[GZLOSt+00] *Carsten Griwodz, Michael Zink, Michael Liepert, Giwon On, Ralf Steinmetz;* **Multicast for Savings in Cache-based Video Distribution;** Multimedia Computing and Networking 2000. SPIE, January 2000.

Multicast for Savings in Cache-based Video Distribution

Carsten Griwodz[1][a], Michael Zink[a], Michael Liepert[a], Giwon On[a], Ralf Steinmetz[a,b]
[a]KOM, Darmstadt University of Technology, Merckstrasse 25, 64283 Darmstadt, Germany
[b]GMD IPSI, Dolivostr. 15, 64293 Darmstadt, Germany

ABSTRACT

Internet video-on-demand (VoD) today streams videos directly from server to clients, because re-distribution is not established yet. Intranet solutions exist but are typically managed centrally. Caching may overcome these management needs, however existing web caching strategies are not applicable because they work in different conditions.

We propose movie distribution by means of caching, and study the feasibility from the service providers' point of view. We introduce the combination of our reliable multicast protocol LCRTP for caching hierarchies combined with our enhancement to the *patching* technique for bandwidth friendly True VoD, not depending on network resource guarantees.

LCRTP is an RFC-conforming extension to the application-level protocol RTP that allows the receivers to allocate exactly the required space for lost data, and supports the retransmission after the initial transfer. However, without additional techniques, this does not save a lot of capacity in real life scenarios. To increase the savings we combine LCRTP with *patching*.

Patching saves server capacity in centralized systems. It works by streaming a video from start to end to the first client that requests this movie. Requests that follow in a limited temporal interval are served by transmitting sufficient information to join the initial stream, and an additional "patch" stream for the missing initial portion of the movie. These subsequent clients use local cyclic buffers to delay play-out of multicast portions of the movie. There is an optimal time before retransmitting a complete movie by multicast, mostly depending on the frequency of requests for a movie. *Patching* can also be applied recursively to patches.

This paper motivates the combined technique and details the elements of an implementation.

Keywords: Wide-Area Caching Architecture, Video on Demand, Internet

1 INTRODUCTION

Internet VoD today is mostly dominated by systems that serve only a small amount of users and have a small library of video clips instead of full size movies. The length and especially the quality of current video clips are very limited, and not applicable at all for commercial VoD. One of the major limitations of current systems is the necessity to stream the video clip directly from a central server to each client individually, because re-distribution is not established yet. Multicast as a transmission method can not be used efficiently in these systems and therefore bandwidth is wasted leading to higher network costs. Intranet solutions have existed for a while, they use distributed systems but are typically managed from a central site. Since they are used only locally savings that are achieved by using multicast are not as significant as they could be in larger systems.

We expect that the growth of the Internet, the support for partial resource guarantees in the backbone through Differentiated Services [24], and the integration of services will make the idea of wide-area distribution of commercial quality video over networks without central management feasible. Intelligent caching can be helpful in this for two reasons:

- movies have a considerable life cycle, i.e. typically their popularity increases steeply early after their release date, reaches a peak, and decreases towards a minimal residual popularity
- the movement of full-sized high quality movies among caches is severely more expensive then that of current internet videos because of the required bandwidth and storage.

We address these issues with the following efforts:

- Investigate more complex strategies and structures to position and access copies of expensive files. Since the number of requests that reach these caches is less than for web caches and requested files are mostly very large, we do not want to streamline caches mainly for simplicity and thus rapid request answers. This is unlike the current approach of the web caching communities to efficiently handle co-operative caches [1].
- Support the delivery of large bulk data files across networks with resource guarantees.
- Evaluate approaches like Hint-based caching [23] for applicability to movie caching as well.
- Support for untrusted caches [15].

1. Correspondence: Email: medianode@kom.tu-darmstadt.de

We want to specifically handle movie distribution by means of caching, and in this paper we present a feasibility study from the service providers' point of view. Our multicast distribution scheme assumes that packet loss is somewhat limited to achieve an acceptable to viewers of rare content. With Differentiated Services available to achieve virtual leased lines in the backbone, TCP will not remain the only appropriate protocol able to deliver streamed data with sufficiently low loss over long distances, we present LCRTP, which is an example approach. Some people argue that bandwidth will soon become abundant at least in the backbone and that high-quality unicast transmissions will become commercially feasible. We do not believe that this pipe dream will come true outside of research networks in the foreseeable future.

