WWW Cache Modelling Toolbox

George Bilchev, Chris Roadknight, Ian Marshall, and Sverrir Olafsson

BT Research Laboratories, Martlesham Heath, Ipswich, Suffolk, IP5 3RE, UK
{george.bilchev, sverrir.olafsson}@bt-sys.bt.co.uk
{roadknic, marshall}@drake.bt.co.uk

Abstract. This paper develops and implements a World Wide Web cache infrastructure model which is to be used for analysis of features that are otherwise difficult to get from existing log data or for evaluation of non-existing cache scenarios. A prominent feature of our model that differentiates it from other similar models is its dynamical aspect, which allows for the investigation of temporal features. Using the model we verify and quantify an observation made from real log data that the popularity of Web pages diversifies the higher we go in the cache hierarchy. We then use the model to predict the cache population dynamics in a hypothetical scenario of sufficiently large caches.

1. Introduction

Proxy caching has become an established technique for enabling effective file delivery within the World Wide Web architecture [1][2]. The addition of file caching agents adds many positive features including robustness (by distributing files more widely), a possible reduction in total bandwidth requirements (by moving popular files near to the clients) and a reduction in pressure on origin servers, especially on those serving popular files. Understanding precise costs and benefits of inserting caches into the network is a highly desirable goal for network management and design.

To gain an understanding of what affects a cache's behaviour and performance it has been essential to analyse behaviour of existing WWW caches currently in operation [3][4][5]. This analysis gives us some information about the inter-relationships of cache metrics and possible causes of observed behaviour [6] but only covers caches in existing locations, serving existing communities. It is therefore highly desirable to be able to model cache behaviour so that non-existing cache scenarios can be evaluated. A cache enabled WWW modelling toolbox would undoubtedly be of use to network planners but also to many Internet researchers looking for a simple, flexible model to test theories with.

In this paper we develop a WWW cache model that is easy to use, cheap to implement and fast to simulate. The model only requires a few simple input values and yet is realistic enough to verify observed data from real caches. We believe the model will be of particular interest to operator technical staff who are not PhDs and work under short time scales.

2. Previous Work

There are two main types of WWW cache modelling approaches. The first type concerns modelling individual components such as generating representative Web traffic and feeding real caches in order to test and analyse them. Two well known examples of this approach are the Wisconsin Proxy Benchmark [7] and SURGE [8].

The second approach consists of mathematical modelling at a higher level of abstraction where the variables of interest usually represent average values. For example, [9] develops explicit formulas for the hit rate as a function of cache size of a single proxy using probability theory. Another example is [10] where the authors describe a model comprising various levels of a caching infrastructure. The advantage of the mathematical approach is that the models are relatively fast to simulate. The disadvantage is that often these models lack the desired detail, i.e., oversimplifying assumptions are made which might turn out to be significant.

In this paper we have adopted the second approach since it allows for larger scale models that are cheap to implement and easy to simulate.

3. Modelling WWW Caches

Previous mathematical models of WWW caching [9][10] mostly consider variables representing some aggregated average value, i.e., average distance of a WWW server from its clients, average document size, average number of requests made by a single client, etc. Although these models are not computationally very intensive to simulate, their modelling granularity makes

it difficult, if not impossible, to investigate parameters of interest such as the effect of document modifications for example [9, p.15].

Since one of our main goals in modelling Internet caching is to carefully consider the dynamics of the caching process and its scalability with respect to file modifications rate, file popularity, cache size, cache management algorithms, etc., we require greater modelling detail. Therefore, we have selected to model the individual pages with attributes such as popularity rank and time-stamp. Also, in order to have a desired degree of freedom in our simulations we have chosen to use synthesised input data (as opposed to a log-based trace-driven simulation). This has been achieved by developing a realistic model of a user community, which generates file requests from a Zipf-like distribution following a specific daily activity pattern [11]. The rest of this section is organised as follows. First we develop models of a user community and a single proxy. Then we describe how single proxy models are connected into an overall model of a caching infrastructure.

