Performance Analysis of a Caching Algorithm for a Catch-Up Television Service

Z. Avramova*, D. De Vleeschauwer**, S. Wittevrongel*, H. Bruneel*

* SMACS Research Group, TELIN, Faculty of Engineering, Ghent University, Ghent, Belgium.
** Bell Labs, Alcatel-Lucent Bell, Antwerp, Belgium.
{kayzlat, sw, hb}@ telin.ugent.be, danny.de_vleeschauwer@alcatel-lucent.be

ABSTRACT

The Catch-Up TV (CUTV) service allows users to watch video content that was previously broadcast live on TV channels and later placed on an on-line video store. Upon a request from a user to watch a recently missed episode of his/her favourite TV series, the content is streamed from the video server to the customer's receiver device. This requires that an individual flow is set up for the duration of the video, and since it is hard to impossible to employ multicast streaming for this purpose (as users seldomly issue a request for the same episode at the same time), these flows are unicast.

In this paper we demonstrate that with the growing popularity of the CUTV service, the number of simultaneously running unicast flows on the aggregation parts of the network threaten to lead to an unwieldy increase in required bandwidth. Anticipating this problem and trying to alleviate it, the network operators deploy caches in strategic places in the network. We investigate the performance of such a caching strategy and the impact of its size and the cache update logic.

We first analyse and model the evolution of video popularity over time based on traces we collected during ten months. Through simulations we compare the performance of the traditional LRU (Least-Recently Used) and LFU (Least-Frequently Used) caching algorithms to our own algorithm. We also compare their performance with a "perfect" caching algorithm, which *knows* and hence does not *have to estimate* the video request rates.

In the experimental data we see that the video parameters from the popularity evolution law can be clustered. Therefore we investigate theoretical models than can capture these clusters and we study the impact of clustering on the caching performance. Finally, some considerations on the optimal cache placement are presented.

Keywords IPTV, on-demand services, caching, Catch-up TV.

1. INTRODUCTION

There is a clearly pronounced trend towards more interactive TeleVision (TV) services, offering the users complete control over when and where they watch the available content. Various (near) on-demand services have already been proposed over broadcast networks [7],[14],[15],[19],[20]. However, an Internet Protocol (IP) network is excellently suited to support such interactive TV services, because it has a native return channel to easily convey the user requests (or preferences) to the control center and because it can straightforwardly address each user individually. In the remainder of this paper we concentrate on TV offered over a walled-garden IP network, referred to as IPTV, but many of the conclusions carry over to other TV content distribution platforms.

For offering traditional broadcast services, also referred to as Linear Programming TV (LPTV), the IPTV operator relies on multicast to save transport capacity. More specifically, each channel of the offered bouquet needs to be transported at most once on the feeder link towards a multicast-enabled node. If there is at least one user viewing the channel and served by that node, the channel needs to be transported exactly once on the feeder link. A node can duplicate a flow it receives from the feeder link to as many destinations as have subscribed to that channel. The channel does not need to be transported at all over the feeder link, if there are no users served by that node who are viewing that particular channel. This technique saves considerable transport capacity on those feeder links [3],[19].

Besides this traditional LPTV service, interactive services are becoming popular. One such example is Catch-Up TV (CUTV). CUTV allows the users to select a program that was recently aired, without having to indicate this intention prior to the airing time (as the user would have had to do with a traditional video recorder).

For these interactive services, multicast can no longer be relied upon. In principle, for each request a unicast flow must be set up from the origin server to the user. That unicast flow transports the content destined for that particular user (and no packets are duplicated in the network). Once these interactive services increase in popularity, they will have an overwhelming impact on the traffic volume in some parts of the network. Fortunately, redirection mechanisms exist such that the content does not always need to be transported from the origin server [4],[16]. The user still asks the origin server for a flow, but the server redirects the user to a cache close to that

user. This allows a subset of the content to be duplicated in caches deployed in the network or at the user premises. When such caches are installed in strategic places in the network, the flows destined for individual users do not need to travel all the way from the origin server, but can start from the cache. Unfortunately the storage capacity of these caches is limited. As such it needs to be carefully decided which objects are stored in the cache at any moment in time. It is obvious that this decision to cache (or not to cache) an object should be based on its popularity (or more precisely, its expected request rate). Caching the most popular objects (at the expense of the less popular ones) ensures that a maximal amount of requests can be served from the cache. In this way, reduction in the traffic volume on the feeder links upstream of the cache can be achieved.

Caching strategies have been used for more than a decade to cache large web objects [16]. Dimensioning of caches for on-demand television services has been studied in [19] in the context of a cable television distribution network, in [9] in the context of Internet TV and in [8],[13],[18] in an IPTV context.

Often, when determining the cache size, it is assumed that the popularity distribution of the offered multimedia content is a priori known and static. Yet, in reality, the popularity of the objects is not known by the cache and evolves over time. Therefore the popularity of objects needs to be predicted, measured and tracked over time by monitoring and aggregating the user requests for the objects [8]. The new interactive television services pose new requirements on the cache and the caching algorithm, because the typical lifetime of the objects is different from the one of web objects and the objects themselves are much larger.

The requests a group of users generates for a set of objects offered in an on-demand service form a random process governed by a popularity law. The popularity distribution of (web) objects has been studied extensively [5]. Often the popularity is modelled as a static one by a Zipf or Zipf-Mandelbrot law of which the parameters are tuned by collecting the user requests for the objects over a long period of time. These popularity laws do not capture the fact that the popularity of objects evolves over time. In some studies this volatility of objects has been taken into account by feeding the caching algorithm with a trace obtained from observing requests on an existing web server

[6]. In this paper we use the model introduced in [2] that takes the evolution of the video object popularity into account. We tune the parameters of this model based on data we collected for a CUTV service.

In order to make sensible caching decisions, an algorithm to track the momentary popularity is needed. We use a caching algorithm based on the one introduced in [8]. It relies only on the observed requests to track the popularity. Notice that the two traditional caching algorithms, i.e., Least-Recently Used (LRU) and Least-Frequently Used (LFU), (implicitly) measure the popularities of the objects as well [17]. We will assess how the caching algorithm studied in this paper decreases the (peak) bit rate of the aggregate of unicast flows on the feeder link upstream of the cache, while still keeping the cache updates to a minimum. We will compare its performance to a benchmark caching algorithm that is aware of the true object popularity and to the performance of the LRU and LFU algorithms.

This paper is organised as follows. In the following Section 2 we present schematically the IPTV architecture, the new (time-shifted) services, the purpose and the possible locations of caches. In Section 3 we discuss statistics we collected related to a CUTV service and suggest clustering of the parameters. In Section 4 we motivate the necessity of introducing caches on certain network nodes by showing that otherwise the required capacity risks to grow exorbitantly when the new interactive services gain in popularity. In the following Section 5 we describe the proposed caching algorithm as well as a "perfect" caching algorithm and the two corresponding simulators we wrote. Subsequently, in the next Section 6, we present results by the two described simulators. Finally, Section 7 summarises the conclusions of our work.

2. IPTV ARCHITECTURE AND SERVICES

2.1 IPTV Architecture

For a detailed description of the end-to-end IPTV architecture we refer to [1]. In this paper we focus on the access network illustrated in Figure 1, which shows that the access network supporting IPTV services is a tree network. At the root of the tree, content is injected, either on a multicast tree in case of the traditional LPTV service or on an on-demand server, e.g., in the case of CUTV, from where the user can access it via a unicast flow. These (content

dissemination) servers are connected to a Service Router (SR) through a distribution network. An aggregation network connects the SR to tens of Digital Subscriber Line Access Multiplexers (DSLAMs). A DSLAM usually serves a couple of hundred of Home Gateways (HGs) (not shown in the figure), each supporting a couple of set-top boxes (STBs) that a typical household has. All of these STBs are not necessarily actively using IPTV services at a random moment in time.