We found in the past that the *patching* approach by Hua et.al. [18] has provided an interesting means for saving server capacity in centralized systems.

We combine the ideas of movie caching and *patching* and demonstrate that the technical means have been developed already; non-withstanding the fact that the current elements to such a solution are very expensive or not in a product state yet. This leads to less network load and a better performing video server since the load can be dynamically optimized.

In the remainder of this paper we shortly introduce the referenced techniques of caching, *patching*, reliably multicasting and streaming from incomplete files. Then, exemplarily we calculate the different network costs for the unicast transmission and the enhanced transmissions with caching and *patching* to compare the network costs for the different techniques in order to show that caching and *patching* improve the performance of a distributed VoD architecture. Finally, we present our combination of *patching* with caching. We present our specific approaches for caching hierarchies of *patching*.

1.1 Caching

Caching can be used to reduce network costs as a) a stand alone version or b) in combination with multicasting. Network costs are reduced in a way that popular movies are stored in cache servers close to the requesting clients which leads to a shorter distance between source and client and also less network usage. These facts are well known from the use of caching in the web but nevertheless it must be mentioned that web and movie caching have some significant differences. Current internet caching strategies can remain rather simple, since the assumption of a large cache and small data items is valid for the vast part of web traffic. Furthermore, the distributed content is typically free or not commercially relevant. This permits to mostly ignore security and copyright issues as well. In contrast to web caching the content on video caches will in most cases not be free and the size of the data in comparison to the storage space is much bigger.

1.2 Patching

For the exploitation of multicast in TVoD systems, several approaches have been presented in the past. [3] introduces *batching*, which works by collecting requests that arrive within a certain cycle. At the end of the cycle they are serviced from the same file and buffer. [4] modifies this approach towards dynamic batching, which services requests as soon as a stream becomes available. [16] proposes *piggybacking*, which works by starting one stream for each request and subsequently joining streams of the same title that have been started in short sequence. The means is a speed increase of the later stream and/or a speed decrease of the earlier stream until they join. [6] and [7] introduce *content insertion* to force larger numbers of streams into a time window which is small enough to allow the use of the piggybacking technique. As content to be inserted, advertisements or extensions to introducing scenes are proposed as fill content.

A relatively new approach is *patching*, invented by Hua et.al. The basic approach, presented in [18], is the creation of a multicast group for the delivery of a video stream to a requesting client. If another client requests the same video shortly after the start of this transmission, this client starts storing the multicast transmission in a local cache immediately. The server sends a unicast stream to this client containing the missing initial portion of the video, until the cached portion is reached. Then, the client uses its cache as a cyclic buffer.

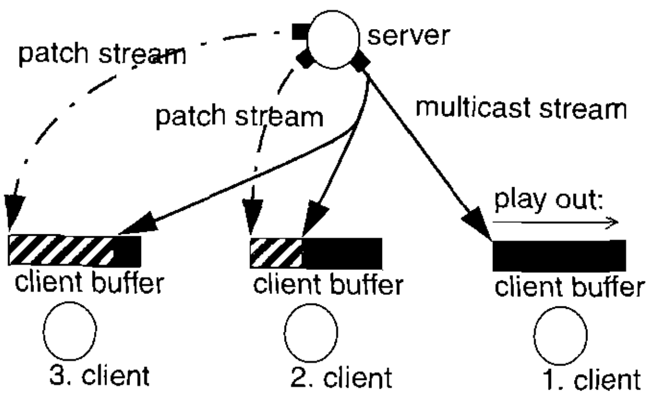

Figure 1: Buffer usage in patching

Patching works by delivering a full movie from start to end to the first client that requests this movie, while subsequent requests in a temporal interval after each multicast movie are not served by transmitting the same movie again. Instead, the client is provided with sufficient information to join the initial stream, and an additional patch stream for the missing initial portion of the movie. These subsequent clients provided with patch streams use local cyclic buffers to delay play-out of received main multicast portion of the movie.

In [13] we present λ*-patching*, an optimization of the *patching* technique by calculating optimal retransmission times for multicast streams based on the measured interarrival time $1/\lambda$, which allows the server to tune the restart times for complete movies on a per-stream basis and thus, to tune the average number of required simultaneous server streams. Based on this number the overall streaming costs for the server can be determined.

