Optimization of cache expiration dates in content networks

Héctor Cancela
Universidad de la República, Uruguay
J. Herrera y Reissig 565
Montevideo, Uruguay
+598-2-7114244 ext. 113
cancela@fing.edu.uy

Pablo Rodríguez-Bocca
Universidad de la República, Uruguay
J. Herrera y Reissig 565
Montevideo, Uruguay
+598-2-7114244 ext. 113
prbocca@fing.edu.uy

ABSTRACT

One of the fundamental decisions in content networks is how the information about the existing contents is deployed and accessed. In particular, in many content network architectures, there are access nodes which cache information about the contents location, in order to ensure a quick answer to queries formulated from the peer nodes.

In this work we present a simplified model of the costs and restrictions associated with these cache expiration dates in a content network, which regulate the proportion of queries which will be answered on the basis of cached information, vs. those which will give rise to additional searches in the network backbone.

This model gives rise to a mathematical programming formulation which can be useful to determine the optimal cache expiration dates in order to maximize the total information discovered, while respecting the operational constraints of the network. We apply the model to DNS data to obtain some insights about the behavior of the model and the optimization procedures.

1. INTRODUCTION

A content network is a network where the addressing and the routing of the information is based on the content description, instead of on its physical or logical location [7][8][10]. Content networks are usually virtual networks based over the IP infrastructure of Internet or of a corporative network, and use mechanisms to allow accessing a content when there is no fixed, single, link between the content and the host or the hosts where this content is located. Even more, the content is usually subject to re-allocations, replications, and even deletions from the different nodes of the network.

In the last years many different kinds of content networks have been developed and deployed in widely varying contexts: they include peer-to-peer networks, collaborative networks, cooperative Web caching, content distribution networks, subscribe-publish networks, content-based sensor networks, backup networks, distributed computing, instant messaging, and multiplayer games. The ability of content networks to take into account different application requirements and to gracefully scale with the number of users have been a main factor in this growth [14][15][16].

As we have previously discussed, in a content network the addressing and routing are based on the content description, instead of on its location. This means that every content network is actually a knowledge network, where the knowledge is the information about the location of the nodes where each specific content is to be found: this is "meta-information", in the sense of being the information about the information contents themselves.

The objective of the network is to be able to answer each content query with the most complete possible set of nodes where this content is to be found; this corresponds to discover the content location in the most effective and efficient possible way.

As both nodes and contents are continuously going in and out of the network, the task of maintaining updated the network meta-information is very difficult and represents an important communication cost. In this context, cache nodes are used to hold the available meta-information; as this information is continuously getting outdated, the cache nodes must decide when to discard it, which means increasing communication overhead for the sake of improving the quality of the answers.

These last years have seen an explosion on the design and deployment of different kinds of content networks, in most cases without a clear understanding of the interaction between the network components neither of the tuning of the network architecture and parameters to ensure robustness and scalability and to improve performances. This in turn has lead to a still small but growing number of empirical studies (based on large number of observations of a given network activity) [6][16][18][19][23], and of analytical models which can be fitted to the observations in

order to better understand and eventually to predict different aspects of network behavior [3][14][15][20][21].

In this work, we develop a simplified model of a content network, and in particular of the number of correct answers to a query as a function of the information expiration times used at the cache nodes, presented in Section 2; to the best of our knowledge, this is an aspect that has not been previously treated analytically in the literature. This model gives rise to a mathematical programming formulation discussed in Section 3, which can be used to find the expiration times maximizing the correct answers to the queries received; a numerical illustration is shown in Section 4, followed by some conclusions in Section 5.

2. CONTENT CACHING PROBLEM FORMULATION

This section formalizes the problem of caching meta-information in a content network in order to maximize the number of correct answers to the queries, while respecting the bandwidth constraints.

2.1 Network components description

Figure 1 presents a graphical representation of a content network composed of source nodes and querying nodes (which may actually coincide), of cache nodes (also called aggregation nodes), and of a backbone (which will not be modeled in detail). This is a conceptual division, which not necessarily has a correspondence at the equipment level, as the same computer may act at the same time as a source node, a querying node, a cache node, and a backbone node. The contents hosted in the network belong to a given set *C*, which is partitioned into K content classes, such that if two contents belong to the same class, all their parameters are identical ($C = C_1 \cup C_2 ... \cup C_K$); we denote by l_k the number of contents of this class: $\|C_k\| = l_k \ \forall k \in [1..K]$. The total number of contents in the network is then: $\|C\| = \sum_{k \in K} \|C_k\| = \sum_{k \in K} l_k$.

We will look at the steady state behavior of the content network, so that we will not need to explicitly model time. This hypothesis can be justified by the fact that the rates of content locations changes and the rates of cache turnovers expiration times are usually much faster than the changes in the statistical properties of the user population.