3.1. Model of a Single Proxy

A schematic representation of a single proxy cache as seen in the simulation toolbox is shown in fig. 1. The proxy can accept input from a number of user communities. Each user community is modelled by daily activity pattern (fig. 2a) and popularity statistics (fig 2b).

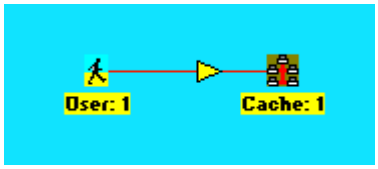


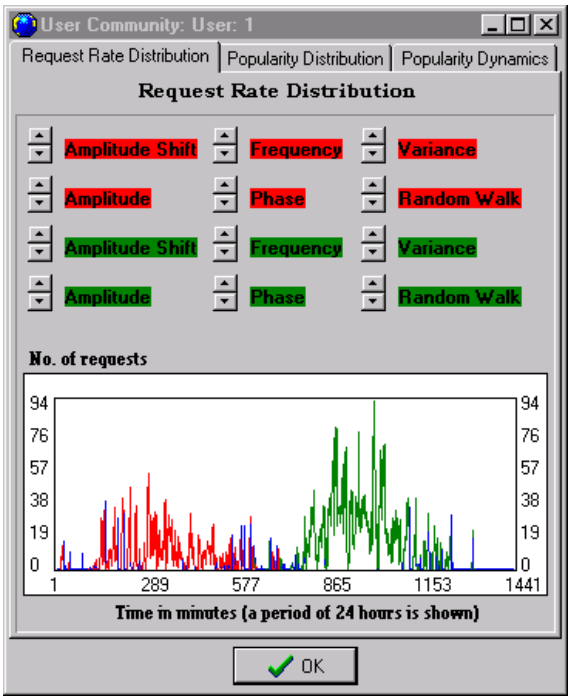
Fig. 1. A schematic representation of a single proxy cache as seen in the simulation toolbox. The cache is fed in by one user community annotated as "User: 1".

The daily activity pattern consists of an underlying trend and a stochastic component. To model the underlying trend we suggest using a superposition of periodic functions:

$$y_i^{\text{trend}}(t) = \max\left\{a_i + b_i \sin(2\pi c_i \frac{t}{T} + d_i), 0\right\}$$
$$y^{\text{trend}}(t) = \max_i\left\{y_i^{\text{trend}}(t)\right\} \quad (1)$$

where a_i is an amplitude shift, b_i is the amplitude, c_i is the frequency, d_i is the phase and T is the period during which cyclic patterns are observed.

a)



b)

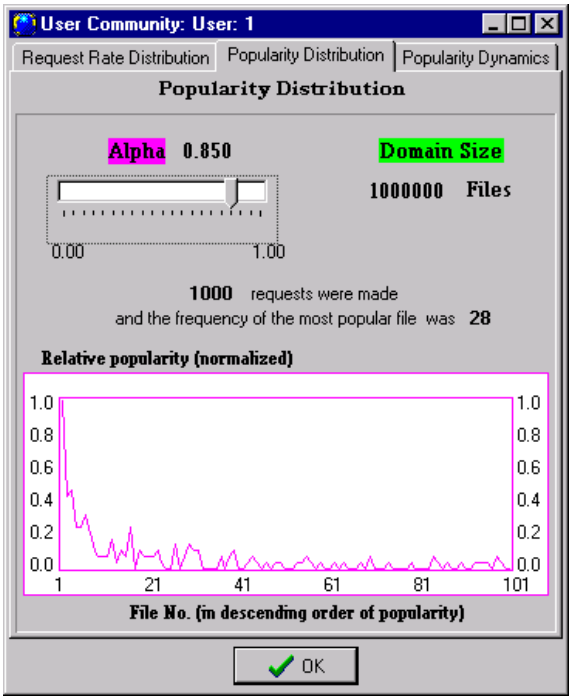


Fig. 2. Interface to the user community model. The graph on the left shows how the daily activity pattern can be tuned to approximate an observed pattern. The graph on the right shows the tuning of the file popularity as observed from the user community model.

