PEDECIBA Informática
Instituto de Computación – Facultad de Ingeniería
Universidad de la República
Montevideo, Uruguay

Reporte Técnico RT 05-07

# A mathematical programming formulation of optimal cache expiration dates in content networks

Héctor Cancela Pablo Rodríguez Boca

2005

A mathematical programming formulation of optimal cache expiration dates in content networks
Héctor Cancela Bosi; Pablo Rodríguez Boca

ISSN 0797-6410
Reporte Técnico RT 05-07
PEDECIBA
Instituto de Computación – Facultad de Ingeniería
Universidad de la República
Montevideo, Uruguay, 2005

# A mathematical programming formulation of optimal cache expiration dates in content networks

Héctor Cancela
cancela@fing.edu.uy
Departamento de Investigación Operativa
Instituto de Computación
Facultad de Ingeniería
Universidad de la República, Uruguay

Pablo Rodríguez Bocca
prbocca@fing.edu.uy
Departamento de Investigación Operativa
Instituto de Computación
Facultad de Ingeniería
Universidad de la República, Uruguay

**ABSTRACT**
One of the fundamental decisions in content networks is how the information about the existing contents is deployed and accessed. In particular, there are two main alternatives, either to publish the information when contents are changed, or to search for the contents when a query is received. Even if some networks only use one of these alternatives, in general it is better to employ a mix of both strategies. This implies evaluating the tradeoff between these alternatives, in order to decide the characteristics of the mix. In this work we develop a simplified model of the costs and restrictions associated with cache expiration dates in a cache node in a content network; these expiration dates regulate the proportion of queries which will be answered on the basis of published information, vs. those which will give rise to additional searches in the network backbone. Based on this model, we present 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.

**Keywords**
Peer-to-peer networks, mathematical programming, optimization.

**RESUMEN**
En las redes de contenido, uno de los puntos fundamentales es la decisión sobre cómo acceder y distribuir la información sobre los contenidos existentes. En particular, hay dos alternativas principales, publicar la información cuando los contenidos cambian, o buscar los contenidos cuando se recibe una consulta. En general, se emplea una combinación de ambas alternativas, debiéndose evaluar la mejor manera de realizar la misma. En este trabajo, desarrollamos un modelo simplificado de los costos y restricciones asociados con las fechas de expiración de cache en nodos cache. Estas fechas regulan la proporción de consultas que serán contestadas en base a la información publicada, y aquellas que darán lugar a una búsqueda en el backbone. Basados en este modelo, presentamos una formulación de programación matemática que puede ser empleada para determinar las fechas de expiración óptimas de manera de maximizar el total de información encontrada, respetando las restricciones operacionales (de ancho de banda disponible en los nodos cache).

**Palabras clave**

Redes de pares, programación matemática, optimización.

## 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 [12][13][14].

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.

There are two main strategies to discover the meta-information, namely publication and search. By publication we mean the process by which a network node unrequestedly sends meta-information it possesses to the remaining nodes. By search we mean the process by which a node asks the remaining ones to send it the meta-information they possess. By analogy with logistics, we can say that publication is an "information push" strategy, and search an "information pull" strategy.

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. Both publishing and search can contribute towards this task, but their relative efficiency varies, so that there is a tradeoff between their frequency and modality of application. 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][14][15][16][20], 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][12][13][17][18].

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; this will be our Content Caching Problem (CCP).

### 2.1 Network components description

We will look at the content network as composed of source nodes and querying nodes (which may be the same), of cache nodes (also called aggregation nodes), and of a backbone (which will not be further modeled); a graphical representation can be seen in Figure 1. This division is actually virtual, as a same physical node may act at the same time as a source node, a querying node, a cache node, and a backbone node. We will also separately model the contents of the network (which will belong to a set C). The content network is considered to be in steady state, so that we will not need to explicitly model the time; this assumption is justified by the fact that the time rate at which contents appear and disappear, and cache expiration times, are usually much faster than the times by which the statistical properties of the user population change.