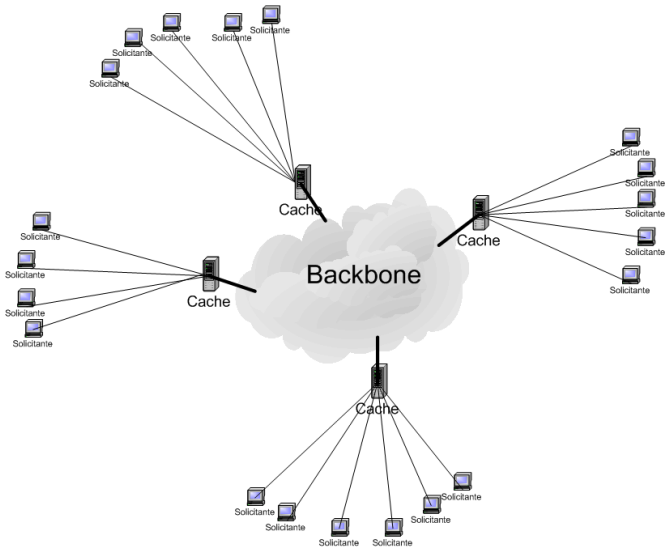

Figure 1: Simplified view of a content network

The users of the network will query about each content belonging to class *k* with query frequency f_k . We suppose that the number of users is large enough so that for each content, the queries follow a Poisson process of rate f_k This means that $S_k(T)$, the number of queries for each content of class *k* in a given time interval *T*, will have the following distribution:

$$p(S_k(T) = n) = \frac{(f_k T)^n e^{-f_k T}}{n!} \quad \forall k \in C, \forall n \in \aleph, \forall T \in \Re^+ .$$

Also T_{S_k} , the time between two consecutive queries for each different content in class *k*, will be an exponentially distributed random variable with parameter f_k :

$$p(T_{S_k} \le t) = \begin{cases} 1 - e^{-f_k t} & t \ge 0 \\ 0 & t < 0 \end{cases},$$

$$\overline{T_{S_k}} = \mathrm{E}\{T_{S_k}\} = 1/f_k .$$

The contents will be located in the source nodes; each source node decides when to start and when to end lodging the different contents. This leads to a different birth-and-death process for each content of class *k*, which we will suppose will be of M/M/∞ type and parameters λ_k and μ_k (respectively, the rates of start and end of lodgment of a content of class *k* at a source node); if we suppose that at moment t_0 the network is in stationary state, and $A_k(t_0)$ is the (random) number of source nodes lodging content *k* at t_0 we have that:

$$p(A_k(t_0)=n)=\frac{\left(\lambda_k/\mu_k\right)^n e^{-\lambda_k/\mu_k}}{n!}\quad \forall k\in C, \forall n\in \aleph..$$

From this distribution, we can find the expected number of source nodes lodging content k (i.e., the expected number of times this content will be replicated in the network):

$$\overline{A_{k=}}=\mathrm{E}\{A_k(t_0)\}=\sum_{n\geq 0} n.p(A_k(t_0)=n)=$$

$$=\sum_{n\geq 1}\frac{\left(\lambda_k/\mu_k\right)^n e^{-\lambda_k/\mu_k}}{(n-1)!}=e^{-\lambda_k/\mu_k}\left(\lambda_k/\mu_k\right)\sum_{n\geq 1}\frac{\left(\lambda_k/\mu_k\right)^{n-1}}{(n-1)!}=$$

$$=e^{-\lambda_k/\mu_k}\left(\lambda_k/\mu_k\right)e^{\lambda_k/\mu_k}=\lambda_k/\mu_k\ .$$

In general, querying nodes are not able to search directly in the backbone, and usually connect to at least one aggregation node in order to route their queries. The aggregation node concentrates all queries of its connected nodes and consults the backbone when it is not able to directly answer the queries received. One of the objectives of having aggregation nodes is to minimize the number of searches in the backbone; to do this, aggregation nodes maintain a cache of the results of recent queries, and are then also called cache nodes. The behavior of a cache node is very simple: when a query over content k arrives, if the answer is present in the cache it is returned; otherwise, the cache node starts a search in the backbone to obtain the information and answer the query; this information is then stored in the cache, for a prefixed time d_k, afterwards it expires.

One of the reasons for deleting out-dated information is that the results of a query will only be valid for a given time interval, as the nodes which hosted this content can disconnect or delete the content of interest, and new nodes can connect or start to publish the content. Suppose the cache node queried the backbone at time t_0 for a given content of class *k* and received in answer the information about $A_k(t_0)$ source nodes which hosted this content at that time. From then on, we can consider that the number of valid locations for this content known to the cache node will evolve like a stochastic pure-death process, with death parameter μ_k, as the source nodes will disconnect or delete the contents, until a new query is routed to the backbone.