Once the trend has been approximated the stochastic component can be modelled as a Brownian motion:

$$y^{\mathrm{BM}}(t) = y^{\mathrm{BM}}(t-1) + \eta \qquad (2)$$

Two points are worth mentioning. First, since the number of requested files is always non-negative we have to truncate a negative value of $y^{\mathrm{BM}}(t)$ to zero. Second, bursts in positive direction are higher than bursts in negative direction. To accommodate for this we define η as:

$$\eta = \begin{cases} \eta' & \text{if } \eta' > 0 \\ \dfrac{\eta'}{\lambda} & \text{otherwise} \end{cases} \qquad (3)$$

where $\eta' \in \mathrm{Norm}(0, \sigma)$ and λ is a parameter determining the ratio between the heights of the positive and negative bursts. The second modification also has the effect of reducing the number of times the series has to be truncated due to negative values.

Since the auto-correlated stochastic component (2) must be superimposed on the trend (1), a way of "guiding" the random walk of the Brownian motion towards the trend without destroying the desired properties is needed. We suggest using a sequence of non-overlapping random walks each starting from around the trend:

$$y(k\Delta t) = y^{\mathrm{trend}}(k\Delta t) + Norm(0, \sigma^{\mathrm{trend}}) \qquad (4)$$

i.e., at each time step $k\Delta t, k = 0,1,2,\ldots$, a Brownian motion process begins for Δt steps:

$$y^{\mathrm{BM}}(k\Delta t + m) = y^{\mathrm{BM}}(k\Delta t + m - 1) + \eta \qquad (5)$$

where, $m = 1,2,\ldots,\Delta t - 1$. Then it stops and a new process begins. This completes our model of the intensity of the http requests (i.e. the daily activity pattern). But before we can use it in our simulations of Internet caches we also need to define the popularity distribution of the requests. There is significant evidence in the literature suggesting that the popularity distribution follows a Zipf's-like law, where the relative popularity of the i^{th} most popular file is given by:

$$p_i^{relative} = \frac{1}{i^{\alpha}} \qquad (6)$$

Therefore, we also need a random number generator that produces Zipf's distributed numbers. We define it in the following way. First the total domain size *N* and the exponent α must be specified. Then the probability of selecting file *i* is given by:

$$p_i = \frac{i^{-\alpha}}{\sum_{j=1}^{N} j^{-\alpha}} \qquad (7)$$

A uniform random number *n* is generated in the range between 0 and 1 (most programming languages have already defined uniform random number generators) and an index *k* is found such that the following inequalities hold:

$$n \le \sum_{j=1}^{k} p_j \qquad n > \sum_{j=1}^{k+1} p_j \qquad 8)$$

The index *k* is the desired random number coming from the specified Zipf's-like distribution.

After developing the user community model we proceed with the cache proxy model. The cache proxy is modelled by Web content expiry statistics (fig. 3) and it also implements a simplified caching algorithm. The expiry statistics models both the rate of change of Web pages (reflecting server assigned TTL) and cache purging due to stale data (i.e., cache assigned TTL). For example, fig. 3 shows that in this particular case about 14% of the pages are not cacheable (i.e., cookies, stock quotes, etc.) and that the cache purges all files older than about two months (80640 minutes). These parameters are, of course, flexible. If the cache is of limited size, the model also implements a cache replacement algorithm. Currently only the well-known least recently used (LRU) data replacement algorithm is considered.

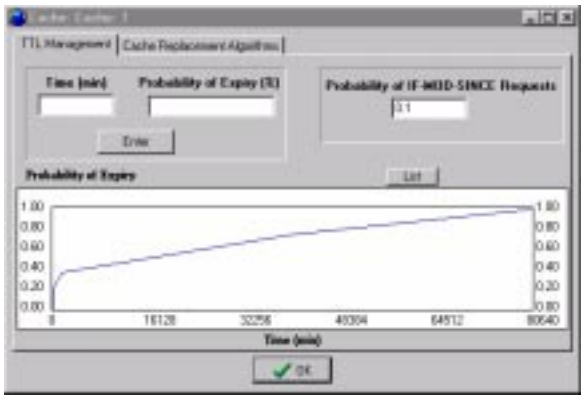


Fig. 3. Parameters defining a proxy cache model. The probability of expiry reflects both the server assigned TTL and the cache assigned TTL.