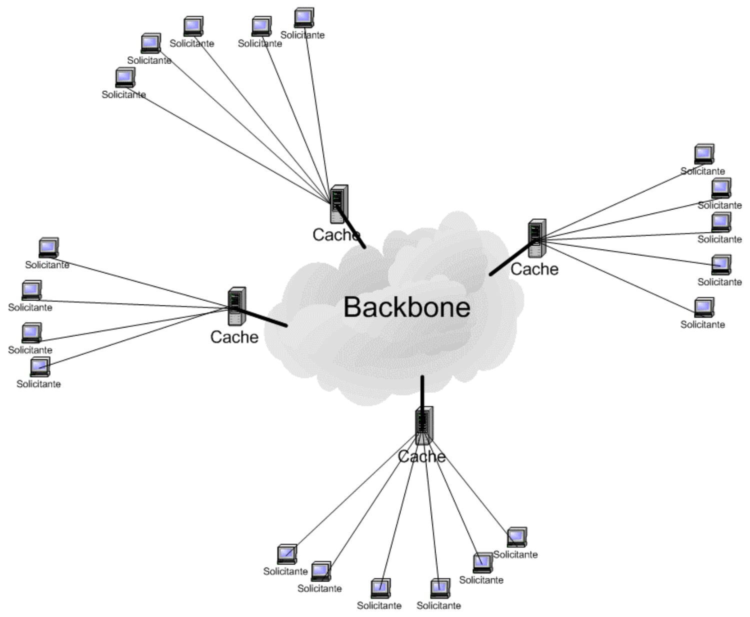


**Figure 1: Simplified view of a content network**

The users of the network will query about each content k with a different 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 content 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, will be an exponentially distributed random variable with parameter $f_k$:

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

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 k, which we will suppose will be of $M/M/\infty$ type and parameters $\lambda_k$ and $\mu_k$ (respectively, the rates of start and end of lodgement of content 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:

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

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):

$$\overline{A_k} = \mathrm{E}\left\{A_k(t_0)\right\} = \sum_{n \ge 0} n.p\left(A_k(t_0) = n\right) = \sum_{n \ge 1} \frac{\left(\lambda_k/\mu_k\right)^n e^{-\lambda_k/\mu_k}}{(n-1)!} =$$

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

The only routing nodes we will consider are aggregation nodes. 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 content 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 content k known to the cache node will evolve like a stochastic pure-death process, with death parameter $\mu_k$, as the source nodes will disconnect or delete the contents, until a new query is routed to the backbone.

We can then compute the mean number of valid locations 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 behavior of a cache node is then essentially composed of a repeated cycle, which starts with a first query of content *k*, leading to a backbone search; then a period of fixed duration $d_k$, where all queries arriving are answered with the information contained in the cache memory; and then, after the expiration of the cache contents, a period of random duration, until a new query for content *k* arrives, re-starting all 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 the period where the contents are cached, and by the period where the contents are not cached. $T_{C_k} = d_k \, T_{NC_k}$ 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:

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

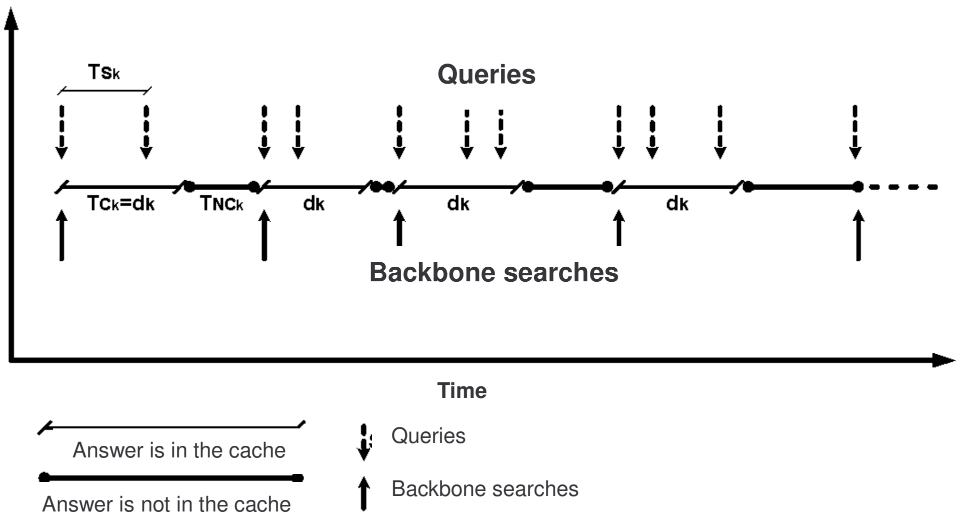


**Figure 2 – cyclic behavior at cache nodes.**

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 *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 receives employs $\beta_S$ bytes in mean, and that its answer employs $\alpha_S$ 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 as additional parameters $\beta_B$ ,the message size of queries to be sent to the backbone, and $\alpha_B$ which is 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 *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 content *k* locations in the backbone is $\overline{A_k} = \lambda_k / \mu_k$, we arrive to the following formula: $(\text{input bandwidth}) = \beta_S \sum_{k\in C} f_k + \alpha_B \sum_{k\in C} \frac{f_k}{1+d_k f_k} \overline{A_k}$.

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

We can then mathematically formulate the bandwidth constraints as follows:

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

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

## 2.3 Expected number of correct answers

The network primary objective is to be able to give the most complete 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 content *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 to the following formulation for each content *k*:

$$\overline{R_k} = \mathrm{E}\{R_k\} = \sum_{n\geq 0} \mathrm{E}\{R_k | n \text{ additional queries}\} p(n \text{ additional queries}) =$$

$$= \mathrm{E}\{R_{k,NC}\} p(0 \text{ additional queries}) + \sum_{n\geq 1} \left( \frac{\mathrm{E}\{R_{k,NC}\} + \sum_{m=1}^{n} \mathrm{E}\{R_{k,Cm} | n \text{ add. queries}\}}{n+1} \right) p(n \text{ additional queries}).$$

where $R_{k,NC}$ is the answer to the inicial query (transmitted to the backbone, and whose answers are stored in the cache), and $R_{k,C1} \ldots R_{k,Cn}$ 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, $E\{R_{k,NC}\} = \overline{A_k} = \frac{\lambda_k}{\mu_k}$.

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 :

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

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 follow an uniform distribution. This means that the expected mean value of the number of answers received to the queries during this interval will be equal to the expected value of valid content locations in the interval (i.e, the expectation over the queries will be equal to the expectation over the time interval, a PASTA – Poisson Arrivals See Time Averages result). As the number of valid know locations known at time *t* after the last query is equal to $\frac{\lambda_k}{\mu_k} e^{-\mu_k t}$, then its expectation over the interval of duration $d_k$ is

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

Then we have that:

.

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

Combining all these results, we find

$$\overline{R_k} = \mathrm{E}\{R_k\} = \sum_{n\geq 0} \mathrm{E}\{R_k | n \text{ queries}\} p(n \text{ queries}) =$$

$$= \mathrm{E}\{R_{k,NC}\} p(0 \text{ queries}) + \sum_{n\geq 1} \left( \frac{\mathrm{E}\{R_{k,NC}\} + \sum_{m=1}^{n} \mathrm{E}\{R_{k,Cm} | n \text{ queries}\}}{n+1} \right) p(n \text{ queries}) =$$

$$= \sum_{n\geq 0} \left( \frac{\frac{\lambda_k}{\mu_k} + n \frac{\lambda_k}{\mu_k^2 d_k}\left(1 - e^{-\mu_k d_k}\right)}{n+1} \right) \frac{(f_k d_k)^n e^{-f_k d_k}}{n!} =$$