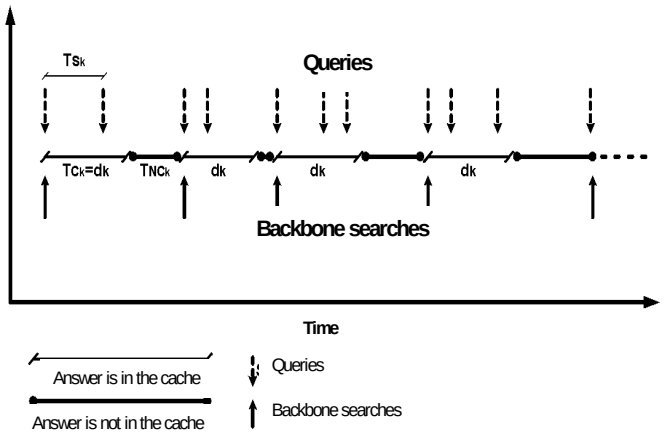

Figure 2 – cyclic behavior at cache nodes.

We can then compute the mean number of valid locations of a given content known by a cache node at time $t_0 + t$ when the last query answered by the backbone has been at time t_0 :

$$\begin{pmatrix}\text{mean number of valid content}\\ \text{locations } t \text{ time units after}\\ \text{the last backbone query}\end{pmatrix}=$$

$$=\sum_{n\geq 0} n.p(A_k(t_0)=n)p\left(T_{V_k}>t_0+t\middle|T_{V_k}>t_0\right)=$$

$$=\sum_{n\geq 0} n.p(A_k(t_0)=n)p\left(T_{V_k}>t\right)=\sum_{n\geq 1}\frac{\left(\lambda_k/\mu_k\right)^n e^{-\lambda_k/\mu_k}}{(n-1)!}e^{-\mu_k t}=$$

$$=e^{-\lambda_k/\mu_k}\left(\lambda_k/\mu_k\right)e^{-\mu_k t}\sum_{n\geq 1}\frac{\left(\lambda_k/\mu_k\right)^{n-1}}{(n-1)!}=$$

$$=e^{-\lambda_k/\mu_k}\left(\lambda_k/\mu_k\right)e^{-\mu_k t}e^{\lambda_k/\mu_k}=\lambda_k/\mu_k\ e^{-\mu_k t}.$$

The cache node follows a cyclic behaviour, starting with a first query of a content of class *k*, leading to a backbone search; following with a period of fixed duration d_k, where all queries arriving are answered using the information present in the cache; and then, after the expiration of the cache contents, a period of random duration, until a new query for this particular content arrives, re-starting the cycle again. By the hypothesis of Poisson arrivals for queries, this last period follows an exponential distribution of parameter f_k (the query frequency). Figure 2 shows a scheme of this cycle, where we denote by $T_{C_k}=d_k$ the period where the contents are cached, and by T_{NC_k} the period where the contents are not cached. The mean length

of the cycle is then $d_k + 1/f_k$; in each cycle there is only a single search in the backbone (when the cycle starts), this can be used to compute the rate of backbone searches as follows:

$$(\text{backbone searches per time unit}) = \frac{\#\,\text{searches}}{\text{total cycle time}} = \frac{1}{d_k + 1/f_k} = \frac{f_k}{1 + d_k f_k}.$$

As the query frequency is fixed externally, the only free variables we can adjust at cache nodes to define their behavior are the content expiration dates d_k for every content of class *k*.

2.2 Bandwidth constraints

Cache nodes have input and output bandwidth constraints, which can limit the number of queries they can receive, process, answer and eventually pass on to the backbone. We will try to formulate these constraints in terms of the previously defined parameters and of the free variables d_k .

We denote by BW_{IN} and BW_{OUT} the maximum input and output bandwidth a cache node is able to employ. We suppose that each query the cache nodes receive employs β_S bytes in mean and that its answer employs α_S bytes per location information to be sent (then, the answer varies in size depending the number of known node locations where a content is stored). We also use other additional parameters: β_B , the message size of queries to be sent to the backbone, and α_B the message size per location of the answers received from the backbone.

Then, the input bandwidth to be used by the cache node corresponds to the sum of the size of the queries received from the querying nodes (at a rate f_k per content of class *k*), and of the answers sent by the backbone when queried about a specific content. As we know that the backbone search frequency is $\frac{f_k}{1 + d_k f_k}$, and the mean number of backbone locations for each content of class *k* is $\overline{A_k} = \lambda_k / \mu_k$, taking into account that there l_k contents belonging to class k , we arrive to the following formula:

$$(\text{input bandwidth}) = \beta_S \sum_{k \in K} l_k f_k + \alpha_B \sum_{k \in K} \frac{l_k f_k}{1 + d_k f_k} \frac{\lambda_k}{\mu_k}..$$

Similarly, the output bandwidth corresponds to the sum of the queries transmitted to the backbone plus the content locations answered to the querying nodes in response to their queries, leading to the formulation

$$(\text{output bandwidth}) = \alpha_S \sum_{k \in C} l_k f_k \overline{A_k} + \beta_B \sum_{k \in C} \frac{l_k f_k}{1 + d_k f_k}.$$