The simulation works as follows. The proxy cache receives requests for individual files. It checks if the requested file has already been registered in the cache model before. If not, the file is time-stamped, registered in the cache and a miss is reported (i.e., this reflects downloading the file from the origin server and caching it). If the requested file has been registered in the cache before, the proxy checks the TTL. If the file has expired the proxy verifies whether the file has changed. If so, a miss is reported and the time stamp of the file is reset (this reflects downloading of the new version of the file from the origin server). If the file has not changed, a hit is reported and again the time stamp is reset.

3.2. Overall Network Model

Once the single proxy has been modelled, we can build meshes of interconnected proxy caches. To achieve this the simulation toolbox allows two caches to be linked together. We consider two types of links:

- *Parental* links in which a miss is propagated to the parent cache and it is responsible for providing that file back either from its cache neighbourhood or from the origin server. On its way back the file is cached at each parental level.
- *Peer* or *sibling* links, in which a peer proxy only checks if it has the file in its neighbourhood, but does not download it from the origin server in case of a miss.

Using the above-described links we can build a number of caching structures. For example, fig. 4 shows how two user communities can be "merged" together to use the same proxy cache. This can be useful to analyse file popularity (see end of next section), i.e., what happens to the file popularity generated by a superposition of user communities as compared to the individually generated file popularity.

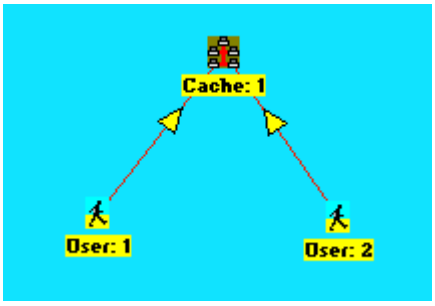


Fig. 4. Merging two user communities

As another example consider fig. 5, which shows how several first level caches can be connected to a parental higher level cache. This can be useful to analyse indirect co-operation among caches from the same level.

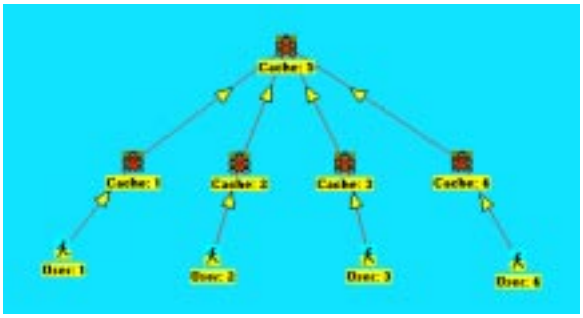


Fig. 5. A higher level cache.

4. Simulations and Analysis

In order to give an idea about what kind of analysis can be done with our toolbox this section presents some simulation examples.

In a previous paper [6] we have investigated file popularity statistics from real log files. One of the major hypotheses we attempted to analyse was how file popularity changes at various levels of the caching infrastructure. Two results have been found: 1) the time during which samples are collected doesn't affect the file popularity curve, and 2) higher level caches exhibit smaller absolute values for α, where α is a parameter from the Zipf-like popularity distribution:

$$p_i = i^{\alpha}$$

and usually has negative values in the range between –0.5 and –1. In order to test the above findings using our toolbox we design three experiments. In each of the experiments we use a user community that feeds a single proxy cache (as shown in fig. 1). Three different user communities have been tested: 1) a user community that generates 15,500 file request per day coming from a Zipf-like distribution with α=–0.75, 2) a user community with the same activity, but with α=–0.65, and 3) a more active user community generating 50,000 requests per day with α=–0.75. We have simulated the three cases and approximated the popularity distribution using the same methods as in [6]. Using a sample size of 500,000 requests we have found that the approximation of α for user communities one and three doesn't seem to be affected by their activity. In both cases the approximated value of α is

–0.73, which is 2.7% different than the expected. This discrepancy has been observed in all three cases and doesn't seem to significantly decrease with increasing the sample size beyond 500,000. The most probable explanation for this is that the employed approximation method gives a slightly biased estimate.

