EUROPEAN TRANSACTIONS ON TELECOMMUNICATIONS
Eur. Trans. Telecomms. 2008; **19**:111–122
DOI: 10.1002/ett.1181

Communication Networks

Proxy caching algorithms and implementation for time-shifted TV services

Tim Wauters[1]*, Wim Van De Meerssche[1], Peter Backx[1], Filip De Turck[1], Bart Dhoedt[1], Piet Demeester[1], Tom Van Caenegem[2] and Erwin Six[2]

[1]*Department of Information Technology (INTEC), Ghent University—IMEC—IBBT, Gaston Crommenlaan 8, bus 201, B-9050 Ghent, Belgium*
[2]*Alcatel R&I, Access and Edge, Francis Wellesplein 1, B-2018 Antwerp, Belgium*

SUMMARY

The increasing popularity of multimedia streaming applications introduces new challenges in content distribution networks (CDNs). Streaming services such as Video on Demand (VoD) or digital television over the Internet (IPTV) are very bandwidth-intensive and cannot tolerate the high start-up delays and poor loss properties of today's Internet. To solve these problems, caching (the initial segment of) popular streams at proxies could be envisaged. This paper presents a novel caching algorithm and architecture for time-shifted television (tsTV) and its implementation, using the IETF's Real-Time Streaming Protocol (RTSP). The algorithm uses sliding caching windows with sizes depending on content popularity and/or distance metrics. The caches can work in stand-alone mode as well as in co-operative mode. This paper shows that the network load can already be reduced considerably using small diskless caches, especially when using co-operative caching. A prototype implementation is detailed and evaluated through performance measurements. Copyright © 2007 John Wiley & Sons, Ltd.

1. INTRODUCTION

During the last few years, the architectural model of access networks has evolved towards multi-service and multi-provider networks. Ethernet as well as full IP alternatives have been investigated as viable connectionless successors, for the legacy ATM-based platforms. While the introduction of Ethernet up to the edge (e.g. through VLANs) solves some the of the existing PPP problems, new ones are created. Traffic segregation issues, because of address resolution complications, prevent large-scale access network deployments and, therefore, an IP-aware network model [31] is often considered a valuable alternative.

One of the emerging services is television over IP. Currently, IPTV services are generally limited to Video on Demand (VoD, per TV program) and Broadcast TV (per TV channel). Video servers are typically located at the edge of the core network and put a heavy burden on the access networks in case of large deployments, possibly causing the network to congest. The solution proposed in this paper is to focus on time-shifted television (tsTV) and to deploy additional distributed caches and streamers in the aggregation nodes, co-operating on both a peer-to-peer and a hierarchical level. This approach offers an alternative for deploying dedicated home equipment for video storage, such as a home Personal Video Recorder (PVR), that has limited throughput capacity and is rather expensive. Time-shifted TV enables the end-user to watch a broadcasted TV program with a time shift, that is the end-user can start watching the TV program from the beginning, although the broadcasting of that program has already started or is already finished.

As shown in Figure 1, the popularity of a television program typically reaches its peak value within several minutes after the initial broadcast of the program and

---

* Correspondence to: Tim Wauters, University of Ghent, INTEC Gaston Crommenlaan 8, bus 201, B-9050 Ghent, Belgium.
E-mail: tim.wauters@intec.ugent.be

Received 23 December 2005
Revised 25 October 2006
Accepted 7 November 2006

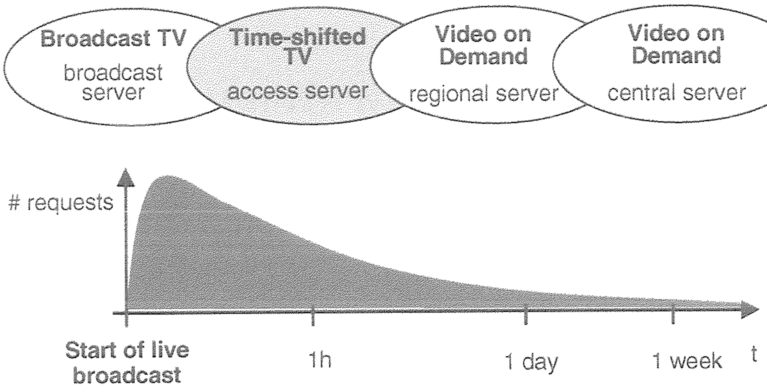
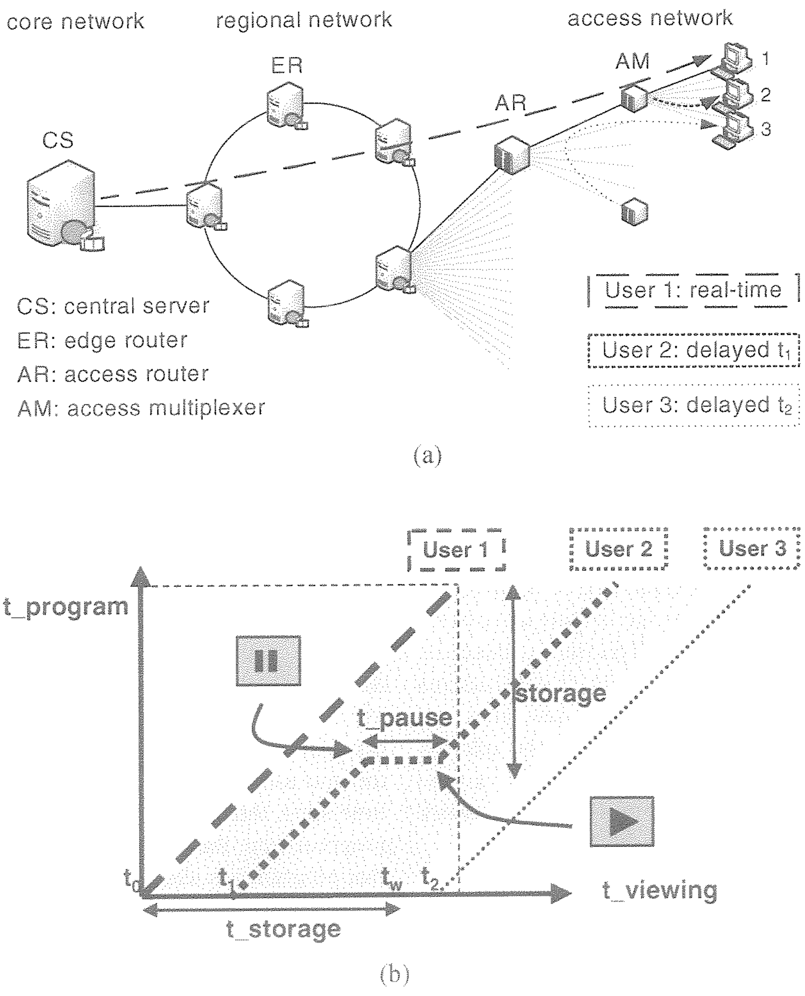


Figure 1. Delivery mechanisms for IPTV.