$$= \frac{\lambda_k}{\mu_k} e^{-f_k d_k} \sum_{n\geq 0} \frac{(f_k d_k)^n}{(n+1)!} + \frac{\lambda_k}{\mu_k^2 d_k} e^{-f_k d_k} \left(1 - e^{-\mu_k d_k}\right) \sum_{n\geq 0} n \frac{(f_k d_k)^n}{(n+1)!} =$$

$$= \frac{\lambda_k}{\mu_k} e^{-f_k d_k} \frac{\left(e^{f_k d_k} - 1\right)}{f_k d_k} + \frac{\lambda_k}{\mu_k^2 d_k} e^{-f_k d_k} \left(1 - e^{-\mu_k d_k}\right) \left[ e^{f_k d_k} - \frac{\left(e^{f_k d_k} - 1\right)}{f_k d_k} \right] =$$

$$= \frac{\lambda_k}{\mu_k^2 f_k d_k} \left[ \mu_k \left(1 - e^{-f_k d_k}\right) + f_k \left(1 - e^{-\mu_k d_k}\right) - \frac{1}{d_k} \left(1 - e^{-f_k d_k}\right)\left(1 - e^{-\mu_k d_k}\right) \right].$$

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

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

## 3. 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:

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

s.t.

$$\beta_S \sum_{k\in C} f_k + \alpha_B \sum_{k\in C} \frac{f_k}{1 + d_k f_k} \frac{\lambda_k}{\mu_k} \leq BW_{IN}$$

$$\alpha_S \sum_{k\in C} f_k \frac{\lambda_k}{\mu_k} + \beta_B \sum_{k\in C} \frac{f_k}{1 + d_k f_k} \leq BW_{OUT}$$

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

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

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.

## 3.1 Content class based alternative formulation.

In most cases, content networks manage a very large number of different contents. These means that the previous formulation will have a large class of decision variables $d_k$, an additional difficulty for the numerical solution of the problem. On the other hand, for simplicity design reasons, the networks will in general treat in the same way contents that have similar characteristics. It is then possible to group all contents in a certain number of content classes, such that all contents within a class have relatively homogenous characteristics.

Formalizing,, we suppose that all contents $c \in C$ are grouped into $K$ content classes, such that if two contents belong to the same class, all their parameters are identical:

$$C = C_1 \cup C_2 \ldots \cup C_K$$

$$\left.\begin{array}{l} \forall c_i, c_j \in C_k \\ \forall k \in [1..K] \end{array}\right\} \Rightarrow \begin{cases} f_{c_i} = f_{c_j} \\ \lambda_{c_i} = \lambda_{c_j} \\ \mu_{c_i} = \mu_{c_j} \end{cases}$$

The size of class $k$, denoted by $l_k$, is 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 now define the Content Class Caching Problem. We enumerate the parameters of the problem (and give their dimensional units between brackets):

- $l_k$: number of contents belonging to class $k$ ($[l_k] = 1$).
- $f_k$: query rate for class $k$ contents $k$ ($[\mathrm{f_k}] = 1/\mathrm{sec.}$).
- $\lambda_k$: rate for source arrival for class $k$ contents ($[\lambda_k] = 1/\mathrm{sec.}$) .
- $\mu_k$: rate for content deletion in sources for class $k$ contents. ($[\mu_k] = 1/\mathrm{sec.}$) .
- $\alpha_S$: size per location answered in response to a content query ($[\alpha_S] = bytes$).
- $\alpha_B$: size per location answered in response to a backbone search ($[\alpha_B] = bytes$).
- $\beta_S$: size of a content query packet ($[\beta_S] = bytes$).
- $\beta_B$: size of a backbone search packet ($[\beta_B] = bytes$).
- $BW_{IN}$, $BW_{OUT}$: input and output bandwidth restrictions in the cache node ($[BW_{IN}] = [BW_{OUT}] = bytes/\mathrm{sec.}$).
- $d_k$: cache expiration times for class $k$ contents. ($[d_k] = \mathrm{sec.}$)