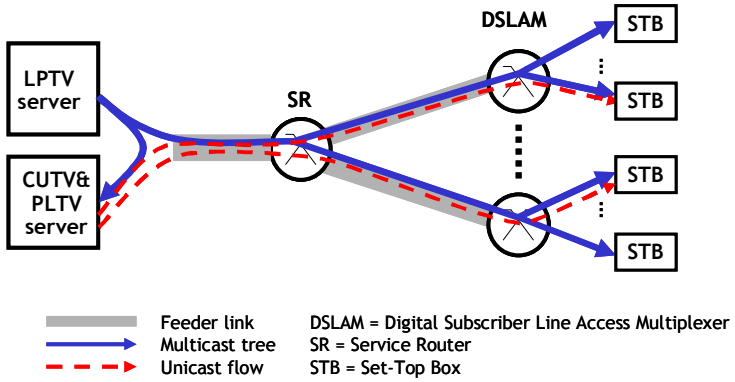


Figure 1: Access network supporting IPTV services.

Although Figure 1 presents a schematised IPTV access network in the specific case of IPTV over a DSL network, only small modifications are needed for the cases of a non-DSL topology e.g., a (coaxial or hybrid) Cable TV architecture. The only difference is that specific building blocks (QAM modules, etc.) are shared media rather than a dedicated line per user.

Every of the depicted nodes (outside of the content servers) can contain a video replication cache, but in this paper we consider caches that are deployed strategically only in the SRs or the DSLAMs. A cache located on a STB is not considered here because in the topology of Figure 1 only a few end users are connected to it and once a video content has been downloaded and watched, the probability of it being requested again is negligible. This would change if the STBs are running a peer-to-peer protocol, but we do not consider this because the upstream link rate of the DSL line is in most cases configured to a small value.

2.2 Linear Programming TV (LPTV)

The traditional TV broadcast model (LPTV) is based on the "push content" paradigm, where the broadcast operator decides on the bouquet of offered TV channels (and related services) and the subscriber is a passive consumer of those. The content is transmitted live (aired in real-time) and requires streaming technology (often in multicast mode) to the end user. Sometimes a cache is deployed to enable a Pause-Live-TV service, which allows a limited trick-play functionality (pause, fast-backward, fast-forward to live, …).

The required bandwidth depends rather on the size of the bouquet of TV channels and services than on the number of users. Since the number of these services is constant (more precisely varying slowly with time), the required bandwidth is not hard to estimate taking into account the user behaviour. As explained in [3], resource demand reduction can be achieved in this case mainly by the multicast technique or the switched multicast technique.

2.3 Catch-up TV (CUTV)

Many television content providers already offer a CUTV service either via collaboration with an IPTV provider over a walled-garden IPTV system or via a web portal over the open Internet. Examples of the latter are: ABC's "Full Episode Player" [10], BBC's "iPlayer" [11] and the Dutch "Uitzendinggemist" [12], Hulu [20]. Although they are offered over the open Internet, they can typically only be accessed in their country of origin. The offered content is either of a lower resolution than the original aired content or the user cannot directly watch the content after he or she selected it, but only after some initial start-up buffering time. These limitations on the Quality of Service (QoS) parameters are imposed from the limited available network resources.

Thus, CUTV is not a live service anymore (like the LPTV one), but still a streaming service (as a general rule) for the time of the video watching duration (we do not consider an on-demand download service). For every requested multimedia (video) object, a dedicated unicast connection to the STB is set up, originating in the first service node up in the distribution tree which contains the requested object. In this way, the required bandwidth is proportional to the number of active users. This threatens to lead to explosion in resources demand. A popular solution to that is to provide the service nodes (e.g., SRs, DSLAMs) with caches. Those exploit the fact that a large number of users are connected to this node and there exists a non-negligible probability that new viewers request video content that was

already viewed by other users in the group served by that node. This statistical property of the user behaviour, coupled with a good estimation of which are the most popular objects to be cached, can lead to a considerable bandwidth demand reduction (on the distribution or aggregation links).

A good caching algorithm though, has an adequate estimate of the video objects popularity at any moment, an estimation as close as possible to the objective videos popularity distribution. A high Hit Ratio (HR), i.e., the fraction of the requests that can be served from the cache, ensures capacity saving on the link upstream of the cache provided that the cache is large enough and the cache has an appropriate cache logic to estimate the content popularity. In the following Section 3 we propose a function which models the popularity evolution over time of a video object and we describe the data set on which its viability has been tested.

3. USER BEHAVIOUR AND MULTIMEDIA OBJECTS POPULARITY

3.1 Experimental Data

We have monitored a CUTV web site [12] in order to collect experimental data and analyse how the popularity (under the form of request rate) of video objects evolves over time. Once aired live on the standard LPTV, a video object is placed in this on-line store from where it can be requested for viewing again later at any time. Initially, we built a data set of video IDs collected during three months (the videos uploaded, introduced every day in a period of three months). We consecutively continued monitoring the accumulated number of views these videos obtain on a daily basis for seven more months. In other words each day we collected the cumulative views up to that day. So, the longest video traces are about 10 months long while the shortest traces are 7 months long. We collected 2140 video views traces, but during our analysis and modelling process we discarded some 500 which were not well-modelled for various reasons (mainly because they were aired more than one time during the monitoring period, which gives them a boost halfway and hence a trace like this is in fact a shifted sum of several separate shorter traces, which we did not attempt to model).

3.2 Generic User Behaviour Model

In [2] we proposed an analytical model for the way accumulated views to videos grow. In fact, this analytical model describes the (measured or predicted) accumulated number of views for a video up to a certain moment in time – *t*.

The time derivative of this model gives the request rate over the small interval [t, t+dt]. This analytical expression is a generic template that can model the popularity evolution over time of a video trace and depending on the value of the parameters can expose an exponential (short-tail) decay and a power-law (long-tail) decay. The model we proposed in [2] can degenerate into either a power-law or exponential decay law depending on a form-determining parameter, denoted as α. This allows a large range of video popularity decays to be modelled. By fitting this model to the traces we will find out whether the traces are heavy-tailed or not.

Let $I_k(t)$ denote the accumulated number of views (i.e., cumulative distribution of the popularity of the video trace), i.e., $I_k(t)$ denotes the total demand for watching the k–th video until time instant t. This function, which depends on four parameters, has the following shape:

$$I_k(t) = \rho_k \left[1 - \left(1 + \frac{\left(0.5^{-\frac{1}{\alpha_k}} - 1 \right)(t - \theta_k)}{\tau_k} \right)^{-\alpha_k} \right] \text{ if } t > \theta_k \text{ and } I_k(t) = 0 \text{ if } t < \theta_k \,. \quad (1)$$

The parameter ρ_k represents the total number of views a video k receives in its complete lifetime. The parameter θ_k is the introduction time of a video, i.e, the time when a previously broadcast video object is uploaded to the on-line CUTV video store. The parameter τ_k can be interpreted as the video's half-lifetime, more precisely the time by which half of the total views are accumulated. Thus τ_k is the median of the distribution, corresponding to a point in time when half of the potential population has been reached (assuming a user watches only once a given video). The subscript k towards all the parameters expresses the fact that they are associated with a given video k and are different for every separate video object.

The parameter α_k in eq. (1) is the form-determining parameter of the function, that is, it determines whether it is heavy-tailed or not and it is also video-object specific (it also has a subscript k). In Figure 2 we illustrate the function of eq. (1) with three values for the α parameter (namely α = 1, 10, 100) and demonstrate how with large α the function converges fast to an exponential function. Otherwise, if α is smaller than 10 (and close to 1), the trace

has a heavy tail. This duality of the function accounts for the following physical phenomena: in the case the popularity evolution of video k is described by an exponential function, once the half-life time τ_k is reached, the video gathers the rest of its views very quickly; on the contrary, in the case of a power-law decay, the video requests are spread over a considerable time after the half-lifetime is surpassed.