We can then formulate the bandwidth constraints as follows:

$$\beta_S \sum_{k \in C} l_k f_k + \alpha_B \sum_{k \in C} \frac{l_k f_k}{1 + d_k f_k} \overline{A_k} \le BW_{IN},$$

$$\alpha_S \sum_{k \in C} l_k f_k \overline{A_k} + \beta_B \sum_{k \in C} \frac{l_k f_k}{1 + d_k f_k} \le BW_{OUT}.$$

2.3 Expected number of correct answers

The network primary objective is to be able to give the most complete and correct information to the queries received. To formalize this objective, we develop an expression for the number of correct answers (i.e., the number of valid content locations) answered to the querying nodes. In particular, if we denote by R_k the random variable corresponding to the number of content locations answered to a query for a content of class *k*, we want to compute its expected value $\overline{R_k}$. We know that during a cache node cycle, there will be at least one query (at the start of the cycle), and a random number of additional queries during the period where the content locations are stored in the cache, of duration d_k (as when the cache contents expire, the first query arriving will lead to the start of a new cycle). This leads for each content of class *k* to the following formulation:

$$\begin{aligned}
\overline{R_k} &= \mathrm{E}\{R_k\} = \\
&= \sum_{n \ge 0} \mathrm{E}\{R_k \mid n \text{ additional queries}\} p(n \text{ additional queries}) = \\
&= \mathrm{E}\{R_{k,NC}\} p(0 \text{ additional queries}) + \\
&\quad \sum_{n \ge 1} \left(\frac{\mathrm{E}\{R_{k,NC}\} + \sum_{m=1}^{n} \mathrm{E}\{R_{k,Cm} \mid n \text{ add. queries}\}}{n+1} \right) p(n \text{ add. queries}).
\end{aligned}$$

where $R_{k,NC}$ is the answer to the initial query (transmitted to the backbone, and whose answers are stored in the cache), and $R_{k,C1} ... R_{k,Cn}$ are the answers to the following queries during the time period starting with the first query and of duration d_k .

The expected number of correct responses to the first query is exactly the expected number of nodes hosting the contents, $\mathrm{E}\{R_{k,NC}\} = \overline{A_k} = \frac{\lambda_k}{\mu_k}$.

For the following queries, we use on one hand the fact that query arrivals follow a Poisson process of rate f_k , so that

the probability of observing n arrivals during a time interval of length d_k is:

$$p(S_k(d_k)=n)=\frac{(f_k d_k)^n e^{-f_k d_k}}{n!}\ \ \forall k\in C,\forall n\in\aleph,\forall d_k\in\Re^+.$$

On the other hand, it is a well-known fact (see for instance the discussion in [5]) that the distribution of the arrivals of a Poisson process within a fixed interval follows a uniform distribution. As a result, the expectation of the number of correct answers given to the queries during this interval will be equal to the mean number of valid content locations in the interval. As the number of valid locations known at time t after the last query is equal to $\frac{\lambda_k}{\mu_k}e^{-\mu_k t}$, then its mean over the interval of duration d_k is

$$\frac{\int_0^{d_k}\frac{\lambda_k}{\mu_k}e^{-\mu_k t}.dt}{d_k}=\frac{-\frac{\lambda_k}{{\mu_k}^2}e^{-\mu_k t}\Big|_0^{d_k}}{d_k}=$$
$$=\frac{\lambda_k}{{\mu_k}^2 d_k}\left(1-e^{-\mu_k d_k}\right)\quad .$$

Then we have that:

$$\sum_{m=1}^{n}\mathrm{E}\{R_{k,Cm}|n\text{ queries}\}=$$
$$=n\begin{pmatrix}\text{mean number of valid locations known to the}\\ \text{cache node in the time interval }(t_0,t_0+d_k]\end{pmatrix}=$$
$$=n\frac{\lambda_k}{{\mu_k}^2 d_k}\left(1-e^{-\mu_k d_k}\right)\ \ \forall k\in C.$$

Combining all these results, we find

$$\overline{R_k}=\mathrm{E}\{R_k\}=\sum_{n\ge0}\mathrm{E}\{R_k|n\text{ queries}\}p(n\text{ queries})=$$
$$=\mathrm{E}\{R_{k,NC}\}p(0\text{ queries})+$$
$$\sum_{n\ge1}\left(\frac{\mathrm{E}\{R_{k,NC}\}+\sum_{m=1}^{n}\mathrm{E}\{R_{k,Cm}|n\text{ queries}\}}{n+1}\right)p(n\text{ queries})=$$
$$=\sum_{n\ge0}\left(\frac{\frac{\lambda_k}{\mu_k}+n\frac{\lambda_k}{{\mu_k}^2 d_k}\left(1-e^{-\mu_k d_k}\right)}{n+1}\right)\frac{(f_k d_k)^n e^{-f_k d_k}}{n!}=$$
$$=\frac{\lambda_k}{\mu_k}e^{-f_k d_k}\sum_{n\ge0}\frac{(f_k d_k)^n}{(n+1)!}+\frac{\lambda_k}{{\mu_k}^2 d_k}e^{-f_k d_k}\left(1-e^{-\mu_k d_k}\right)\sum_{n\ge0}n\frac{(f_k d_k)^n}{(n+1)!}=$$
$$=\frac{\lambda_k}{\mu_k}e^{-f_k d_k}\frac{\left(e^{f_k d_k}-1\right)}{f_k d_k}+\frac{\lambda_k}{{\mu_k}^2 d_k}e^{-f_k d_k}\left(1-e^{-\mu_k d_k}\right)\left[e^{f_k d_k}-\frac{\left(e^{f_k d_k}-1\right)}{f_k d_k}\right]=$$
$$=\frac{\lambda_k}{{\mu_k}^2 f_k d_k}\left[\mu_k\left(1-e^{-f_k d_k}\right)+f_k\left(1-e^{-\mu_k d_k}\right)-\frac{1}{d_k}\left(1-e^{-f_k d_k}\right)\left(1-e^{-\mu_k d_k}\right)\right].$$