The problem is then formalized as follows:

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

subject to :

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

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

$d_k \in \Re^+ \ \forall k \in K$ decision variables

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

## 4. NUMERICAL ILLUSTRATION

### 4.1 Content caching problem case study.

In this section we present a numerical illustration over a case study, where the data was generated with information available in different literature sources especially referring to Gnutella or similar peer-to-peer (P2P) file sharing networks [1][4][9][15][19]. We have chosen P2P networks which are a specially successful category of content networks, for which there is also much quantitative information available.

| Parameters | Values |
|---|---|
| $T$ : time units | 1 hour |
| $C$ : number of different contents | 878691 |
| $\overline{f}$ : average content query rate | 0.037938251 hr$^{-1}$ |
| $f_{max}$ : maximum content query rate | 1000 hr$^{-1}$ |
| $\overline{\lambda}$ : average content storage rate | 11.09749966 hr$^{-1}$ |
| $\overline{\mu}$ : average content location validity rate | 1 hr$^{-1}$ |
| $\left(\frac{\lambda}{\mu}\right)_{max}$ : maximum allowed number of locations answered in response to a content query | 200 |
| $\alpha_S$ : size of a the answer to a content query | 100 bytes |
| $\alpha_B$ : size of the answer of a backbone search | 310 bytes |
| $\beta_S$ : size of a content query | 94 bytes |
| $\beta_B$ : size of a backbone search packet | 291.4 bytes |
| $BW_{IN}$ : input bandwidth | 921600000 bytes/hr. |
| $BW_{OUT}$ : output bandwidth | 460800000 bytes/hr. |

**Table 1: parameter values for the case study.**

Table 1 summarizes the main parameters of the case study. We generated ten instances of this detailed case study (using a random number generator with different seeds), including the data for the 878691 different contents (which correspond to the number of Gnutella contents in the study by Chu [1] ), where the distributions for the query frequency follow a modified Pareto distribution law taking into account the "fetch-at-most-once" effect (see [4] for a discussion of this observed network behavior). Regarding the frequency of arrival of new storage locations for each content, we suppose that it is linearly related to the query frequency, following the hypothesis mostly used in the literature (an exception is the work by Qin [9] which also studies a square root dependency). For the bandwidth constraints, we suppose that the cache nodes will be equipped with an ADSL 2/1 Mbps connection as reference value.

### 4.2 Class content caching problem case study.

As we discussed in Section 3, it is next to impossible to directly solve the CCP problem generated, a non-linear problem in 878691 independent variables (one for each content). As an alternative, we cluster the contents into a small number of homogeneous content classes, and solve the resulting CCCP problem. 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, for each of the 10 different CCP problems generated.

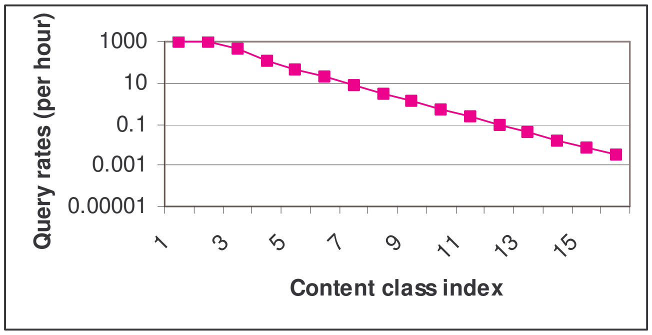
In order to solve the different problems formulated, we used AMPL, an algebraic modeling language for mathematical programming problems, in conjunction with MINOS (version 5.5), an optimization solver. In the Appendix, we give some examples of how the optimization problem is formulated in terms of the 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 2. 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 grow, 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 | 0.999379 | 0.001 | 823707506 | 298854619 |
| 8 | 0.991449 | 0.031 | 865591906 | 371027243 |
| 16 | 0.990825 | 0.201 | 870184506 | 377776027 |
| 32 | 0.997348 | 0.197 | 871334506 | 379988698 |
| 128 | 0.999432 | 0.347 | 871674006 | 380579793 |