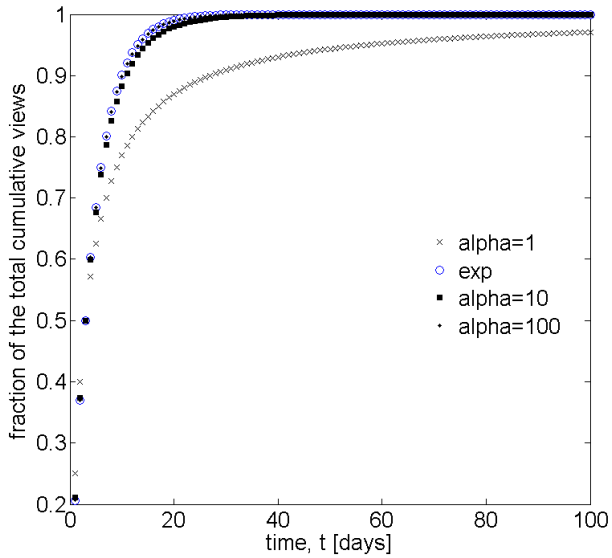


Figure 2 When the form-determining parameter $\alpha \to \infty$, the function of eq. (1) approximates an exponential function; the other parameters are ρ=1, τ=3, θ=0.

3.3 Estimating Parameters from the Experimental Data

We applied the methodology described in detail in [2] to estimate the three model parameters $(\rho_k, \tau_k, \alpha_k)$ (of eq. (1)) of each video k of the collected video views traces (the fourth one, i.e., the introduction time θ_k, is given and we do not have to estimate it). In Figure 3 we depict three typical cumulative views traces from the data set and we superimpose the approximation with the function of eq. (1). The time unit is a day. We demonstrate how well "Trace A" (power-law) and "Trace B" (exponential form) are approximated by the synthesised traces with the estimated parameters. Unfortunately, our function fails to accurately model "Trace C". The episode of "Trace C" has been aired a second time approximately 5 months after the first broadcast and this gives a boost in the accumulated views shortly after day 150, which our model does not capture. We could have split this trace into two parts: before and after the re-airing time, but we chose to focus our analysis on clean traces.

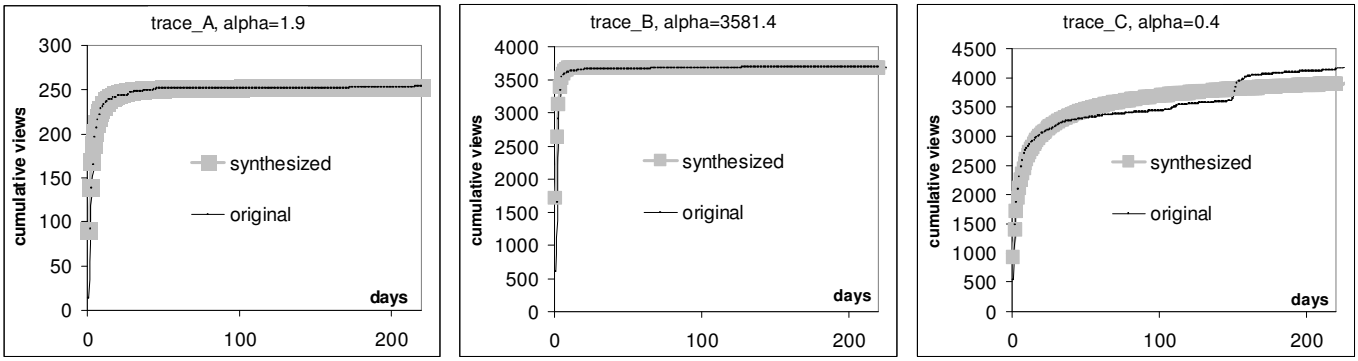
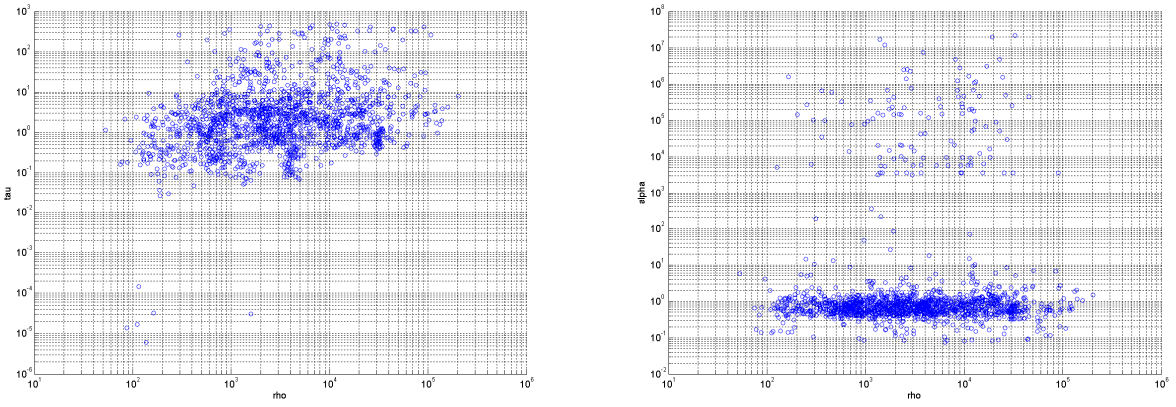


Figure 3 Three video views traces (dark curve) approximated with the proposed function in eq. (1) (light curve) with the corresponding estimated parameters values.



(a) (b)

Figure 4 Two sections of the 3D-graph of (ρ,τ,α): (a) ρ-τ section showing no correlation between ρ and τ, (b) ρ-α section demonstrating the two α clusters.

We note first that the (ρ_k,τ_k,α_k) parameter sets associated with the different traces are independent of one another. In Figure 4 (a) we plot the τ values of all the videos versus their respective ρ values. The scatter plot shows that there is a negligible correlation between ρ and τ (in fact the correlation coefficient between both variables is 0.095), so that we consider them as independent. In Figure 4 (b) we show the scatter plot of α versus ρ and two clusters for α can be discerned. The cluster with small α values corresponds to the traces with power-law form of the popularity decay, while the smaller cluster of large α values pertains to the traces that "die out" fast (i.e., have an exponential behaviour) regardless of what part of the subscribers audience they have reached (i.e., independently of the value of the ρ parameter). Since ρ and τ are independent, we do not need to know their joint distributions, and knowing their marginal distributions is sufficient. Further in this paper we will assume that the marginal distributions of the ρ and τ variables are exponentially distributed while the parameter α can take a finite number of fixed values (in most

cases one or two) and we will check below to what extent this assumption holds. The magnitude of the form factor α, namely is it "small" or "large", is sufficiently relevant for the popularity evolution decay (as is illustrated in Figure 2). We select 10 as the formal boundary value for classifying in the category "small" or "large" for the form-parameter of a given trace.

The analysis of the traces shows that the average τ is 14 days, i.e., in the first two weeks most of the videos have received half of the views they will ever get throughout their lifetime. However, if we cluster the traces by the value of their α parameter (see the case of two clusters in TABLE I), the 92 % of them with "small" α have a mean decay time τ of 13 days and the rest 8 % (with exponential request rate decay) have a τ of 28.5 days.

We can consider and exploit the video traces parameters as one cluster, or we can divide the data set into two clusters (e.g., in the manner described above based on the value of α); we can even further subdivide into smaller clusters, e.g., by setting one threshold value for every of the three parameters which results in 8 clusters (see TABLE I for the average values of the parameters within a cluster). We further experimented by subdividing the larger cluster of small α by two threshold values of τ, thus obtaining 10 clusters with average parameter values also given in TABLE I.