(a)

(b)

Figure 2. Time-shifted television: (a) typical network topology and (b) tsTV streaming diagram.

DOI: 10.10

exponentially decreases afterwards. This means that caching a segment with a sliding window of several minutes, for each current program can serve a considerable part of all user requests for that program from start to finish.

In Figure 2a, user 1 is the first to request a certain television program and gets served from the central server. Afterwards, other requesting users (e.g. user 2) can be served by the proxy, as long as the window of the requested program is still growing. After several minutes, the window stops growing and begins sliding, so that user 3 cannot be served anymore and will be redirected to the (central or regional) server or, in case of co-operative caching, to a neighbour proxy with the appropriate segment, if present. Another view of the same situation is given in Figure 2b. User 1 watches the program in real time, user 2 can be served within the cached window, while user 3 cannot. Pausing (parallel to the horizontal axis) can also be supported within the segment window, as well as fast forward or rewind (parallel to the vertical axis).

The above-mentioned servers can be distinguished as follows, based on their location and the streaming service offered:

- The *central server* stores all available content (TV programs), but typically only has to serve requests for less popular or older programs.
- The *regional server* generally stores more popular programs, such as recently broadcasted series or talk shows. These servers are located at the edge of the core network (see Figure 2a), thereby restricting the tsTV video traffic within the access networks.
- The *proxy servers* are located close to the users and only store fragments of the most popular content. Contrary to the traditional VoD technology service that would best fit the central and regional servers, the optimal technology choice for these proxy servers is to make use of a caching strategy, dynamically tailored to the tsTV service consumption pattern. Furthermore, interactive commands can be handled by these proxy servers, as long as the time-shifts remain within the stored segment window.

The remainder of this paper is structured as follows. Research work related to this research is briefly discussed in Section 2. Section 3 presents an analytical model of the sliding-interval caching problem with fixed window sizes, for comparison with our caching algorithms and to have an initial estimate of the required storage space. The next section introduces our sliding-interval caching algorithm, for both stand-alone and co-operative caching. It determines the location and the size of the different segments at the proxy caches, at run-time. In Section 5, the Real-Time Streaming Protocol (RTSP) implementation is studied more detail and evaluated through measurements. Sectio concludes this paper and presents ideas for future work.

2. BACKGROUND AND RELATED WORK

In this paper, a greedy sliding-interval co-operative cachi algorithm will be presented. This section gives an overvi of existing solutions in research and explains this choice

Previous studies on proxy caching techniques [2] or d tributed replica placement strategies for content distributi networks (CDNs) [3–8] show that greedy algorithms tl take distance metrics and content popularity into accou perform better than more straightforward heuristics, such least recently used (LRU) or least frequently used (LFU

Segment-based caching techniques have been studi extensively for streaming media, due to the huge size multimedia streams, compared to traditional web objec A survey on different strategies such as prefix cachi [9], segment caching [1, 10–12], rate-split caching [1 and sliding-interval caching [14, 15] has been present in Reference [2]. The main goal of prefix caching is reduce the start-up delay by caching the initial portion the stream at the proxy. This paradigm is generalised l segment caching, where cache decisions are made for series of segments of the stream. In rate-split caching, t partitioning is done along the rate axis, instead of along t time axis. This way, the cache takes care of the peak rat in VBR streaming, while the backbone only has to co with the lower constant rate.

Of particular interest for this study is sliding-interv caching, where the cached portion of the stream is initial a growing prefix, but afterwards, a dynamically updat sliding interval. This way, consecutive requests can l served from start to finish within this window. Since mc tsTV service requests are expected to arrive early, after tl start of the initial broadcast of a prime-time program, window size of several minutes can be sufficient. Anoth advantage of this technique is the support of interacti operations such as pause, fast-forward and rewind, at lea within the segment interval.

A more advanced aspect is the use of co-operati proxy caching [16–18], where a better performance th with independent proxies can be achieved through lo balancing and improved system scalability. In this case, it important to continuously keep track of cache states. No that contrary to standard co-operative proxy caching, the is no need to switch to segments on other proxies, wh using co-operative proxy caching with sliding interval

DOI: 10.1002/e

Similar peer-to-peer caching techniques have also been introduced in streaming CDNs, where whole files are stored instead of segments [19].

Several studies such as Reference [20] have been investigating the implementation of segment-based caching techniques on proxies using the RTP/RTCP/RTSP protocol suite.

The originality of this work is in the combination of the abovementioned techniques, applying the p2p and caching mechanisms from previously studied VoD content placement algorithms to sliding-interval caching. The proposed storage model is evaluated and implemented for IPTV, as a novel time-shifted TV service. The RTSP protocol allows for transparent request forwarding, which further optimises the content placement by creating one large virtual cache.

3. ANALYTICAL APPROACH

Before presenting our sliding-interval caching algorithm, we introduce an analytical model of a tsTV solution based on sliding-interval caching with fixed window sizes, offering a method to estimate the required storage space in the network.

3.1. Model parameters

Consider a model where each TV program is characterised by a start time τ_i, a duration T_i and a function $\lambda_i(t)$, representing the request arrival rate for this program. $N(t)$ denotes the total number of programs with $\tau_i \leqslant t$. The proxy cache I, placed between the server and the clients, contains the first X min of any currently streaming file with $t - T_i \leqslant \tau_i \leqslant t$.