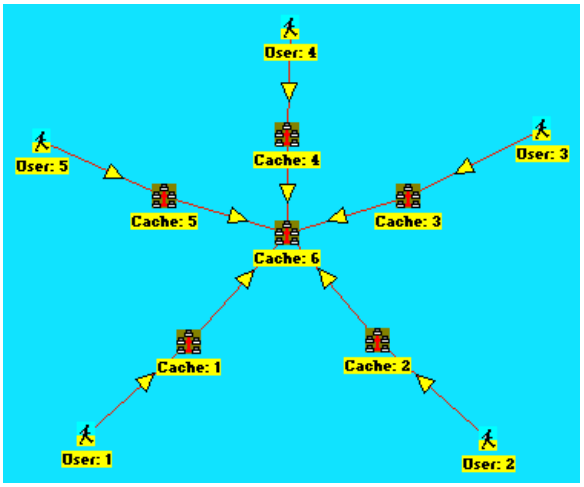


Fig. 6. Architecture of a higher level cache used in the experiments.

A new experiment has been staged in order to verify our second finding that the absolute value of the α parameter decreases in higher cache levels. The structure of the experiment is shown in fig. 6, where five first level caches feed one parental second level cache. All of the user communities generate file requests with α=–0.75 (i.e., the measured approximated value of α will be –0.73). We approximate the α parameter at the second level cache using the same methods as described in [6] and used in the previous experiments. We repeat the above experiment for various first level cache sizes (all first level caches have the same size in this scenario). The following results have been obtained:

Level 1 Cache Size (No. of cached files as % from the overall simulated domain)	**Level 2 α**	**R^2**
10	–0.4632	0.9319
3	–0.5696	0.9050
1.5	–0.5986	0.9014
1	–0.6024	0.8973
0.5	–0.6166	0.9044
0.25	–0.6631	0.9365
0.1	–0.6858	0.9104
0.01	–0.6930	0.9566

Results indicate that the more effective the first level caches (i.e., more cached files in this case), the smaller the absolute value of the α parameter. This means that the value of α in the higher level caches is significantly affected by the filtering effect of the lower level caches. In practice the cache efficiency due to cache size can be roughly approximated using estimations of the total static Web space. As of March 1998 the total number of static Web pages was estimated at 275,000,000 pages [12]. Assuming the average size of a page is 13K this totals to 3,575 Gbytes. A first level cache such as Funet [13] is 12Gbytes which is 0.34% of the total domain size. Thus, for example, we would expect a cache connected to a higher level cache such as NLANR [14] to have absolute value of α about 0.1 larger, which is approximately what was observed in [6].

In the above experiment we have used user communities with the same file popularity and first level caches with the same size. In practice, however, it is more likely that user communities with different file popularity will feed higher level caches and also that the first level caches will be of different sizes and efficiency. Moreover, most probably there will be other features that will also affect the popularity exponent value such as the popularity ranking order. The developed toolbox can be used to further understand these processes. For example, we have tested the change in the popularity exponent α caused by "merging" two user communities (fig. 4). The simulation predicted that the absolute value of the exponent of the merged user community would be greater than the average of the two exponents, provided that the user communities are equally active and using the same ranking order. We have later verified these predictions by analysing log data from Funet [13], NLANR [14], and EduWeb [15]. In the first experiment we have merged 500,000 requests received at each of NLANR and EduWeb, and have calculated the new popularity exponent. In the second experiment we have taken two 500,000 requests from the same cache (Funet) and after merging them we have calculated the new popularity exponent. Results are shown in the following table:

	First 500,000 Requests	Second 500,000 Requests	Aver.	Merged	diff
Model Prediction	–0.730	–0.624	–0.677	–0.690	1.92%
NLANR and EduWeb	–0.703	–0.858	–0.780	–0.794	1.79%
Funet	–0.738	–0.676	–0.707	–0.718	1.56%

The calculated popularity exponent of the merged requests in all cases is less than the average as predicted by the model. It is possible, however, that sometimes the prediction will fail due to features not yet incorporated into the model such as dynamics of the popularity ranking order.

