishnamurthy, C. Wills, and Y. Zhang, “On the use and performance of content distribution networks”, in Proceedings of the ACM SIGCOMM Internet Measurement Workshop, November 2001.

[13] T. M. Kroeger, D. D. E. Long, and J. C. Mogul, “Exploring the Bounds of Web Latency Reduction from Caching and Prefetching”, in Proceedings of the USENIX Symposium on Internet Technologies and Systems, December 1997.

[14] B. Li, M. J. Golin, G. F. Italiano, and X. Deng, “On the placement of Web proxies in the Internet”, in Proceedings of the INFOCOM 2000, March 2000.

[15] E. J. O’Neil, P. E. O’Neil, and G. Weikum, “An Optimal Proof of the LRU-K Page Replacement Algorithm”, in Journal of the ACM, Vol. 46 No. 1, 1999.

[16] L. Qiu, V. N. Padmanabhan, and G. M. Voelker, “On the placement of Web server replicas”, in Proceedings of the IEEE INFOCOM 2001, April 2001.

[17] M. Rabinovich and O. Spatscheck, “Web Caching and Replication”, Addison-Wesley, ISBN: 0-201-61570-3, 2002

[18] L. Rizzo and L. Vicisano, “Replacement Policies for a Proxy Cache”, Research Note RN/98/13, Department of Computer Science, University College London, 1998.

[19] J. Shim, P. Scheuermann and R. Vingralek, “Proxy Cache Algorithms: Design, Implementation and Performance”, in IEEE Transactions on Knowledge and Data Engineering, vol. 11, no. 4, July/August 1999.

[20] R. Tewari, H. M. Vin, A. Dan and D. Sitaram, “Resource-based Caching for Web Servers”, in Proceedings of the SPIE/ACM Conference on Multimedia Computing and Networking, January 1998.

[21] A. Venkataramani, P. Weidmann, and M . Dahlin, "Bandwidth constrained placement in a WAN", Twentieth ACM Symposium on Principles of Distributed Computing (PODC 2001), August 2001.

[22] S. Williams, M. Abrams, C. R. Standridge, G. Abdulla and E. A. Fox, “Removal Policies in Network Caches for World-Wide Web Documents”, in Proceedings of the ACM SIGCOMM, 1996.

[23] A. Wolman, G. Voelker, N. Sharma, N. Cardwell, M, Brown, T. Landray, D. Pinnel, A. Karlin, and H. Levy, “Organization-Based Analysis of Web-Object Sharing and Caching”, In Proceedings of the USENIX Symposium on Internet Technologies and Systems, 1999.

[24] ProxyTeller: A Proxy Cache Placement Tool. Demo available at http://pcp.softnet.tuc.gr

[25] Detailed Performance Results of Cache Replacement Algorithms, available at http://pcp.softnet.tuc.gr/docs/ProxyPrf.htm