3.2. Cache hit rate

We derive an expression for the hit rate of cache I, $h_I(t)$. Consider further the time period $]t, t + \Delta t[$, then the total number of requests is given by

$$\sum_{i=1}^{N(t)} \lambda_i(t)\Delta(t).$$

To find the total number of successful requests (i.e. requests that can be served by the cache) for the currently broadcasted program j in a single channel situation, we assume a uniform distribution for τ_j and make the following observations:

- these requests have to arrive at most X minutes after τ_j
- only a fraction X/T_j of the requests is served from cache I

Therefore the total number of successful requests i given by

$$\lambda_j(t)\Delta(t)\frac{X}{T_j}$$

Averaging over all programs j for which $t - X \leqslant \tau_j$ ≤ t, multiplying by the total number of channels K an supposing that popularity and duration are uncorrelated, w obtain the following expression:

$$h_I(t) = K\frac{<\lambda_i(t)>*}{<T>}\frac{X}{\sum\limits_{i=1}^{N(t)}\lambda_j(t)},$$

with <>* denoting averaging, on the condition that $t - X$ ≤ $\tau_j \leqslant t$. Supposing further that λ_i is a separable function o i and t, such that $\lambda_i(t) = \lambda_i f(t - \tau_I)$, with $f(t)$ a normalise function such that $f(t) = 0$ for $t < 0$ and

$$\int\limits_0^\infty f(t)\mathrm{d}t = 1,$$

we can write:

$$<\lambda_j(t)>* = <\lambda_j><f(t - \tau_j)>*$$

$$= \frac{<\lambda_j>}{X}\int\limits_0^\infty f(t)\mathrm{d}t$$

as long as $X << T>$. Hence,

$$h_I(t) = K\frac{<\lambda>}{<T>}\frac{\int_0^X f(t)\,\mathrm{d}t}{\sum\limits_{i=1}^{N(t)}\lambda_i(t)}$$

Further consider a time period P, then the total numb of broadcasted programs is $N(P) = KP/<T>$. Suppose user group of size G, each requesting r programs per seco on average, then the total number of requests is given b GrP. Therefore, the average number of requests for a lo enough period of time will satisfy

$$<\lambda>= \frac{GrP}{N(P)} = \frac{GrP}{KP/<T>} = \frac{Gr<T>}{K}$$

On the other hand, the total number of requests per tin unit is given by

$$\sum_{i=1}^{N(t)} \lambda_i(t) = Gr,$$

simplifying our expression for the cache I hit ratio to

$$h_I = \int_0^X f(t)\mathrm{d}t$$

Taking for $f(t)$ an exponentially decreasing function $b\exp(-bt)$ (for $t > 0$), we get

$$h_I = 1 - e^{-bX}$$

as long as $X << T >$. The size of cache I is simply KX.

3.3. *Example results*

Figure 3 shows the server load for different values of the cached segment size. If the content popularity only decreases slowly (e.g. by 10% after each interval, $b = -\ln(0.9)/\Delta$), the server load cannot be reduced significantly. When the content popularity is halved after each interval $\Delta (b = -\ln(0.5)/\Delta)$, the server load is halved as well when the segment size is Δ (Figure 3). It is then given by

$$1 - h_I = \left(\frac{1}{2}\right)^a,$$

if $X = a\Delta$.

Similar results for the server load can be found using the sliding-interval caching algorithm presented in the following section (compare the curve for $b = -\ln(0.5)/\Delta$ to the 's->c1' curve in Figure 10a, for stand-alone caches at level 2 and a halved content popularity after Δ).

4. SLIDING-INTERVAL CACHING ALGORITHM

Our caching algorithm for tsTV services is presented in this section. Since we assume that, in general, only segments of programs will be stored, cache sizes can be limited to a few gigabytes (corresponding to a few hours of streaming content). This way smaller streaming servers can be deployed closer to the users, without increasing the installation cost excessively.

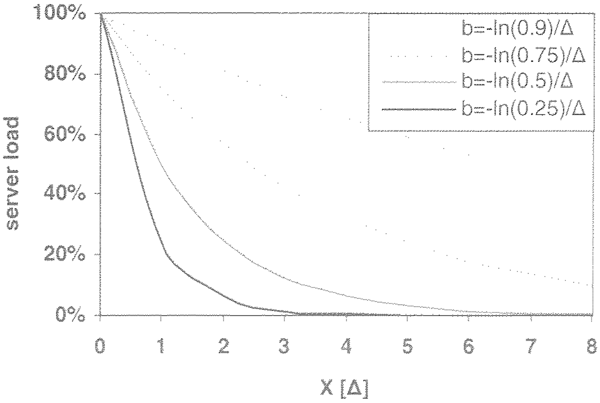


Figure 3. Analytical solution for the server load, for different values of the segment size.

4.1. *Basic principle*

The cache will be virtually split up in two parts: a small part S and a main part L. Part S will be used to cache the first few (e.g. 5) minutes of every newly requested (or broadcasted) program, mainly to determine its initial popularity. Its size is generally smaller than 1 GB (typically 1 h of streaming content).

Part L will be used to store the segments (with growing or sliding windows) of the currently most popular programs. The actual size of each segment in part L will be determined and, if necessary, adapted after each interval Δ (e.g. 5 min). During Δ, the cache is learning about the popularity of the programs.

Figure 4 shows the basic principle of the tsTV caching algorithm. During each interval Δ, program requests arrive at the different proxies. Each time, a parameter $A_{n,p}$ will be updated in proxy n, for program p. In general, this parameter tries to determine the popularity of the program, while taking distance metrics into account.

This means that a (segment of a) popular program might not be cached, because a nearby proxy already stores that (segment of the) program. $A_{n,p}$ is calculated as follows:

> *Every time a request for program* p *arrives at proxy* n, $A_{n,p}$ *is increased by 1 (popularity only) or by the hopcount between proxy* n *and the serving node (popularity and distance).*

After each interval Δ, first all segments (sliding or growing) with status set to 'occupied' are stored in L. Afterwards, L is filled with segments with growing

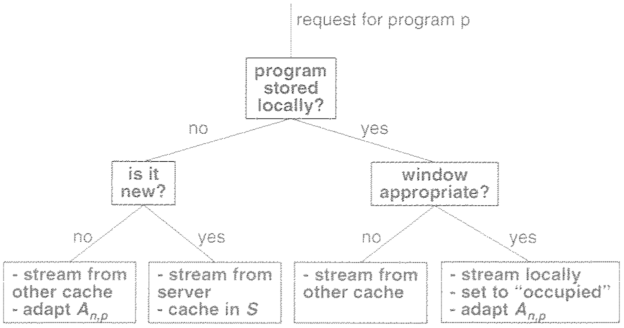
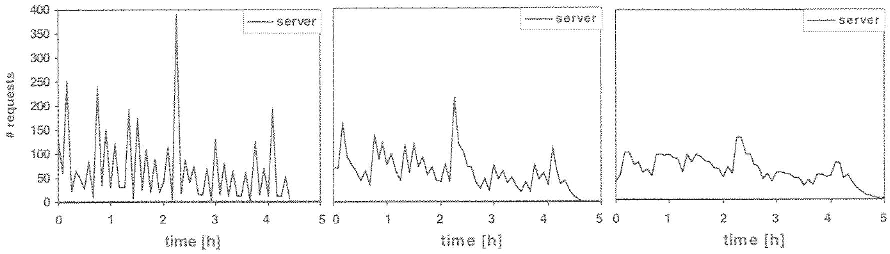


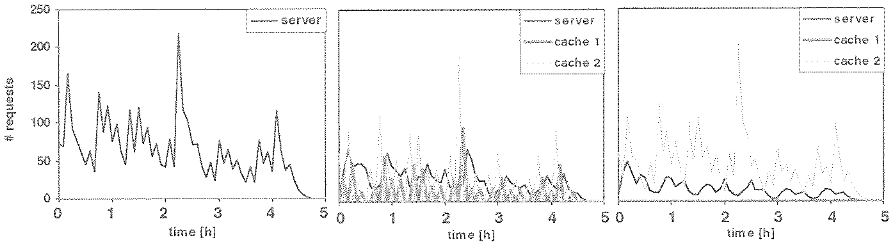
Figure 4. Basic principle of the tsTV caching algorithm at each proxy.