Finally, we can compute the expected number of correct answers taking into account all contents; this is the function we would like to maximize:

$$\sum_{k\in C}\overline{R_k}l_k f_k=$$
$$=\sum_{k\in C}\frac{l_k\lambda_k}{{\mu_k}^2 d_k}\left[\mu_k\left(1-e^{-f_k d_k}\right)+f_k\left(1-e^{-\mu_k d_k}\right)-\frac{1}{d_k}\left(1-e^{-f_k d_k}\right)\left(1-e^{-\mu_k d_k}\right)\right]$$

2.4 Mathematical programming formulation

If we put together the network objective and the bandwidth restrictions discussed in the previous section, we arrive to the following formulation of our CCP problem:

$$\max_{d_k\in\Re^+}\sum_{k\in K}\frac{l_k\lambda_k}{{\mu_k}^2 d_k}\left[\mu_k\left(1-e^{-f_k d_k}\right)+f_k\left(1-e^{-\mu_k d_k}\right)-\frac{1}{d_k}\left(1-e^{-f_k d_k}\right)\left(1-e^{-\mu_k d_k}\right)\right]$$

s.t.

$$\beta_S\sum_{k\in K}l_k f_k+\alpha_B\sum_{k\in K}\frac{l_k f_k}{1+d_k f_k}\frac{\lambda_k}{\mu_k}\le BW_{IN}$$

$$\alpha_S\sum_{k\in K}l_k f_k\frac{\lambda_k}{\mu_k}+\beta_B\sum_{k\in K}\frac{l_k f_k}{1+d_k f_k}\le BW_{OUT}$$

$d_k\in\Re^+$ decision variables, for $k\in K$

$l_k,f_k,\lambda_k,\mu_k,\alpha_S,\alpha_B,\beta_S,\beta_B,BW_{IN},BW_{OUT}\in\Re^+\ \ \forall k\in K$.

This is a non-linear optimization problem, both in the restrictions and in the objective function. If we study it in detail, we can see that both the feasible solution space and the objective function are convex. As the problem is stated as a maximization one, a convex objective function will in general lead to multiple local optima.

3. A CASE STUDY ILLUSTRATION

In order to apply framework that we have presented to a given network, it is necessary to obtain from measurements or previous studies the information about the parameters of the class contents. The case of the Gnutella peer-to-peer (P2P) file sharing network has been developed as part of the MSc. Thesis of one of the authors of the present paper [17]. In the following section, we look at the DNS (Domain Name System) [11][12], which is the system used on Internet in order to map symbolic domain names (such as

www.fing.edu.uy) into IP addresses corresponding to actual computers in the network (such as 164.73.32.3). The network actually modeled corresponds to the .uy (Uruguay) subdomain of Internet, and the parameters used are based on real data obtained thanks to the support of ANTEL (Administración Nacional de Telecomunicaciones, www.antel.com.uy), which is a state-owned company, and the largest telco in Uruguay.

3.1 Parameters for the DNS network.

DNS (Domain Name System) can be seen as a content network with hierarchical distribution, where the information consistency is one of the most important objectives. DNS is based on a network of recursive servers, which pass on queries until finding authoritative answers, which minimize the probability of information inconsistency (even if the correctness cannot be completely guaranteed).

In our case, we are interested in recursive servers, which correspond to the aggregation nodes of our general model. We collected real data from the recursive servers at ANTEL, which daily serve hundreds of thousands users, with daily peak query rates of approximately 1800 queries per second. The parameters of the model are summarized in Table 1.

Parameters	Values
C : number of different contents	220107
$\overline{f}$: average content query rate	Empirical distribution (heavy tailed, see Figure 3)
$\overline{\lambda}$: average content storage rate	0.8361 hr^{-1}
$\overline{\mu}$: average content location validity rate	0.5158 hr^{-1}
α_S : size of a the answer to a content query	169.6 bytes
α_B : size of the answer of a backbone search	1150.3 bytes
β_S : size of a content query	80.45 bytes
β_B : size of a backbone search packet	385.6 bytes
BW_{IN} : input bandwidth	1.933×10^8 bytes/hr.
BW_{OUT} : output bandwidth	3.445×10^8 bytes/hr.