**Table 2: Average results for 10 (randomly generated) cases.**

We have also looked in detail at the solutions given by the optimization model. As a representative, we can look at the results of one of the instances of the 16 class CCCP model. In Figure 3, we can see on the left the distribution (in logarithmic scale) of the query rates for the different content classes; the difference between query rates go across 6 magnitude orders. On the left, we can see (also in logarithmic scale) the results of the optimization, namely the values of the cache expiration dates for each of the 16 content classes. It is clear that, although here we can also appreciate wide differences in scale, there is no direct relation with the input data shown on the left.



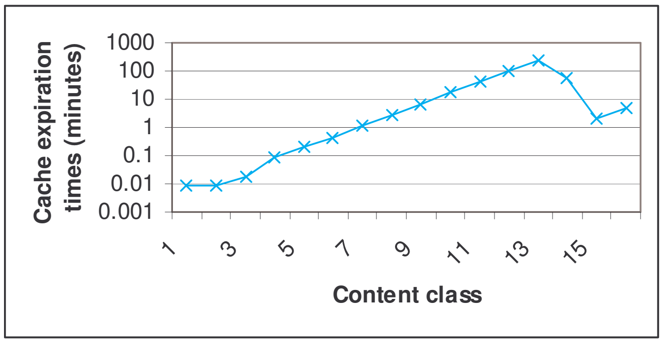


**Figure 3 - input data and output results for a 16 class CCCP instance.**

## 5. CONCLUSIONS

In this paper, we have developed a model of the impact that cache expiration times have on the total number of correct answers to queries in a content network, and on the bandwidth usage. This model has been used to develop 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. In order to cope with the explosion of free variables, we have also developed an alternative formulation based on treating identically groups of similar contents. To show the feasibility of employing the mathematical programming formulation, we used a set of test cases generated randomly in such a way that they comply with previously published information about existing networks. The results show that the computational requirements are modest, and that the model results can lead to non-intuitive solutions giving 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 could include using the model with test cases corresponding to content network of different characteristics (although the model is generic, the test data shown in this paper corresponds to a peer-to-peer file sharing network,). 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.

## 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] 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).

[12] 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).

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

[14] 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.

[15] 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

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

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

[18]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

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

[20] 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 CCCP problem instances. Figure 4 corresponds to the CCCP model. Figure 5 contains the AMPL commands used to solve the problem. Figure 6 shows the detailed data corresponding to one of the 8 class instances (generated with seed 1).

```
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 for CCCP problem**

```
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 for solving the CCCP problem**

```
param K        := 8;

param alphaS   := 100.00000000;
param alphaB   := 310.00000000;
param betaS    := 94.00000000;
param betaB    := 291.40000000;
param BWin     := 921600000.00000000;
param BWout    := 460800000.00000000;

param f :=
        1 1000.00000000
        2 216.15859672
        3 32.96348528
        4 5.12742726
        5 0.81727228
        6 0.14024312
        7 0.02464053
        8 0.00447485;
param lamda :=
        1 200.00000000
        2 200.00000000
        3 200.00000000
        4 200.00000000
        5 171.38934707
        6 41.02318819
        7 7.20772083
        8 1.30896076;
param mu :=
        1 1.00000000
        2 1.00000000
        3 1.00000000
        4 1.00000000
        5 1.00000000
        6 1.00000000
        7 1.00000000
        8 1.00000000;
param l :=
        1 4.00000000
        2 15.00000000
        3 98.00000000
        4 629.00000000
        5 3944.00000000
        6 22981.00000000
        7 130797.00000000
        8 720223.00000000;
```

**Figure 6: Detailed data for one of the 8-class instances of CCCP**