DOI: 10.1002/ett



(a) (b) (c)

Figure 5. Server load without caches. All requests per program are made within 5(a), 30(b) or 60(c) min. 3000 requests are mad
total.



(a) (b) (c)

Figure 6. Server load with caches. All requests are made within 30 min. The cache sizes are 0 GB (a), 0.5 GB (b) and 4 GB (c).

windows for the most popular programs (i.e. with the highest values of $A_{n,p}$). All other segments are dropped, S is cleared and all values of $A_{n,p}$ are reset to 0.

The influence of premature termination, for example due to channel hopping, on the caching behaviour is much smaller than for the storage of whole video files [21], since the small cache part S, which is cleared after every learning interval Δ, handles most of these specific requests.

4.2. Numerical results for stand-alone caching

4.2.1. Input parameters

To illustrate the caching principle, a first set of simulations was performed on one branch of the access network tree of Figure 2a: a regional server with two hierarchical caches (Figure 7). The regional server offers 20 channels: 5 very popular channels (80% of all requests), 5 less popular channels (10% of all requests) and 10 unpopular channels (10% of all requests). The top five channels are served as a tsTV service, the other channels through standard VoD technology on the regional server.

The popularity of the programs per channel follows a Zipf-like distribution with parameter $\beta = 0.7$ (the popularity of the i'th most popular program is proportional to $i^{-\beta}$). A total of 3000 requests are made during one evening, of which 200, for the most popular program
the most popular channel.

The popularity of a program reaches a peak du
the first interval Δ (=5 min) and decreases exponenti
afterwards (halved every interval Δ) (similar to Figure
Each channel offers six programs of 45 min per even
with a streaming bandwidth of 2.5 Mbps (1 GB per hou

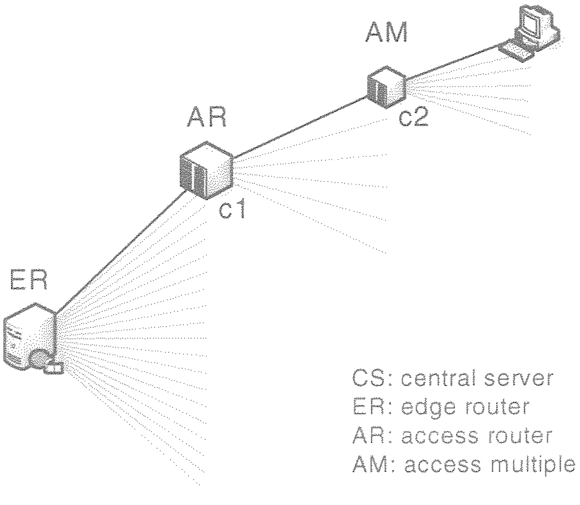


Figure 7. Basic access network topology.

Note that the values for these input parameters are very general, since Zipf-like content popularities are very commonly used in content distribution networks [22] and the relative results below are independent of the number of TV programs or their duration or bandwidth.

The caching algorithms have been implemented in C++ and evaluated in a standard discrete event simulator using the LEDA [23] library.

4.2.2. *Server and cache load*

In case no caches are in use, the load on the regional server is shown in Figure 5 (cumulated per interval Δ). The longer the period during which all requests are made, the smoother the traffic at the server (the total number of requests and the exponential decrease remain the same, while the initial popularity is different).

In Figure 6, caches are introduced. When both cache sizes are limited to 0.5 GB (*S* only: the number of channels times Δ or 25 min), the server load is already much lower and the caches serve most of the tsTV requests. What happens is that cache 1 (closest to the server) and cache 2 first store all 5-min prefixes of each new program, but since only cache 2 receives new requests afterwards, cache 1 will drop these segments after Δ. Afterwards cache 1 will store the next 5 min of each program, while cache 2 is storing the sliding 'occupied' windows from the first interval. This means that the caches serve all requests made during the first 10 min of each single program.

For infinite cache sizes (or 4 GB or higher in this example), the regional server only serves the VoD requests for channels 6–20. Cache 2 stores and serves all currently broadcasted programs.

More detail on the regional server and cache load is given in Figure 8 (tsTV only, top five channels). Note that the server load never drops to 0, since at least the first request for a certain program has to be served from the regional server. In Figure 9, the server load is shown for different values of the maximum request period per program. Since no upstream links are used in these simulations, the bandwidth on the links can easily be determined from the server and cache load.

4.3. *Numerical results for co-operative caching*

The same caching principles can be applied for a co-operative caching mechanism, where caches on the same level of the broadcast tree can collaborate, using peer-to-peer protocols to exchange information on stored content. Contrary to stand-alone caching, where a request that cannot be served is forwarded to the next cache on the path to

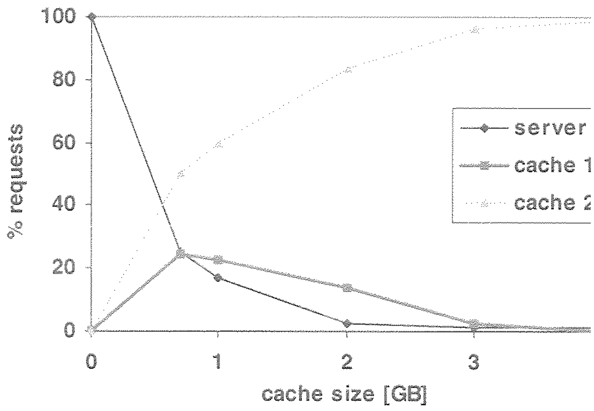


Figure 8. Server and cache load. All requests are made wit
30 min.

the central server (hierarchical caching), caches can no
forward requests to caches on the same level. Howev
the decision on when to store a certain fragment not on
depends on the value of $A_{n,p}$, but also on the source no
serving the request. Two different approaches have be
implemented.