Table 1: parameter values for the case study.

As previously discussed, in our study we only took into account the behavior of the Uruguayan domains, i.e., those whose names finish by `.uy`.

We collected 10 consecutive days of recursive server logs, in order to estimate the total number of contents and the distribution of query rates. ANTEL DNS infrastructure employs the BIND software (Berkeley Internet name domain, by Internet Software Consortium, http://www.isc.org/products/BIND), which was very useful as the logs it collects contain very detailed information. The log files were processed using a BerkeleyDB data base (Sleepycat Software Inc.. http://www.sleepycat.com/) to obtain the statistics of the domain queries.

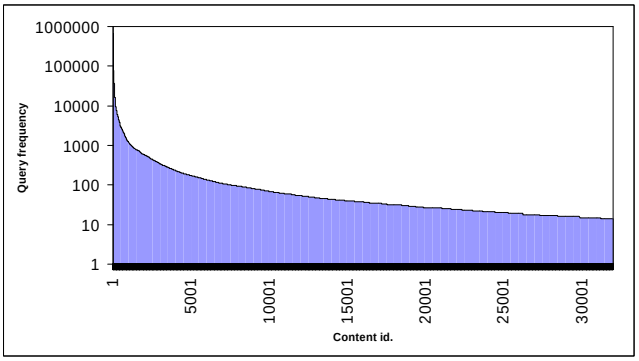

Figure 3: domain query rates distribution (tail cut off).

The largest part of the `.uy` domains actually belong to the `.com.uy` zone, which is administered by ANTEL. We took the historical information of the domain changes between October 2003 and October 2005; this information was used to compute overall storage and validity rates for the contents.

To compute the mean packet sizes, we also collected information about the packets transmitted and received by a DNS recursive server (in this case, a 15 minutes detailed sample provided us with enough information). Table 2 shows the measurements, which were the basis for computing the Alfa and Beta parameters shown on Table 1

	Number	Total bytes transmitted.	Number of registers
Queries	366148	29458160	402098
Answer to queries	168530	40089683	236386
Backbone searches	61481	5594012	293552
Answers to backbone searches	56442	10774381	293552

Table 2: statistics for packet sizes.

Finally, in the case of ANTEL recursive servers, the bandwidth limitation is actually driven by the CPU processing power, which limits the number of queries that may be processed by time unit.

In our case, the empirical measurements result in $BW_{IN} = 419.5\,\text{kbps} = 193305600\,\text{bytes/hr}$ and $BW_{OUT} = 747.6\,\text{kbps} = 344494080\,\text{bytes/hr}$.

3.2 Numerical results.

The data collection discussed in the previous section resulted in obtaining detailed query rate information for each of the 220107 contents observed. A possibility would be to create a content class of cardinality 1 for each different content, this would lead to a large non-linear problem in 220107 independent variables. As an alternative, we cluster the contents into a small number of content classes, where in each class we will include contents with equal or at least similar query rates. As it is not a-priori clear what is the best number of classes to use, we experimented with five different values, namely 2, 8, 16, 32, and 128 classes.

In order to solve the different problems formulated, we used AMPL [2], an algebraic modeling language for mathematical programming problems, in conjunction with MINOS (version 5.5) [13], an optimization solver. In the Appendix, we give some examples of how the optimization problem is formulated in terms of the AMPL language, and of the commands to be used to find a numerical solution.

All experiments were run on a PIII 800 MHz computer, with 320 Mb RAM space. The results obtained are summarized in Table 3. The objective function has been normalized, using a tight upper bound, so that the values can be compared directly. Among other observations, we can see that when the number of classes grows, the available resources are being increasingly used. Also, the computational times required to solve the model grow, albeit they remain very modest.

Number of content classes	Normalized Objective function	Execution time (secs.)	Input bandwidth employed (bytes/hr)	Output bandwidth employed (bytes/hr)
2	0. 969623	0.000	193305400	77163700
8	0. 982216	0.020	193305400	77163700
16	0. 997218	0.060	193305400	77163700
32	0. 999742	0.120	193305400	77163700
128	0. 999919	0.347	193305400	77163700

Table 3: Results for different number of content classes.

As we are dealing with aggregated data, it is important to translate back the results into the terms of the original problem. In particular, we now consider again the 220107 different contents, and we evaluate the number of correct answers to queries if we use for each content the cache expiration times given by the optimization models. Table 4 summarizes this comparison.

Number of classes	Normalized objective function for the aggregated problem	Normalized objective function for the original problem
2	0.96962300	0.88249787
8	0.98221600	0.95332949
16	0.99721800	0.99412569
32	0.99974200	0.99966787
128	0.99991900	0.99991900

Table 4: Discrepancies between objective functions for original and aggregated models.