TABLE I Clusters' parameters

1 cluster		
ρ x10^{-6}	τ	α
0.009	14.364	75036.837

2 clusters				
	%	ρ x10^{-6}	τ	α
α<10	0.92	0.009	13.139	0.858
α>10	0.08	0.008	28.474	939955.379

10 clusters				
	%	ρ x10^{-6}	τ	α
ρ < 5000, τ < 14, α<10	0.569	0.002	2.117	0.815
ρ < 5000, τ > 14 & τ < 90, α<10	0.027	0.002	30.710	1.215
ρ < 5000, τ > 90, α<10	0.011	0.002	235.686	0.242
ρ > 5000, τ < 14, α<10	0.266	0.023	2.952	0.930
ρ > 5000, τ > 14 & τ < 90, α<10	0.026	0.026	30.311	1.124
ρ > 5000, τ > 90, α<10	0.021	0.025	276.612	0.655
ρ < 5000, τ < 14, α>10	0.026	0.002	2.085	397199.740
ρ > 5000, τ < 14, α>10	0.018	0.002	66.648	1227178.647
ρ < 5000, τ > 14, α>10	0.021	0.016	2.197	1480276.614
ρ > 5000, τ > 14, α>10	0.015	0.015	62.736	772280.050

8 clusters				
	%	ρ x10^{-6}	τ	α
ρ < 5000, τ < 14, α<10	0.569	0.002	2.114	0.814
ρ > 5000, τ < 14, α<10	0.266	0.023	2.952	0.930
ρ < 5000, τ > 14, α<10	0.038	0.002	90.219	0.933
ρ > 5000, τ > 14, α<10	0.048	0.025	140.831	0.914
ρ < 5000, τ < 14, α>10	0.026	0.002	2.085	397199.740
ρ > 5000, τ < 14, α>10	0.021	0.016	2.197	1480276.614
ρ < 5000, τ > 14, α>10	0.018	0.002	66.648	1227178.647
ρ > 5000, τ > 14, α>10	0.015	0.015	62.736	772280.050

In the case of one cluster, the distributions of ρ and τ are not well modelled by an exponential function and the highest R^2 value (the Pearson product moment correlation coefficient) obtained by fitting with the best suited function is 0.62. In the case of two clusters, the distributions for ρ and τ are more exponential-like, and by fitting

the obtained histograms with the best suiting exponential functions, the obtained R^2 values are between 0.62 and 0.83, except the very low R^2 value for τ in the cluster of small α. The two biggest clusters in the 8- and 10-clusters cases, have ρ and τ parameters better approximated by exponential functions (their R^2 ranges between 0.76 and 0.84).

4. BIT RATE EXPLOSION WITH CUTV

In this paper we consider the transport capacity required on the distribution network (i.e., the link feeding the SR) and on the aggregation network (i.e., the link feeding the DSLAM). To that end, we consider the number of concurrent unicast flows that such a link needs to support. We express the required capacity in terms of the number of flows the link needs to support. We do not discuss in detail how much bandwidth is required by each of these flows and hence the aggregate of unicast flows. For standard definition television this would be around 2 Mbps per flow, while for high definition television it would be about 8 Mbps per flow.

An IPTV operator will dimension the feeder links towards the SR and towards the DSLAM at least to support LPTV. This means typically supporting a bouquet of a few tens to hundreds of LPTV channels. So, for an LPTV service the distribution network and the aggregation network need to support at most this number of LPTV channels (and less if not all channels are watched by a group of users [3]). We assume that this capacity is foreseen and concentrate on the required capacity for a CUTV service.

The required capacity (without caching) for a CUTV service is proportional to the number of users of the service rather than the number of video objects offered as for LPTV. The number of simultaneous unicast flows can be modelled as a stochastic variable: this number of flows increases or decreases as the users select CUTV objects (and finish watching them) and this in turn depends on the evolving popularities. The Complementary Cumulative Distribution Function (CCDF) of this number of unicast flows captures its statistical properties (fluctuations). More precisely, a CCDF graph shows for each value of the capacity (on the abscissa) the probability (on the ordinate) that the aggregate number of simultaneous unicast flows exceeds this value. In this context, this probability can also be interpreted as the fraction of the time that this capacity is exceeded. If the fluctuating

capacity demand exceeds the (chosen) available capacity, then this CUTV system can take any of a couple of actions: it can block the new request, in which case the probability to exceed is the blocking probability (probability of unavailability) of the system or it can allow the new requests, if it is a buffer-equipped network, in which case the probability to exceed is the overflow probability (or packet loss probability, in case the buffers are small). Given the user behaviour model and the response of the system to overflow, this metric can always be translated into a user-perceived quality of experience (QoE) parameter. The CCDF provides all the information that is needed to dimension the link: at a given desired probability of unavailability, the required capacity can be directly read from the CCDF graph. This explains why we choose the performance metric "likelihood to exceed".

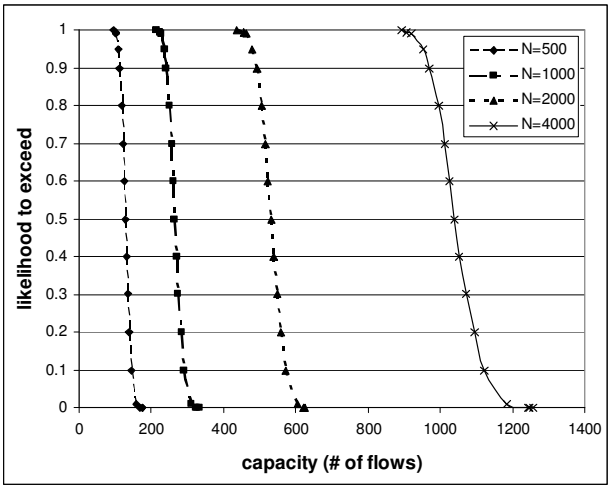


Figure 5: CCDF of the transport capacity on the feeder link towards a node for various user populations sizes N=500, 1000, 2000, 4000 (no clustering, ϕ = 500 files/day, video duration T_v = 0.0625 day).

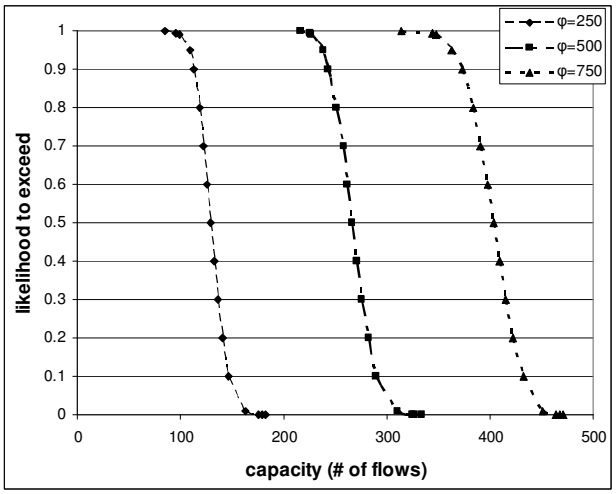


Figure 6: CCDF of the transport capacity on the feeder link towards a node serving a population of N = 1000 users (no clustering, ϕ in files/day, video duration T_v = 0.0625 day).

Figure 5 (obtained with the simulator described later in this paper in Section 5.2) shows the CCDF of the number of simultaneous unicast flows on the feeder links (towards the DSLAM or the SR) as the number N of CUTV users increases (namely for N = 500, 1000, 2000, 4000). It illustrates how the required capacity increases with increasing number of users (concurrently using the CUTV service). Figure 6 (also obtained with the simulator described in Section 5.2) shows how the CCDF of the number of simultaneous unicast flows grows when the video ingestion rate ϕ of a Poisson traffic increases (ϕ can increase, for example, if more channels offer a CUTV service). Both graphs demonstrate that if either the number of users N or the introduction rate ϕ of new episodes increases, the required capacity does too.

We will describe in more detail how these CCDFs are obtained in Section 5.

5. CACHING ALGORITHM

As explained and demonstrated in Section 4, in order to keep under control the traffic in the aggregation part of the network and to avoid bit rate explosion, caches are deployed, situated either in the SRs or the DSLAMs. However, when deploying caches, the "best suited" objects should be stored in them (at the moments when episodes are requested) in order to achieve higher efficiency (this requires a high Hit Ratio (HR), a parameter we define below). With the evolving object popularities this is not straightforward. The caches are deployed with as primary objective to reduce the traffic volume of the aggregate of unicast flows that needs to be supported on the feeder link (in the aggregation and distribution network) upstream of the cache. However, as a secondary objective, the cache updates should be minimised as well (which requires a low Update Ratio (UR), a parameter which is defined below). The former objective ultimately results in a cheaper (aggregation and distribution) network by which the investment in the cache could be justified. The second aim helps to reduce the cost of the cache as each cache update requires computational resources of the caching hardware and consumes capacity on the ingest link to the cache.