The first heuristic only takes the values of $A_{n,p}$ into a
count ('cache from all sources', *CfA*). This means that mo
caches store the same fragments, since content popularity
similar for most nodes. The numerical results will therefo
be comparable to the results for stand-alone caching.

The second heuristic also takes the values for $A_{n,p}$ in
account, but never stores content that is already stored
another cache ('cache from server only', *CfS*). This wa
the central server will be offloaded considerably, even wi
small caches, but many requests will have to be served
other caches over the access network links.

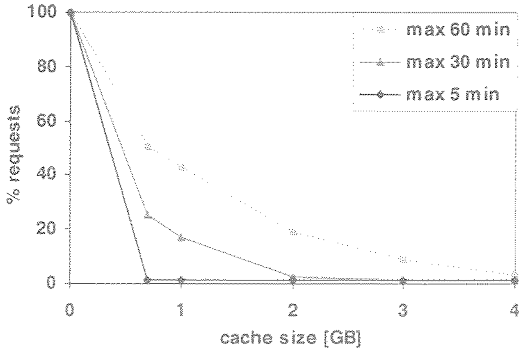


Figure 9. Server load for different values of the maximum reque
period.

DOI: 10.1002/

Both alternatives have their benefits (the first one is optimal in case of larger caches, the second one in case of small caches). The optimal heuristic, however, takes the best of both worlds, storing unique content segments in one part of cache L (called L_1, used in the *CfS* heuristic) and locally popular segments in another part (called L_2, used in the *CfA* heuristic).

This way, the central server load is always minimised first: the expected server load using the storage space combining all parts L_1 can then be determined out of Figure 3. The access network load can be reduced afterwards, if the cache space is large enough. This heuristic is called 'cache from elected sources' (*CfE*).

4.3.1. *Input parameters*

The input parameters for the simulations are the same as in the previous section. The network topology (similar to Figure 7) now consists of a central server, one node at level 1 (without storage capabilities) and six proxy caches at level 2. The level 1 node is connected to the level 2 caches with bidirectional links, so that cache co-operation is possible.

Note that no storage space is available at the level one node so that the results of the simulations for cache co-operation are not influenced by hierarchical caching.

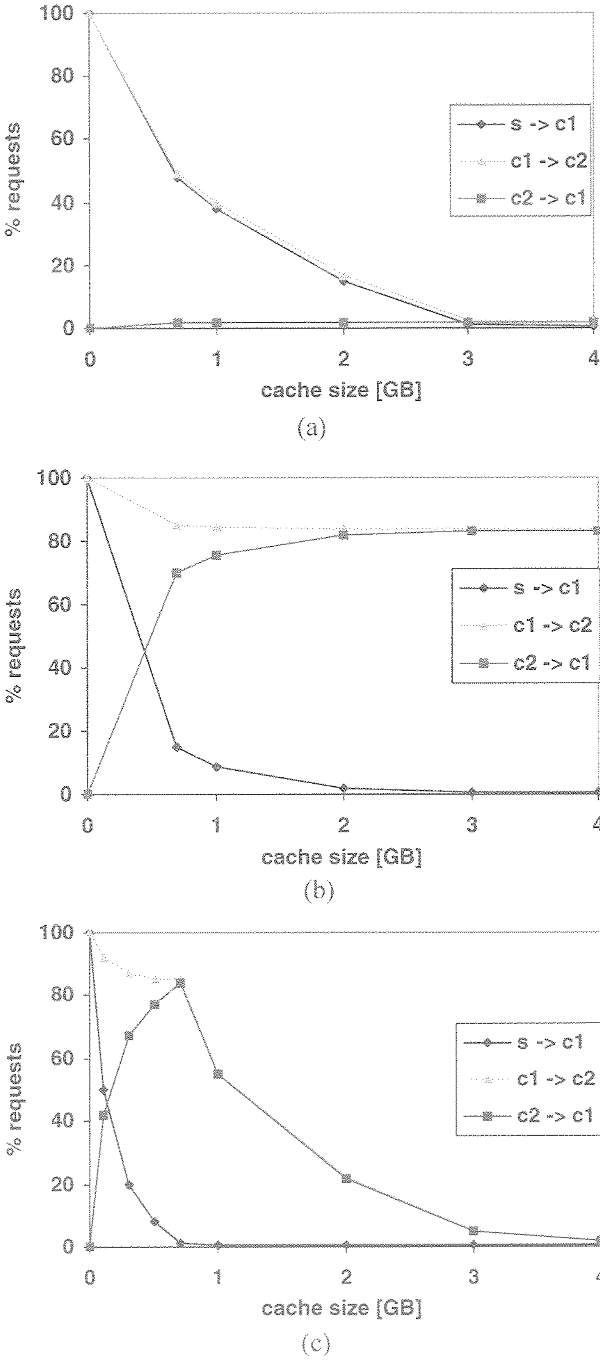
The cost of using the link from the central server to the node at level 1 has been set to 10 (in fact, any value higher than 1 will do), instead of 1. This way, the central server will be avoided when the requested segment can already be found on a neighbour level 2 cache (when calculating the shortest path, using the weighted Dijkstra algorithm).

4.3.2. *Server, cache and network load*

In case of stand-alone caching, the network bandwidth can easily be determined out of the cache and server load (Figure 8), since only downstream traffic is present on the access network. With co-operative caching, the uplinks in the access network are used as well.

Using the *CfA* heuristic (Figure 10a), the server load is almost identical to the case where stand-alone caches on level 2 are used. The only difference is that the central server does not need to serve the first stream to all of the six proxies, but only to one of them. Again the central server load for the tsTV channels drops to (almost) zero when 4 GB caches would be used. The uplinks from the level 2 caches to node 1 are almost never used, since all caches store the same fragments. The results are, therefore, very similar as for stand-alone caching (remember the analytical results of Figure 3, with 1 GB = 10 min per channel = 2Δ).

When the *CfS* heuristic is used (Figure 10b), each 5-min (Δ) fragment is only stored on one cache. This way,



(a)

(b)

(c)

Figure 10. Fraction of the streams on the links between the serv and the level 1 node (s -> c1) and between the level 1 and two nod (downlink c1 -> c2 and uplink c2 -> c1) for the CfA (a), CfS (and CfE (c) heuristics.

the central server load is already almost zero for the tsTV channels, when only 0.5 GB caches are used. The total storage space is then 3 GB, therefore, one could expect that the results for the central server load would correspond to the situation with 3 GB caches in stand-alone mode. This is not entirely the case, since it is possible that the first requests for a new program arrive at caches that have no storage place left in L_1. These first requests are then served by the central server. The 'core network load' (represented by the link 's -> c1') is reduced considerably, while the 'access network load' (represented by the links 'c1 <-> c2') is load balanced.