From this table, we can see that if the number of classes is too low, then the approximation error incurred in the aggregated model is very large, and the percentage of correct answers to queries in the real problem will be much below the nominal values computed by the optimization procedure. This discrepancy gets very quickly irrelevant when the number of classes increase, when we have 128 classes the results coincide.

4. CONCLUSIONS

This paper discusses the impact that cache expiration times have on the behavior of aggregation nodes in a content network. In particular, we present a simplified model to evaluate the total number of correct answers given to content queries, and to evaluate the bandwidth usage. On the basis of this simplified model, we present a mathematical programming formulation, which allows to find optimal values for the cache expiration times in order to maximize the number of correct answers, subject to bandwidth limitations. We have also studied as a particular case the DNS system, in particular for the case of the .uy Internet domain; a comprehensive data collection program has allowed us to obtain the numerical parameters needed to instantiate the optimization model and obtain the corresponding results for the cache expiration dates. The results show that the computational requirements are modest, and that using data with relatively high aggregation we can obtain high performance levels. We think that models of this kind lead to improved understanding of the behavior of content networks, and can be used to test their performance in a wide variety of potential scenarios, which are difficult to test in practice.

Future work includes using the model with test cases corresponding to additional content networks of different characteristics. It is also possible to refine the model to take into account additional features (for example, the search answer packet sizes could be divided into a fixed part plus

a variable, per location answered, part; additional constraints could be added to represent particular features of specific networks). Another interesting point is doing a more detailed analysis of the impact of the number of content classes chosen on the quality of the results obtained, as well as on the computational requirements imposed by the solution methods. Finally, a more difficult challenge is to integrate backbone behavior details into this model, in order to have a more wide perspective on the tradeoffs between information publication and search in a content network.

5. ACKNOWLEDGMENTS

This research has been partially supported by CNPq, Project PROSUL- Proc. no. 490333/04-4, by INRIA associated teams program (team PAIR), and by MEC/BID PDT program, Projects S/C/IF/29/37 and S/C/OP/17/03. We also gratefully acknowledge the support of ANTEL for the DNS data collection exercise.

6. REFERENCES

[1] Chu J., Labonte K., and Levine, B., "Availability and locality measurements of peer-to-peer file systems," in ITCom: Scalability and Traffic Control in IP Networks. *Proceedings of SPIE*, Vol. 4868, July 2002.

[2] Fourer, R., Gay, D.M, and. Kernighan, B.W. *AMPL: A Modeling Language for Mathematical Programming.* Duxbury Press / Brooks/Cole Publishing Company, 2002.

[3] Ge, Z., Figueiredo D., Jaiswal S., Kurose J., Towsley D. Z. Ge, D. R. Figueiredo, S. Jaiswal, J. Kurose, and D. Towsley, "Modeling Peer-Peer File Sharing Systems", Proc. of 22nd IEEE Infocom, 2003.

[4] Gummadi K., Dunn R., Saroiu S., Gribble S., Levy H., and Zahorjan J. Measurement, Modeling and Analysis of a Peer-to-Peer File-Sharing Workload. *Proceedings of the 19th ACM Symposium of Operating Systems Principles (SOSP)*, Bolton Landing, NY, October 2003.

[5] ITC, in cooperation with ITU-D SG2. *Teletraffic Engineering Handbook*, Draft-version. www.tele.dtu.dk/teletraffic (homepage maintained by V. B. Iversen; Last accessed 26 May 2005).

[6] Jovanovic, M., *Scalability Issues in Large Peer-to-Peer Networks - A Case Study of Gnutella*. University of Cincinnati Technical Report 2001. Available at http://www.ececs.uc.edu/~mjovanov/Research/paper.html.

[7] H. T. Kung, et al. *MotusNet: A Content Network.* Technical report. Harvard University. 2001. http://citeseer.nj.nec.com/443175.html.

[8] Kung, H. T., and Wu, C. H. (2002). Content Networks: Taxonomy and New Approaches. Chapter in *The Internet as a Large-Scale Complex System*, Kihong Park and Walter Willinger (Editors), Oxford University Press. 2002.

[9] Lv Q., Cao P., Cohen E., Li K., and Shenker S.. Search and replication in unstructured peer-to-peer networks. In Proceedings of the 16th annual ACM International Conference on supercomputing, 2002.

[10] D. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, Z. Xu. *Peer-to-Peer Computing*. Techical report HPL-2002-57, HP Labs. 2002. http://www.hpl.hp.com/techreports/2002/HPL-2002-57.html.

[11] Mockapetris, P.V. Internet Domain Name System Standard: Domain names - concepts and facilities. Rfc-Editor Home Page, ftp://ftp.rfc-editor.org/in-notes/rfc1034.txt, 1987.

[12] Mockapetris, P.V. Internet Domain Name System Standard: Domain names - implementation and specification. Rfc-Editor Home Page, ftp://ftp.rfc-editor.org/in-notes/rfc1035.txt, 1987.