If the caching algorithm knew the exact popularity of objects at any decision-taking moment, it would have been caching always the most popular video objects for which the highest view request rates are expected. A simulator to assess the performance of the "perfect" caching algorithm is described in Section 5.1.

However, in reality, the actual popularity distribution of objects is unknown and a good caching algorithm tries to approach it by an intelligent guess, making use of the track record kept of the requests to all videos. Section 5.2 is dedicated to the description of such a simulator to assess the performance of three real caching algorithms.

A user request at a given time instant results either in a "hit", in a "miss" or an "update". In case of a "hit", the requested object is found in the cache and a unicast flow is set up from the cache to the user, alleviating the traffic on the feeder link upstream from the cache. In case of a "miss", a unicast flow is set up from the origin server to the user (that travels over all parts of the network). In case of an "update", a unicast flow is set up to feed the cache and the user is served via a unicast flow from the cache by the object that is gradually being built up in the cache. Therefore, unless there is a "hit", the network links upstream of the cache need to support a unicast flow.

There are two main metrics which define the cache performance. One of them is the Hit Ratio (HR) and the other is the Update Ratio (UR). The HR is defined as the fraction of requests that can be served directly by the cache. It is easy to see that the average number of flows on the feeder link towards a node with a cache is (1-HR) times the average number of flows in case this node would not have a cache. The Update Ratio (UR) is the ratio between the number of cache updates and the number of ingested objects observed over a very long period of time. If we denote by K the number of objects ingested in the CUTV on-line store, by M - the number of requests, by M_h - the number of hits and by K_u - the number of updates, then HR = M_h/M and UR = K_u/K. The HR is always smaller than 1 and a value close to 1 should be aimed for, while the UR is unbounded and a value as low as possible should be aimed for.

5.1 Perfect Caching Algorithm

In a perfect caching algorithm, the objects popularity distribution is perfectly known at any given moment and thus, the best decision on which objects to be cached is taken. We constructed a simulator which assesses the performance of such a perfect caching algorithm and therefore gives the benchmark, i.e., the maximum achievable HR with a given cache size *L*. In fact, in this simulator we introduce video objects in the video store according to a Poisson process and associate with each video object a popularity evolution curve (determined by eq. (1)). The concrete ρ_k, τ_k and α_k video parameters are assigned according to the statistics described in Section 3.3. We assume that the caching algorithm is aware of the actual video popularity ranks at any given moment. The simulator makes a "snapshot" of the momentary videos popularity distribution at a randomly chosen point in time and stores the first *L* most popular videos ranked up in the list. The HR is the average of the sum of all momentary request rates of the cached objects at that random moment in time divided by the average of all momentary request rates at that random moment in time. The averages are estimated by generating a sufficiently large amount of random time instants.

To determine the statistics of the parameters ρ_k, τ_k and α_k, our simulator can have two types of input:

- *either* every introduced video trace imitates one of the 1640 analysed traces, selected at random; by the three estimated parameters $(\rho_k, \tau_k, \alpha_k)$ on the real trace, eq. (1) gives an answer on the video popularity at any moment in the lifetime of the synthesised trace;

- *or* videos are introduced with $(\rho_k, \tau_k, \alpha_k)$ parameters as belonging to a cluster (see TABLE I); the α_k parameter is taken as a constant specific for the given cluster, while ρ_k and τ_k are associated with exponential distributions with an average value defined from the cluster.

5.2 Real Caching Algorithm

The video popularity evolution curves for the real caching algorithm are generated in the same way as for the perfect caching algorithm. In contrast with the perfect caching algorithm, the real caching algorithm is not aware of

the popularity evolution curve associated with each video object, but only sees the requests for video objects made by the users.

The real caching algorithm that we consider in this paper operates according to the caching logic described in [8]. It makes the decision to cache or not to cache an object at each user request instant t_i. This caching decision is based on the measured object request rate from the recorded history. In fact, the caching algorithm attempts to track the true object popularity evolution (defined by eq. (1)) as closely as possible. In order to do so, the caching algorithm keeps and updates a vector π of size K_t – one entry for every ingested video object until moment t, thus with entries π_l corresponding to the l-th ingested object. These measures π_l of the video request rate (popularity) are updated at each user request instant t_i according to the rule

$$\begin{aligned} &\pi_l(t_i) = \lambda \cdot \pi_l(t_{i-1}) + (1-\lambda)\cdot \delta_{lk} \quad \forall l = 1,2,...,K_t \\ &\lambda = \exp(-B\cdot(t_i - t_{i-1})) \end{aligned}, \tag{2}$$

where k is the identifier of the object that was requested at instant t_i and δ_{lk} is the Kronecker delta (which is 0 for all l except for $l=k$ where it is 1). At the content ingestion instant θ_k (prior to the first (observed) request for that object), the entry π_k of the vector π is initialised to 0. Notice that π_k is not an estimate of the request rate itself, but is proportional (with factor $B>0$) to the request rate.

Eq. (2) can be interpreted as follows. At any moment in time (and in particular at request times), the k-th entry of the vector π accounts for the number of requests for object k over the past period. The longer the request happened in the past, the less it is weighted. In fact, the procedure boils down to using a running exponential window with width $1/B$. How to tune B will be discussed in the next section.

Consider a cache that has room to contain L objects. Remark that we express the cache size as the number of objects it can store. In principle each object can require a different storage space, but we do not take that into account here. The caching algorithm is such that the L objects with the highest π_l-value reside in the cache. Since at each time instant t_i only one entry of the vector π increases (while the other entries decrease), only the

corresponding object is eligible to be moved to the cache (at the expense of the cached object that has the lowest π_l -value amongst the cache-contained objects).

Following a request, a unicast flow is set up for the requesting user, originating either at the video content server or at a cache. At every request moment, it is decided whether or not the *k*-th object merits moving into the cache based on this new value of π_k. If the decision is in favour of caching the object, a unicast flow is set up from the origin server to the cache and lasts for a duration T_v. In this paper we take this duration to be a constant, but in further elaborations, this could be a random variable that reflects the fact that not all episodes last for the same time or are watched until the end.

We wrote an event-driven simulator that generates the requests for objects according to the non-stationary Poisson process determined by eq. (1), but the ingestion times θ_k are generated according to a stationary Poisson process. The user demand function of eq. (1) statistically determines the requests for the *k*-th object made by a community of *N* users subscribed to the CUTV service as follows. The requests for object *k* are described by a non-stationary Poisson process with a time-varying arrival rate $N \cdot D_k(t)$, where $D_k(t)$ is the derivative of $I_k(t)$. The non-stationary Poisson processes associated with individual objects run concurrently and are statistically independent.

At the ingestion time of the *k*-th object, the parameters of the demand curve (i.e., ρ_k, τ_k and α_k - see eq. (1)), are chosen again in one of the two following ways described in Section 5.1: *either* every introduced video trace imitates one of the 1640 analysed traces, picked at random *or* videos are introduced with $(\rho_k, \tau_k, \alpha_k)$ parameters as belonging to a cluster (see TABLE I).

The curve of eq. (1), specific for the *k*-th object, is used in a non-stationary Poisson process to generate the time instants at which object *k* is requested. The caching algorithm has knowledge of the ingestion times θ_k and sees the request instants t_i for the objects. Although the caching algorithm implicitly tries to estimate the demand curve associated with object *k*, it actually has no access to the parameters of this curve with which the requests were

generated. As such the caching algorithm only has indirect and imperfect knowledge of the actual request rate unlike the perfect caching considered in Section 5.1.

After a prerun sufficiently long such that the system reaches the steady state (see [8]), the simulator starts to count the number K of objects ingested, the number M of requests, the number M_h of hits and the number K_u of updates. Moreover, it tracks the number of concurrent unicast flows upstream of the cache by sampling this evolving random variable at regular time intervals.

After a simulation run long enough for the stochastic fluctuations to have become sufficiently small, the simulator reports the HR = M_h/M and the UR = K_u/K. The CCDF of the required bit rate on the link upstream of the cache is constructed based on the samples of the number of concurrent unicast flows the simulator has accumulated (by probing the link regularly) and the percentiles are calculated on this set of samples. The capacity corresponding to a percentile is a point on the CCDF curve. We calculate enough percentiles to determine the CCDF accurately.