5. Burstiness

Burstiness is a phenomenon frequently observed in data networks [16]. It often has a crucial effect on system's performance and makes it more difficult to provision the right network capacity. Any realistic model, therefore, should accurately reflect the burstiness of the real processes. To test our toolbox we compare the long range correlations of the hit rate exhibited in real log data and those observed in our model (fig. 7). As seen by the values of the Hurst parameter, results clearly indicate that the model provides sufficient degree of burstiness, which agrees with the observed data. We have also identified that the degree of burstiness of our model is mostly affected by the domain size, the long range correlations in the file request pattern and probably the file popularity ranking dynamics. These features are the likely reasons as to why the absolute value of both graphs differs and we plan further studies on the file ranking dynamics to validate in detail the assumptions made in the model.

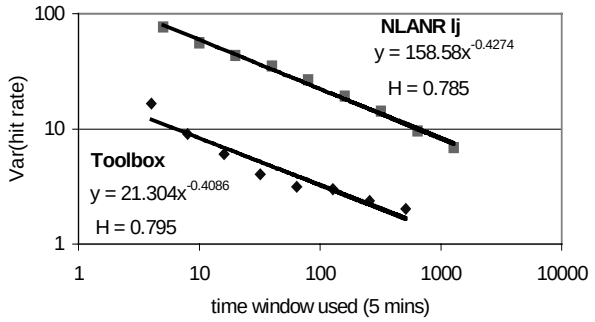


Fig. 7. Hurst parameter approximation for the hit rate measured at NLANR and our toolbox model.

6. Discussion

The presented Internet cache toolbox is aimed at helping to better understand the processes that take place in real caching systems and to develop new infrastructures for data dissemination on an ever-increasing scale. The advantages of building realistic WWW cache models can be exploited to investigate features that are otherwise difficult if not impossible to test with existing log data information. For example, we can consider caches of "unlimited" size and ask the question what is the maximal hit rate that can be achieved. Assuming that the user community stays constant (i.e., the generated file popularity stays the same throughout the period of investigation) the hit rate will mainly be determined by the number of cached files. In general there are two opposite forces that drive the dynamics of the cache population size (i.e., the number of cached files). The incoming file requests tend to build up the cache population while the file expiry statistics and the cache purging heuristics tend to reduce the cache population. We would expect that these two processes eventually will negotiate an equilibrium level, but it is difficult to guess the dynamics of this equilibrium, i.e., is it stationary, self-similar, etc. In order to answer these questions we can run a simulation of a user community and a proxy cache, where the cache is of "unlimited" size (in the implementation the cache size is simply equal to the total domain size). A typical run for a user community generating 15,500 requests per day with popularity α=–0.75 is shown in fig. 8. Clearly the dynamical behaviour of the cache size level exhibits some long-range dependencies as can be seen from the Hurst parameter evaluated in fig. 9.

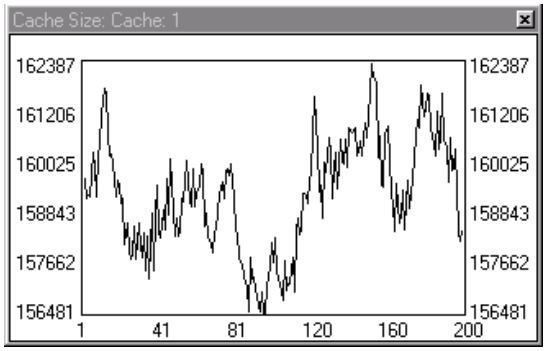


Fig. 8. Dynamics of cache population size of an "unlimited" cache over a period of 200 days. The cache serves a user community generating 15,500 requests per day on average, choosing from a total of 1,000,000 files.