[13] Murtagh, B. A. and Saunders, M. A. *MINOS 5.4 User's Guide*, Report SOL 83-20R, Systems Optimization Laboratory, Stanford University, December 1983 (revised February 1995).

[14] Pandurangan, G., Raghavan, P., and Upfal, E. Building Low-Diameter P2P Networks. In *Proceedings of the 42nd Annual IEEE Symposium on the Foundations of Computer Science (FOCS)* (2001).

[15] Qiu, L., Padmanabham, V. N. , and Voelker, G. M. On the placement of web server replicas. In Proc. 20th IEEE INFOCOM, 2001.

[16] Ripeanu M., Foster I., and Iamnitchi A., Mapping the Gnutella network: Properties of largescale peer-to-peer systems and implications for system design, *IEEE Internet Computing Journal* 6(1), 2002.

[17] Rodríguez-Bocca, P. *Redes de Contenido: Taxonomía y Modelos de evaluación y diseño de los mecanismos de descubrimiento de contenido.* MSc. Thesis, Facultad de Ingenieria, Universidad de la República, Uruguay.

[18] Saroiu, S., Krishna Gummadi, P., and Gribble, S.D. A Measurement Study of Peer-to-Peer File Sharing Systems. In *Proceedings of Multimedia Computing and Networking*, 2002

[19] Sen, S. and Wong, J. Analyzing peer-to-peer traffic across large networks. http://citeseer.nj.nec.com/sen02analyzing.html

[20] Yang, B. and Garcia-Molina, H. Comparing hybrid peer-to-peer systems. In *Proceedings of VLDB'2001*.

[21]Yang B. and Garcia-Molina H. *Designing a super-peer network.*. 5.Technical Report, Stanford University, February 2002. http://dbpubs. stanford.edu:8090/pub/2002-13

[22] Yang, B. and Garcia-Molina, H. Designing a Super-Peer Network. *Proc. of the 19th Intl. Conf. on Data Engineering*, 2003.

[23] Zeinalipour-Yazti, D. and Folias, T.. *A Quantitative Analysis of the Gnutella Network Traffic*.Technical report, Department of Computer Science University of California - Riverside, CA 92507, USA http://www.cs.ucr.edu/~csyiazti/cs204.html

7. APPENDIX: AMPL code examples

We give here more information regarding the AMPL code used for modeling and solving the problem instances. Figure 4 corresponds to the model. Figure 4 contains the AMPL commands used to solve the problem. Figure 6 shows the detailed data corresponding to the 2 class instance.

```
param K >=0, integer;
set CLASS = {1..K};
param f {k in CLASS};
param lamda {k in CLASS};
param mu {k in CLASS};
param l {k in CLASS};
param alphaS      >=0;
param alphaB      >=0;
param betaS       >=0;
param betaB       >=0;
param BWin        >=0;
param BWout       >=0;

var d {k in CLASS} >=0.000001 default 0.000001;

maximize epsilon:
          (sum {k in CLASS}
l[k]*lamda[k]/mu[k]/mu[k]/d[k]*(
                    mu[k]*(1-exp(-f[k]*d[k])) +
                    f[k]*(1-exp(-mu[k]*d[k])) -
                    1/d[k]*(1-exp(-f[k]*d[k]))*
                        (1-exp(-mu[k]*d[k])))/
          (sum {k in CLASS} l[k]*lamda[k]/mu[k]*f[k]);

subject to bitsIn :
 0 <= betaS*(sum {k in CLASS} l[k]*f[k]) +
 alphaB*(sum {k in CLASS}
l[k]*lamda[k]/mu[k]*f[k]/(1+d[k]*f[k]))
   <= BWin;

subject to bitsOut:
 0 <= alphaS*(sum {k in CLASS}
l[k]*lamda[k]/mu[k]*f[k]) +
           betaB*(sum {k in CLASS}
l[k]*f[k]/(1+d[k]*f[k]))
   <= BWout;
```

Figure 4: AMPL model

```
option ampl_include '.';

option solver minos;
option minos_options 'crash_option=0 \
 feasibility_tolerance=1.0e-8 scale=no \
 summary_file=6 summary_frequency=5 \
 timing= 1';

model cccp.mod;
data cccp.dat;
solve;
display epsilon;
display bitsIn.lb, bitsIn.ub, bitsIn.body, bitsIn.slack;
display bitsOut.lb, bitsOut.ub, bitsOut.body,
bitsOut.slack;
display d;
expand bitsIn, bitsOut;
```

Figure 5: AMPL commands

```
param K       := 2;

param alphaS   := 169.59400000;
param alphaB   := 1150.25200000;
param betaS    := 80.45400000;
param betaB    := 385.55700000;
param BWin     := 193305600.00000000;
param BWout    := 344494080.00000000;

param f :=
       1 1059.31197749
       2 0.32706349;
param lamda :=
       1 0.83609821
       2 0.83609821;
param mu :=
       1 0.51581508
       2 0.51581509;
param l :=
       1 68.00000000
       2 220039.00000000;
```

Figure 6: Detailed data for the two-class instance