6. RESULTS

6.1 Comparison of Real Caching Algorithms

In what follows we will speculate on the choice of the parameter B of the caching algorithm. The caching system tends to the Least-Recently Used (LRU) caching strategy if B tends to infinity, while it tends to the Least-Frequently Used (LFU) if B tends to 0 [17]. This is tackled in more detail in [8]. Thus, assigning different values to B, we can imitate the two often used algorithms: LFU and LRU and compare their performance to the performance of the proposed caching algorithm.

In fact, it can be proved that $E[\pi_l]$ evolves over time as the convolution of the true request rate of eq. (1) with the exponential window of eq. (2). So, in order to reduce the noise as much as possible, while still keeping the lag under control, the parameter B should be chosen considerably larger, say about 10 times larger, than $1/\tau_k$. On the other hand, choosing B too high will make the algorithm react too "nervously" as is illustrated in Figure 7. Although an extension with an individual B-value per object k would be feasible, if there would be a means to get an estimate for τ_k, in this paper we try to choose one B that suits all objects.

We explored the influence of the cache tuning parameter B on the hit ratio, HR and the update ratio, UR under a range of cache sizes, namely for L = 625, 1250, 2500, 5000, 10000, 20000, 40000, 80000 and running simulations for four values of B: B = 20 (too large), B = 2.5, B = 0.5 (our choice), and B = 0.1 (too small). Except in this section, in the remainder of this paper we set $B = 7/E[\tau_k]$, which gives a value of B = 0.5 for an average $\tau = 14$ over all the traces (see TABLE I). The results are displayed in Figure 8 and Figure 9.

Figure 8 shows that for all cache sizes, a choice of B in the range 0.5 and 2.5 would hardly impact HR. This figure demonstrates also that with small cache sizes (L = 625 - 5000), setting B too low (corresponding to the LRU algorithm) or too high (corresponding to the LFU algorithm) degrades the performance of the cache in terms of expected hit rate. With small cache sizes, the choice of B has a drastic impact on the update ratio as well, as is demonstrated in Figure 9. With small B value, a low UR is maintained but this is done at the expense of obtaining a low HR. We observe again that the choice of the cache tuning parameter hardly matters for the UR of large buffers.

Figure 8 and Figure 9 illustrate the superiority of the proposed caching algorithm in [8] with respect to the LRU and LFU caching strategies.

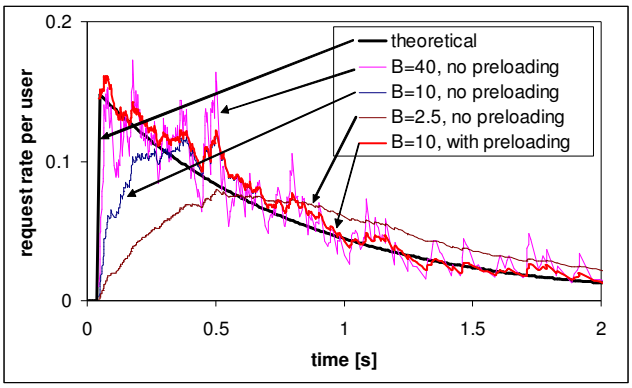


Figure 7 Illustration of the impact of B on the estimate π_k of the true request rate (see [8]).

6.2 Approximation with Clusters

First we want to assess if the list of 1640 traces can be modelled by a set of clusters. Therefore, we use the simulator to assess the performance of the ideal caching algorithm when the video traces parameters are not

clustered (i.e., each time a new video is introduced, its parameters are defined by randomly selecting one parameters set associated with one out of the 1640 traces) to the cases when the video traces are generated according to clustered parameters.

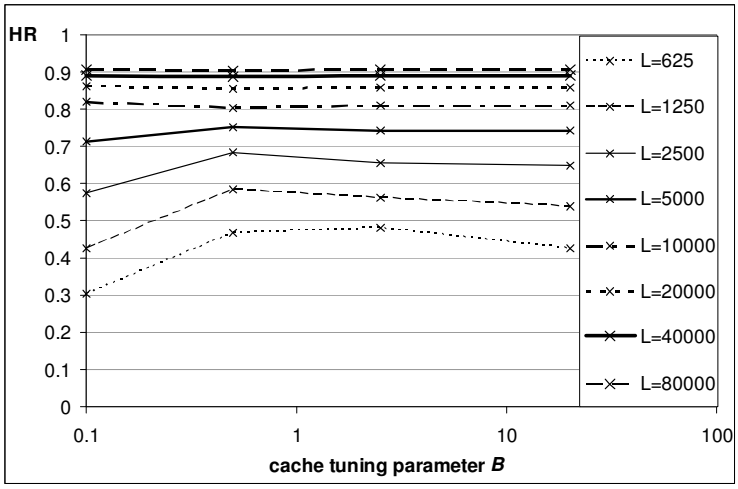


Figure 8 HR in function of *B* for various cache size *L*.

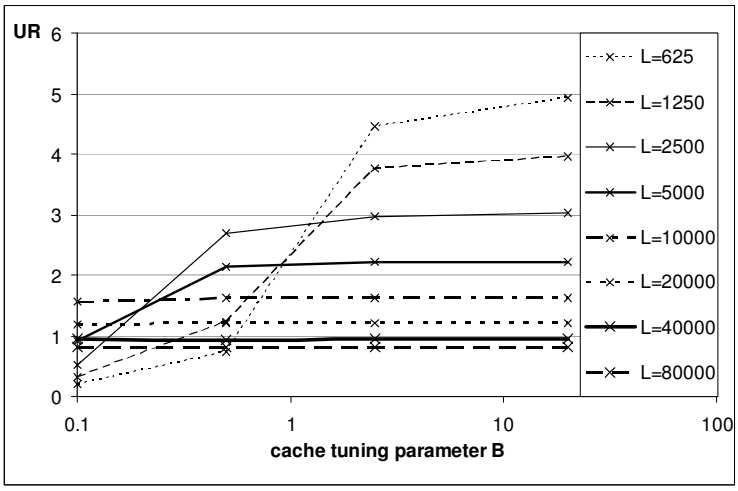


Figure 9 UR in function of *B* for various cache size *L*.

With the perfect caching algorithm of Section 5.1 we generate the benchmark HR curve, for the case where the input is a list of 1640 $(\rho_k, \tau_k, \alpha_k)$ triples (the non-cluster type of video definition) and for the cases with various sets of cluster descriptions (see TABLE I). The results are displayed in Figure 10.

The approximation to the HR curve associated with the list is very poor with 1 and 2 clusters, but it is good with 8 and 10 clusters. The curves associated with 1 and with 2 clusters perform equally bad although it can be noticed

that the curve pertaining to 2 clusters follows the shape of the real HR curve. The reason for this is that in this case the dominant number of videos have a power-law popularity decay shape while the curve with one cluster is dominated by traces with exponential decay (large α_k) and does not reflect the reality.

Defining clusters is tricky and obviously no positive effect is necessarily achieved by simply increasing the number of clusters: the results with 8 clusters slightly outperform those with 10 clusters in the sense that the curve with 10 clusters approximates slightly worse the benchmark curve of non-cluster input video parameters). Constructing adequate clusters can be considered as "classifying and summarising" the behaviour of the 1640 videos in the list.