The *CfE* heuristic (Figure 10c) offers the best of both worlds. The server load is reduced effectively, while, in case of larger caches, the access network is offloaded as well. The server load (link 's -> c1') is even lower then for the *CfS* heuristic. This is due to the RTSP request forwarding mechanism, allowing requests that arrive at a cache that has no storage space left in L_1, to be forwarded automatically to another cache with enough storage space. This way the virtual cache consisting of all parts L_1 is filled up in an optimal way.

5. PROXY IMPLEMENTATION

A transparent RTSP proxy for time-shifted TV has been implemented (in C++) for evaluation purposes. This section gives an overview of the different components and protocols used and evaluates a prototype through performance measurements.

5.1. Functionality

In order to implement the proxy, its functionality is divided into logical parts. The communication with the users and the central server includes messages containing data about which program or channel has to be streamed, or VCR like commands such as PAUSE and STOP. A protocol, commonly used for this interaction is RTSP [24]. The streams themselves are encapsulated and delivered with Real-Time Protocol (RTP), a standard protocol for live streamed media [25].

A first functional component of the proxy is the *RTSP Proxy*, a component that communicates with the tsTV clients and the server using RTSP, interprets their messages and commands the other components to execute these requests. The *RTSP Proxy* component delegates the caching algorithm decisions to another component, the *Cache Verdict Manager*, a component that uses information from the *Cache State Manager*, which is updated through a centralised or distributed Cache State Exchange (CSE) protocol. The task of the *Cacher* component is to sto popular streams, sent to the proxy by the server (or anoth cache), in sliding windows. The streams are sent to t clients from these windows, a function that is handled the *Streamer* component. The proxy also keeps track of t streams that are being sent to the proxy (which progra channel, starting time, …), through the *Stream Track* component, with help from the *Program Guide* compone which communicates with the electronic program gui (EPG) server. The *Packet Handler* acts as an interfac dealing with low-level network interaction. Figure 11 giv an overview of the different components.

5.2. Detailed scenario

Figure 12 shows a detailed setup of a streaming sessi between the client, the proxy caches and the server.

First, the client sends an RTSP request to the server, b this request is intercepted by the proxy. In a first scenar (part 1a in Figure 12), the proxy does not store the request fragment, forwards the request (with the destination address of the proxy) to the server, starts caching the strea from the server and forwards the RTP stream to the us Afterwards, the proxy exchanges its new cache state in distributed way to all other caches (part 2a in Figure 12). a second possible scenario (part 1b in Figure 12), the pro does not store the requested fragment and decides not store the fragment locally. It forwards the RTSP request another (proxy) cache, keeping the destination IP addre of the client.

The other proxy decides to forward the request to t server, caches the fragment locally and sends the RT stream directly to the client. Afterwards, the new cac states are exchanged through a centralised CSE protoc

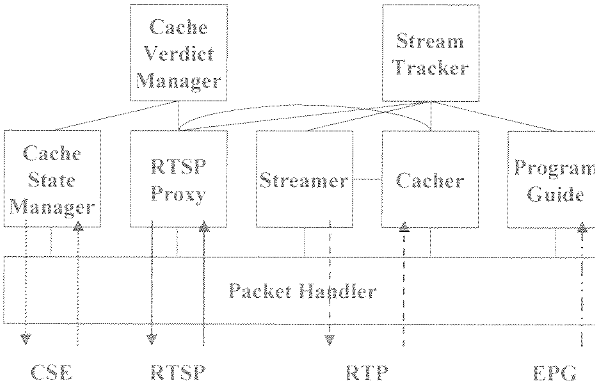


Figure 11. Overview of the different components in the pro cache.

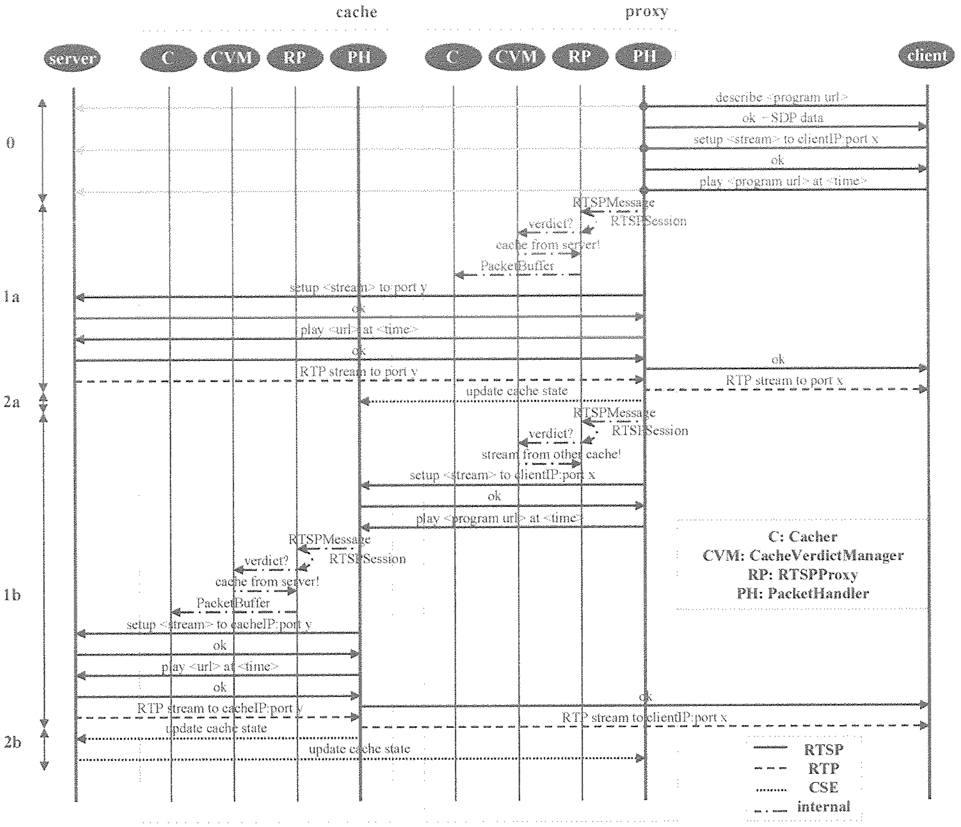


Figure 12. Detailed setup of a streaming session between client, proxy, any other cache and the server. In scenario a, the proxy cach the requested program from the server; in scenario b, the proxy forwards the RTSP request transparently to another cache.