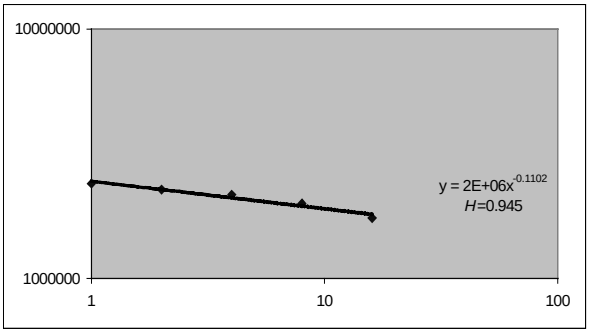


Fig. 9. Evaluation of the Hurst parameter H for the data from fig. 8. The graph shows the variance of the cache population size calculated for different non-overlapping time windows. H is defined as $\beta=2H-2$, where β is the slope of the line fitted to the calculated data points from the graph.

The above experiment can be used to give some indication as to how to dimension the cache size. For example, the cache size may be designed to be proportional to the mean equilibrium cache size from fig. 9.

7. Conclusions

In this paper we have presented an analytic WWW cache model capable of realistically simulating some aspects of real Internet cache structures in real time, with a low overhead in terms of CPU usage and log analysis. The model only requires a few simple input values. Nevertheless, we have demonstrated that the model can be used to verify observed data from real caches and to predict detailed new results. We therefore claim that the simplifying assumptions we have made appear to work. The savings will be of great benefit to operators needing to plan capacity in their cache networks. The model is not yet complete, as there are some extra features to be incorporated such as popularity dynamics.

References

[1] M. Abrams, C.R. Standridge, G. Abdulla, S. Williams and E.A. Fox. Caching Proxies: Limitations and Potentials. *Proc. 4th Inter. World-Wide Web Conference*, Boston, MA, Dec. 1995.

[2] M. Baentsch, L. Baum, G. Molter, S. Rothkugel and P. Sturm. Enhancing the web's infrastructure: From caching to replication. *IEEE Internet Computing.* March 1997. pp. 18-27.

[3] C Roadknight, I Marshall, Variations in cache behaviour, in *Computer Networks and ISDN systems* 30 (1998), pp.733-735.

[4] I Marshall, C Roadknight, Linking cache performance to user behaviour, Computer Networks and IDSN Systems, 30, pp. 2123-2131.

[5] M. Arlitt and C. Williamson. Web server workload characterization: The search for invariants. In Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, May 1996.

[6] C. Roadknight, I. Marshall and D. Vearer. File Popularity Characterisation. Submitted to the 2nd Workshop on Internet Server Performance (WISP 99)

[7] Jussara Almeida and Pei Cao. Measuring Proxy Performance with the Wisconsin Proxy Benchmark. J. of Computer Networks and ISDN Systems, To appear, 1999.

[8] Paul Barford and Mark Crovella. Generating representative web workloads for network and server performance evaluation. In Proceedings of the Joint International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS '98/PERFORMANCE '98), pages 151-160, Madison, WI, June 1998.

[9] Lee Breslau, Pei Cao, Li Fan, Graham Phillips and Scott Shenker, Web Caching and Zipf-like Distributions: Evidence and Implications,. To appear in Proceedings of Infocom'99.

[10] M. Baentsch, A. Lauer, L. Baum, G. Molter, S. Rothkugel, P. Strum, World-Wide Web Caching: The Application-Level View of the Internet, IEEE Communications, June 1997

[11] George Bilchev, Ian Marshall, Sverrir Olafsson, and Chris Roadknight, Modelling Http Traffic Generated by Community of Users, High Performance Computing and Networking Europe '99.

[12] Measuring the Web, http://www.research.digital.com/SRC/whatsnew/sem.html

[13] Funet: The Finnish University and Research Network, http://www.funet.fi/

[14] NLANR: A Distributed Testbed for National Information Provisioning, http://ircache.nlanr.net/

[15] EduWeb: The Leading Education Internet Service, http://www.eduweb.co.uk/

[16] V. Paxson and S Floyd, Wide area traffic: the failure of Poisson modeling. IEEE/ACM Transactions on networking 3, p.226 - 244, 1995