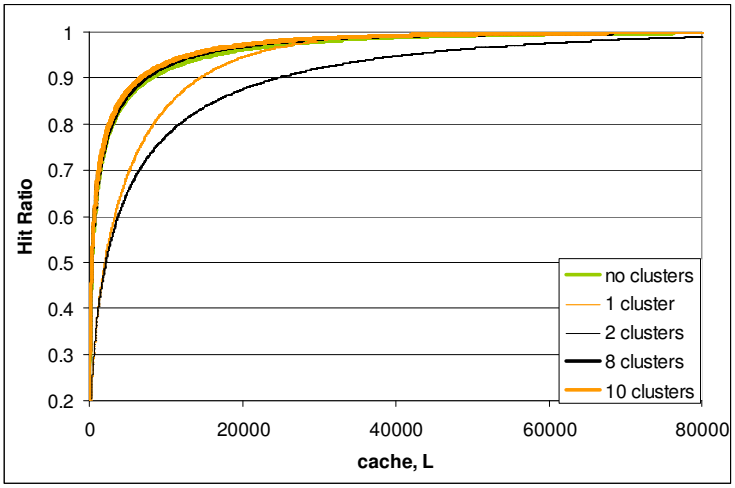


Figure 10 HR curves by the perfect cache simulator: videos are generated either as defined by the list of 1640 videos (no clusters curve) or as defined by 1, 2, 8 or 10 clusters of the (ρ, τ, α) parameters.

6.3 Performance of the Proposed Caching Algorithm

In this section we assess how close the real caching algorithm can approach the perfect caching algorithm performance.

For a node with N = 1000 users, we obtain the HR curves by the proposed caching algorithm for cache sizes of L = 625, 1250, 2500, 5000, 10000, 20000, 40000, 80000 objects. The corresponding curves by both simulators (described respectively in Section 5.1 and 5.2) for 1, 2, 8 and 10 clusters as well as the real HR curves associated with 1640 videos are compared in Figure 11. The dashed lines pertain to the proposed real caching algorithm (described in Section 5.2), while the solid lines pertain to the perfect one (Section 5.1). The corresponding curves are of the same shape, making that also for the real caching algorithm the approximation with 8 and 10 clusters

approaches best the HR curve associated with the list of traces. The HR curves corresponding to the proposed real caching algorithm are always below the ones corresponding to the perfect algorithm.

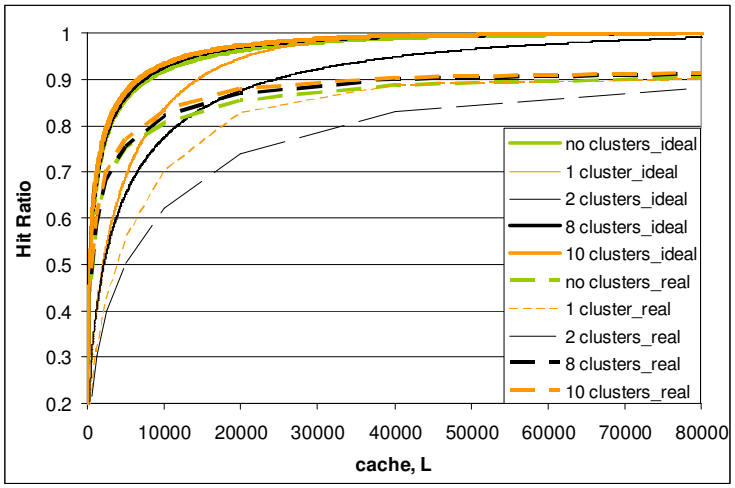


Figure 11 HR curves by the perfect cache simulator (full lines) and by the real caching algorithm (dashed lines): videos are generated either as defined by the list of 1640 videos (no clusters curve) or as defined by 1, 2, 8 or 10 clusters of the (ρ,τ,α) parameters.

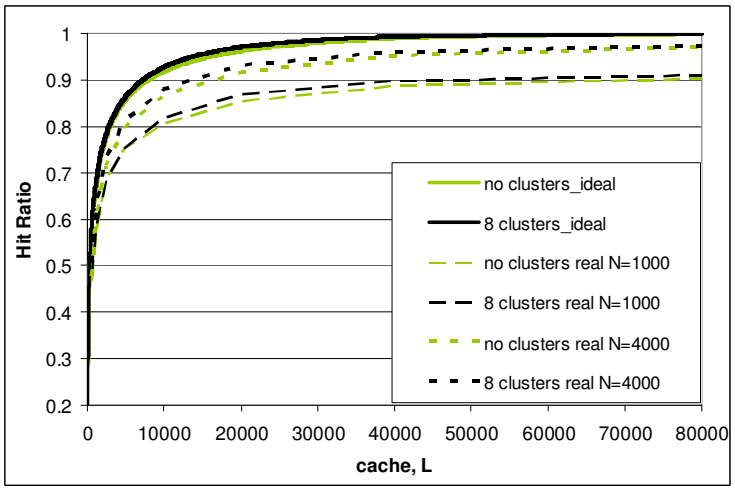


Figure 12 HR curves by the perfect cache simulator (full lines) and by the real caching algorithm (dashed lines) for *N*=1000 and *N*=4000; videos are generated either as defined by the list of 1640 videos (no clusters curve) or as defined by 8 clusters of the (ρ,τ,α) parameters (see TABLE I).

If the number of users is higher, the average HR is higher too. This is illustrated in Figure 12, superimposing results for N=1000 and N=4000 users with the perfect and the proposed real caching algorithm. Videos are generated either as defined by the list of 1640 videos (no-clusters curve) or as defined by 8 clusters of the $(\rho_k, \tau_k, \alpha_k)$ parameters.

6.4 Caching locations and resource demand reduction

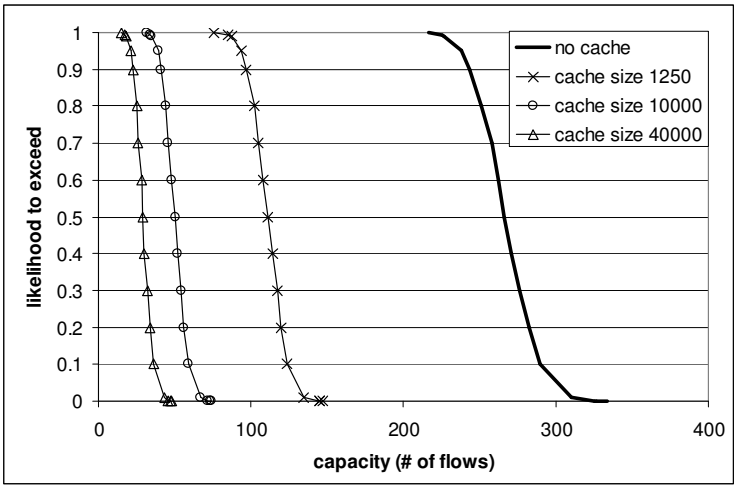
As far as the last mile link is concerned, it is clear that it needs to support as many flows as there are active STBs in the home. Indeed, since it is not unlikely that these STBs are active at the same time (unless the user behaviour could be predicted with a probability very close to 1), it is straightforward to conclude that this worst case determines the transport capacity requirement of that link. Thereafter, in the remainder of this section we focus on the required transport capacity on the distribution and aggregation network, where multiplexing gain can be exploited via introduction of caches. We limit our study to scenarios where caches are installed in only a single level in the distribution tree (i.e., either on the DSLAM or the SR).

The HR is the main factor conditioning the capacity reduction gained with the deployment of a cache in a node (though HR needs to be considered always in combination with the UR for better accuracy).

Based on the conclusion of the results displayed in Figure 12, a cache provided to a node serving a higher number of users will have a higher HR (the HR curve associated with N=4000 users is always higher than the one associated with N=1000 users at equal other conditions). Therefore, a cache in the SR will always lead to a higher bandwidth saving upstream than a cache placed in the DSLAM.