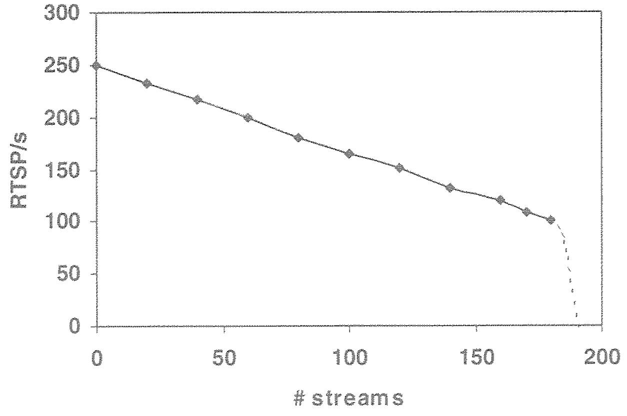


Figure 13. RTSP requests handling (AMD AthlonTM 64 processor).

(Figures 12 and 2b). The second scenario shows how the co-operative caching algorithm (Section 4.3) can efficiently create one large virtual cache, using the 'transparent RTSP request forwarding' principle.

5.3. *Test setup and measurements*

In this section, performance measurements on a prototyp proxy are presented, implemented on an AMD AthlonT 64 processor 3000+ (512 MB RAM). Figure 13 show the number of client RTSP requests that can be handle simultaneously by the proxy, already serving RTP stream (2.5 Mbps) over a gigabit link (560 Mbps throughpu measured with Iperf [26]). The proxy uses high-priorit RTP threads and low-priority RTSP threads. We observ that the RTSP handling decreases linearly and fails 190 simultaneous RTP streams (480 Mbps), due to limite system resources. Figure 14 shows the delay between PLAY request sent by a PC client and the arrival of the fir RTP packet at the PC client, for different configuratio (server-proxy-client). Even when the proxy has to fetch th content from the server, the delay is never higher than 35 m (1000 measurements per configuration). When the prox acts as a mere router, the delay caused by the server (Darw streamer [27]) is less than 1 ms. The delay on the netwo links between server, proxy and client is negligible.

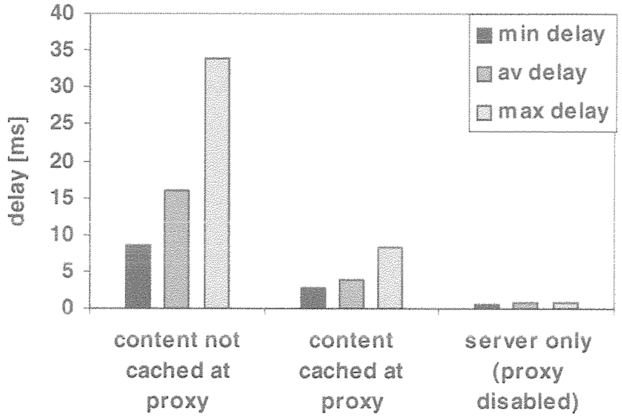


Figure 14. Delay between a client request and the actual start of the RTP stream on a client PC.

6. CONCLUSIONS

In this paper, a novel sliding-interval caching algorithm for a tsTV service was presented. Cache decisions (segment size, stored programs, ...) at low cost distributed streamers are made after each learning interval Δ, based on popularity and distance metrics. Experimental results for a basic network topology showed promising results in terms of server and network load, especially for co-operative caching. An RTSP proxy implementation has been introduced as well. A prototype, integrating the caching algorithms has been built and evaluated through measurements.

Future work includes the introduction of E2E resilience aspects (e.g. RTP retransmission [28]) and other concepts such as storage of content at the user premises, possibly served through peer-to-peer content streaming, will be investigated as well.

ACKNOWLEDGMENT

This work is partly funded by the IST FP6 MUSE project. MUSE contributes to the strategic objective 'Broadband for All' of IST (Information Society Technologies) and it is partially funded by the European Commission.

REFERENCES

1. Miao Z, Ortega A. Scalable proxy caching of video under storage constraints. *IEEE Journal on Selected Areas in Communications* 2002; **20**(7):1315–1327.
2. Liu J, Xu J. Proxy caching for media streaming over the internet. *IEEE Communications Magazine* 2004; **42**(8):88–94.
3. Karlsson M, Mahalingam M. Do we need replica placement algorithms in content delivery networks? In Proceedings of the 7th International Web Content Caching and Distribution Workshop, August 2002.
4. Qiu L, Padmanabhan VN, Voelker GM. On the Placement of Web Server Replicas. *IEEE Infocom* 2001—The Conference on Computer Communications, no. 1, April 2001, pp. 1587–1596.
5. Karlsson M, Karamanolis C, Mahalingam M. A Framework for Evaluating Replica Placement Algorithms. Technical Report HPL-2002, HP Laboratories, July 2002.
6. Kangasharju J, Roberts J, Ross K. Object replication strategies in content distribution networks. *Computer Communications* 2002; **25**(4):376–383.
7. Wauters T, Coppens J, Dhoedt B, Demeester P. Load balancing through efficient distributed content placement. Conference proceedings of NGI 2005, April 2005, Rome, Italy.
8. Coppens J, Wauters T, De Turck F, Dhoedt B, Demeester P. Evaluation of Replica Placement and Retrieval Algorithms in Self-Organizing CDNs. Conference proceedings of IFIP/IEEE International Workshop on Self-Managed Systems & Services SelfMan 2005, May 2005, Nice, France.
9. Sen S, Rexford J, Towsley D. Proxy prefix caching for multimedia streams. IEEE INFOCOM 1999—The Conference on Computer Communications 1999; **1**:1310–1319.
10. Chen S, Shen B, Wee S, Zhang X. Designs of high quality streaming proxy systems. IEEE INFOCOM 2004—The Conference on Computer Communications 2004; **23**(1):1513–1522.
11. Wu K, Yu P, Wolf J. Segment-based proxy caching of multimedia streams. In Proceedings of World Wide Web Conference WWW10, Hong Kong, 2001.
12. Fahmi H, Latif M, Sedigh-Ali S, Ghafoor A, Liu P, Hsu L. Proxy servers for scalable interactive video support. *IEEE Computer* 2001; **43**(9):54–60.
13. Zhang Z, Wang Y, Du D, Su D. Video staging: A proxy-server-based approach to end-to-end video delivery over wide-area networks. *IEEE/ACM Transactions on Networking* 2000; **8**(4):429–442.
14. Tewari R, Vin H, Dan A, Sitaram D. Resource-based caching for Web servers. In Proceedings of SPIE/ACM Conference of Multimedia Computing and Networking MMCN'98, San Jose, CA, 1998.
15. Chen S, Shen B, Yan Y, Basu S, Zhang X. SRB: Shared running buffers in proxy to exploit memory locality of multiple streaming media sessions. In Proceedings of the 24th IEEE International Conference on Distributed Computing Systems (ICDCS), 2004.
16. Acharya S, Smith B. Middleman: A video caching proxy server. In Proceedings of 10th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'00), 2000.
17. Chae Y, Guo K, Buddhikot M, Suri S, Zegura E. Silo, rainbow, and caching token: Schemes for scalable, fault tolerant stream caching. *IEEE Journal on Selected Areas in Communications* 2002; **20**(7):1328–1344.
18. Hofmann M, Ng T, Guo K, Paul S, Zhang H. Caching techniques for Streaming Multimedia over the Internet. Technical Report, Bell Labs, 1999.
19. Turrini D, Panzieri F. Using p2p techniques for content distribution internetworking: a research proposal. In Proceedings of the 2nd IEEE International Conference on Peer-to-Peer Computing, September 2002.
20. Gruber S, Rexford J, Basso A. Protocol Considerations for a Prefix-Caching Proxy for Multimedia Streams. *Computer Networks* 2000; **33**(1–6):657–668.
21. Cahill A, Sreenan C. An Efficient CDN Placement Algorithm for the Delivery of High-Quality TV Content. 9th IASTED International Conference on Internet and Multimedia Systems and Applications (EuroIMSA), Grindelwald, Switzerland, February 2005.
22. Breslau L, Cao P, Fan L, Phillips G, Shenker S. Web Caching and Zipf-like Distributions: Evidence and Implications. *IEEE Infocom 1999*—The Conference on Computer Communications, no. 1, March 1999, pp. 126–134.
23. LEDA library, http://www.algorithmic-solutions.com/enleda.htm
24. RFC 2326, Real Time Streaming Protocol (RTSP).