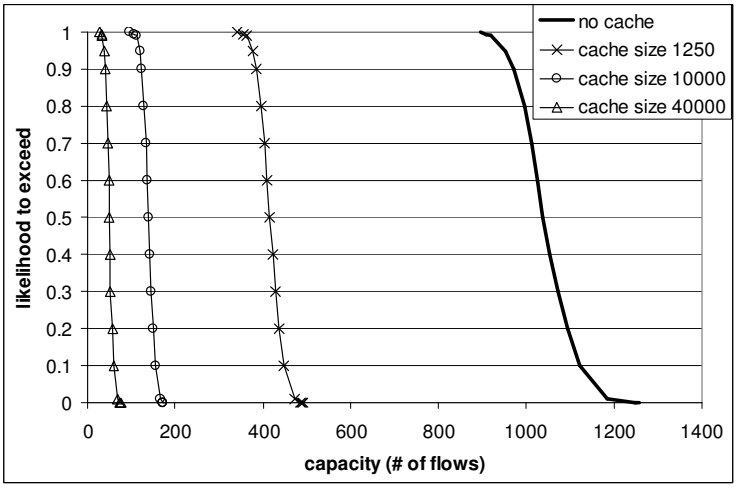
This is corroborated by the results shown in Figure 13 and Figure 14. They display the CCDF graphs of the required bandwidth resources (respectively uplink the DSLAM - Figure 13 or uplink the SR - Figure 14). The case when there is no cache (represented by the thick full line) is compared versus three cases of the cache size respectively L = 1250, 10000, 40000. The table associated to the graph gives the average and the coefficient of variation (CoV) of the capacity demand, as well as the HR and UR (where this applies). Notice that the average saved bandwidth can be calculated by multiplying the HR by the average required capacity in the case with no

cache. The variation of the aggregate capacity (expressed in the CoV) is also always smaller with N=4000 than with N=1000 (the law of large numbers postulates that the average increases linearly, i.e., in this case proportionally to N, while the second moment increases proportionally to $\sqrt{N}$ resulting in a decreasing CoV with growing N). Moreover, since the SR can usually host a larger cache than a DSLAM, this can result in a higher HR.



L	0	1250	10000	40000
average	266.89	110.94	50.08	29.28
CoV	0.07	0.10	0.14	0.18
HR		0.58	0.81	0.89
UR		1.23	1.61	0.94

Figure 13 CCDF of the capacity demand on the aggregation link towards the DSLAM with N=1000 - without cache and with cache.



L	0	1250	10000	40000
average	1044.35	416.11	139.27	50.06
CoV	0.06	0.06	0.09	0.14
HR		0.60	0.87	0.95
UR		1.15	4.48	1.61

Figure 14 CCDF of the capacity demand on the distribution link towards the SR with N=4000 - without cache and with cache.

Finally, note that caching in the DSLAM only makes sense, if the cost of the cache is compensated by the gain in required transport capacity on the aggregate link. Similarly, caching in the SR makes sense if the cost of the cache is compensated by the gain in required transport capacity on the distribution network and if the increase in capacity

on the aggregation network downlink to the DSLAM does not represent an issue. We do not discuss this cost trade-off in this paper.

7. CONCLUSIONS

In this paper we have shown that the transport capacity required to support CUTV services risks to grow enormously with the growing popularity of these services because of their nature to deliver individual streams (through unicast flows). Caches deployed in strategic places in the network controlled by good caching algorithms can alleviate this large increase in traffic volume. The cache performance, expressed by its Hit Ratio (HR) and its Update Ratio (UR), depends on a good caching algorithm, tuned to track well the real objects' request rates.

Therefore, we first presented results on characterising the video popularity evolution (determining the request rate). We modelled a set of monitored video views traces (7 to 10 months long) by a function with three parameters: one pertaining to the potential video object audience; the average decay time; the form of the views distribution tail. 92% of our traces were found to have a power-law form decay, while only 8% follow an exponential video popularity decay. However, as we demonstrate, modelling the cache HR by simply dividing the parameter set into two clusters grossly underestimates the actual HR. This motivated us for further attempts to better classify the traces and we tried to better cluster their parameters.

The impact of the tuning parameter of the proposed cache algorithm has been explored and the superiority of this caching algorithm has been demonstrated in comparison to two other caching algorithms: LRU and LFU. A simulator for a perfect caching algorithm has been constructed and the performance of the proposed caching logic has been studied when defining in various ways the input video parameters (observing in this way the impact of video parameters clustering).

Finally, we briefly presented considerations on the possible cache locations (in the distribution and aggregation network). A cache placed in the SR is slightly better (in terms of HR), because due to the fact that it serves more users, it has a better estimate of the request rate at its disposal. However, a cache in the SR does not impact the

traffic on the feeder link towards the DSLAM, while a cache in the DSLAM reduces the amount of traffic both on the feeder link towards the DSLAM and the SR.

8. ACKNOWLEDGMENTS

This work was carried out in the framework of the Q-Match project sponsored by the Flemish Institute for the Promotion of Scientific and Technological Research in the Industry (IWT).

9. REFERENCES

[1] ATIS Standard. 2007. IPTV High Level Architecture (ATIS-0800007).

[2] Avramova, Z., De Vleeschauwer, D., Wittevrongel, S., and Bruneel, H. 2009. Analysis and Modeling of Video Popularity Evolution in Various Online Video Content Systems: Power-Law versus Exponential Decay. In Proceedings of The First International Conference on Evolving Internet INTERNET 2009, (August 2009, Cannes, France), 95-100.

[3] Avramova, Z., De Vleeschauwer, D., Wittevrongel, S., and Bruneel, H. 2009. Capacity Gain of Mixed Multicast/Unicast Transport Schemes in a TV Distribution Network. In IEEE Trans. on Multimedia, vol. 11, (August 2009), 918-931.

[4] Baggio, A., Van Steen, M. 2005. Distributed Redirection for the World-Wide Web. Computer Networks, vol. 49(6), (2005), 743-765.

[5] Breslau, L., Cao, P., Fan, L., Phillips, G., and Shenker, S. 1999. Web Caching and Zipf-like Distributions: Evidence and Implications. In Proceedings of the Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM99), vol. 1, (New York (NY), USA, March 21-25, 1999), 126-134.

[6] Cárdenas, L. G., Gil, J.A., Domènech, J., Sahuquillo, J., and Pont, A. 2005. Performance comparison of a Web cache simulation framework. In Proceedings of the 19th International Conference on Advanced Information Networking and Applications (AINA05), vol. 2, (Taipei, Taiwan, March 28-30 2005), 281- 284.

[7] Chien, W.-D., Yeh, Y.-S., and Wang, J.-S. 2005. Practical Channel Transition for Near-VOD Services. IEEE Transactions on Broadcasting, vol. 51, no. 3, (September 2005), 360- 365.

[8] De Vleeschauwer, D. and Laevens, K. 2009. Performance of caching algorithms for IPTV on-demand services. IEEE Transactions on Broadcasting, Special Issue on IPTV in Multimedia Broadcasting, vol. 55 (2009), 491-501.

[9] Ho, K.-M., Poon, W.-F., and Lo, K.-T. 2007. Performance Study of Large-Scale Video Streaming Services in Highly Heterogeneous Environment. IEEE Transactions on Broadcasting, vol. 53, no. 4, (December 2007), 763-773.

[10] http://dynamic.abc.go.com/streaming/landing

[11] http://www.bbc.co.uk/iplayer/

[12] http://www.uitzendinggemist.nl

[13] Krogfoss, B., Sofman, L., and Agrawal, A. 2008. Caching architectures and optimization strategies for IPTV Networks. Bell Labs Technical Journal, vol. 13(3), 13–28.

[14] Nikolaus, B., Ott, J., Bormann, C., and Bormann, U. 2005. Generalized Greedy Broadcasting for Efficient Media-on-Demand Transmissions, vol. 51, no. 3, (September 2005), 354- 359.

[15] Poon, W.-F., Lo, K.-T., and Feng, J. 2005. Provision of Continuous VCR Functions in Interactive Broadcast VoD Systems. IEEE Transactions on Broadcasting, vol. 51, vo. 4, (December 2005), 460- 472.

[16] Rabinovich, M., Spatscheck, O. 2002. Web Caching and Replication. Addison Wesley, 2002.

[17] Shi, L., Gu, Z., Wei, L., and Shi, Y. 2006. An Applicative Study of Zipf's Law on Web Cache. International Journal of Information Technology, vol. 12, no. 4, 49-58.

[18] Verhoeyen, M., De Vleeschauwer, D., and Robinson, D. 2008. Content storage architectures for boosted IPTV service. Bell Labs Technical Journal, vol. 13(3), 29–43.

[19] Wauters T. et al. 2007. HFC access network design for switched broadcast TV services. IEEE Transactions on Broadcasting, vol. 53, (June 2007), 588-594.

[20] Hulu service: www.humu.com.