25. RFC 1889. RTP: A Transport Protocol for Real-Time Applications.
26. Iperf, TCP/UDP bandwidth measurement tool, http://dast.nlanr.net/projects/Iperf.
27. Darwin Streaming Server, http://developer.apple.com/opensource/server/streaming
28. IETF draft on the RTP Retransmission Payload Format, http://www.ietf.org/internet-drafts/draft-ietf-avt-rtp-retransmission- 12.txt.
29. Akamai. http://www.akamai.com.
30. Mirror Image. http://www.mirror-image.com.
31. Gilon E, *et al.* Demonstration of an IP aware multi-service a network. Proceedings of BroadBand Europe 2005, December 2 Bordeaux, France.

AUTHORS' BIOGRAPHIES

Tim Wauters received his M.Sc. degree in electrotechnical engineering (option communication techniques) in 2001 from the Unive of Ghent, Belgium. Since September 2001, he has been working on the design of content distribution and peer-to-peer networks i Department of Information Technology (INTEC), at the same university. His work has been published in several scientific publica in international conferences and journals.

Wim Van De Meerssche received his M.Sc. degree in software development in 2004 from the University of Ghent, Belgium. In Au 2004, he started working on software technologies for access networks in the Department of Information Technology (INTEC), a same university. His work has been published in several scientific publications in international conferences.

Peter Backx received his PhD in Computer Science engineering at the University of Ghent in 2005. He specialized in compo based programs and their automatic and distributed deployment in wide-area networks. Since then he has been working at LogicaC Belgium consulting for a wide range of problems from Ajax enabled web frontends to optimizing network traffic of large scale busi applications.

Filip De Turck received his M.Sc. degree in Electronic Engineering from the Ghent University, Belgium, in June 1997. In 2002, he obtained the Ph.D. degree in Electronic Engineering from the same university. From October 1997 to September 2 Filip De Turck was research assistant with the Fund for Scientific Research-Flanders, Belgium (F.W.O.-V.). At the moment, a part-time professor and a post-doctoral fellow of the F.W.O.-V., affiliated with the Department of Information Technology o Ghent University. Filip De Turck is author or co-author of approximately 80 papers published in international journals or i proceedings of international conferences. His main research interests include scalable software architectures for telecommunic network and service management, performance evaluation and optimization of routing, admission control and traffic manageme telecommunication systems.

Bart Dhoedt received a degree in Engineering from the Ghent University in 1990. In September 1990, he joined the Departme Information Technology of the Faculty of Applied Sciences, University of Ghent. His research, addressing the use of micro-o to realize parallel free space optical interconnects, resulted in a PhD degree in 1995. After a 2 year post-doc in opto-electro he became professor at the Faculty of Applied Sciences, Department of Information Technology. Since then, he is responsibl several courses on algorithms, programming and software development. His research interests are software engineering and m & wireless communications. Bart Dhoedt is author or co-author of approximately 100 papers published in international journa in the proceedings of international conferences. His current research addresses software technologies for communication netw peer-to-peer networks, mobile networks and active networks.

Piet Demeester received the Masters degree in Electro-technical engineering and the Ph.D degree from the Ghent University, Belgium in 1984 and 1988, respectively. In 1992 he started a new research activity on broadband communication networks resu in the IBCN-group (INTEC Broadband communications network research group). Since 1993 he became professor at the University where he is responsible for the research and education on communication networks. The research activities cover va communication networks (IP, ATM, SDH, WDM, access, active, mobile), including network planning, network and service manage telecom software, internetworking, network protocols for QoS support, etc. Piet Demeester is author of more than 400 publicatio the area of network design, optimization and management. He is member of the editorial board of several international journal has been member of several technical program committees (ECOC, OFC, DRCN, ICCCN, IZS,...).

Tom Van Caenegem received an M.Sc degree in physical engineering in 1995 and a Ph.D degree in electrotechnical engineeri 2001 from Ghent University, Belgium. In 2001 he joined the Optical Access networks group in the Alcatel Research and Innov (R&I) Center in Antwerp, Belgium. Since 2002, he was active in systems engineering specialising in PON access and was involv FSAN contributing to the G-PON standardisation. In 2005 he joined the Network and Services Architecture subgroup of R&I Access division, where he is involved in IPTV related studies.

Erwin Six received his master's degree in electronic engineering, specialized in telecommunication, from the university of G Belgium in 2001. He joined Alcatel Research & Innovation (R&I) Center in Antwerp, working as a system engineer in the a technology group on Passive Optical Networks (PON). Afterwards, as a member of the network architecture team, he helped the definition of Alcatel's IP DSLAM and Home Device Manager product. Presently, his research is focused on advanced broad access equipment features, for next generation service delivery.

DOI: 10.10