Additionally, the *patching* technique can be applied recursively by sending a second patch to the initial patch in addition to the remaining portion of the full movie transmission and so on. The limit to this is given by:

- the number of streams that can be received by a client in parallel,
- the granularity at which a switching from one patch to another makes sense at the client, and
- the smallest sensible interval size for a specific movie, below which a batching [3] or piggybacking [16] approach can be used without recognizable service degradation for the user.

1.3 Reliable Multicast

The design of a reliable multicast protocol is determined by the requirements of a specific application or area of applications that the protocol is built for. Different applications impose different requirements on the underlying reliable multicast protocol. Our most basic requirements are aim at compatibility with the existing infrastructure in the Internet:

- compatibility with standard RTP clients
- based on regular IP-multicast infrastructure without specialized routers
- minimal additional bandwidth consumption to reduce the penalty of slow links

The IETF's reliable multicast working group's draft "draft-ietf-rmt-design-space-00, The Reliable Multicast Design Space for Bulk Data Transfer", which is valid June 1999 through December 1999, provides a checklist of requirements that provide structure to the design considerations of reliable multicast protocols. The following questions are taken from that draft as a checklist, the answers are those that apply to our scenario.

- Does the application need to know that everyone received the data?
  No. Each receiver individually needs to know that it has received all of the data and does not care about its peers. If the receiver stops complaining about missing data, the sender assumes that it has received all data, but it can not be sure; it may also be the case that the receiver has for some reason decided to delete the data that has been received earlier, and does not need the missing data any more.
- Does the application need to constrain differences between receivers?
  Yes. Commercial movie distribution needs e.g. receiver specific watermarking.
- Does the application need to scale to large numbers of receivers?
  The multicast should scale well to moderate numbers of receivers. However, with the application of the reliable element to fill cache servers, the load should primarily be reduced to simple multicast, with the reliability issue limited to the communication among cache servers, or between cache servers and central server.
- Does the application need to be totally reliable?
  No. We do not guarantee perfect delivery of the video stream to the end-user anyway. The reliable transfer between a sender and a cache server is intended to reduce the number of errors that are perceived by the end-user to an acceptable number. The use of multiple stages of caching between the original source of the movie and the end-user, however, could lead to a multitude of errors, due to the ignorance of packet loss of the standard distribution protocol, when the cache servers do not receive a copy of the movie that is close to perfectly intact.
- Does the application need to provide low-delay delivery?
  Yes, but only low-delay delivery of the basic stream. The error recovery does not need to be performed with a low delay.
- Does the application need to provide time-bounded delivery?
  Yes, but only low-delay delivery of the basic stream. The error recovery can take an arbitrary amount of time, although it would be advantageous to repair the missing sections quickly and increase the quality of the movie that is delivered from the receiving cache.
- Does the application need many interacting senders?
  No. In the basic case we assume an architecture with the cache servers acting as proxies for the end-users and lower level caches, and the distribution of movies from the central server to all listeners of its multicast stream. In the extended case, which includes the *patching* technique, some (a few) senders may interoperate to provide service to a lower level cache

server. This lower level cache server requests a stream from its upstream cache, but since that upstream cache has not received the complete basic transmission of the movie itself (i.e. it operates in write-through mode), it offers to the lower level cache only a patch stream, with the baseline stream to be received from the central server. The result is a non time-critical interaction of few senders.

- Is the application data flow intermittent?
  This is not our intention, unless a continuous data flow of UDP packets would be considered intermittent.
- Does the application need to work in the public Internet?
  Yes.
- Does the application need to work without a return path (e.g. satellite)?
  No. We assume an Internet-style network for communication among cache servers and between cache servers and central server.
- Does the application need to provide secure delivery?
  Yes. Commercial movie distribution has to have guaranteed control over the content distribution.

With this checklist, we look at existing reliable multicast protocols. Examples are SRM (Scalable Reliable Multicast) [5], TRM (Transport Protocol for Reliable Multicast) [9], RMTP (Reliable Multicast Transport Protocol) [9] and LRMP (Light-weight Reliable Multicast Protocol as an Extension to RTP) [11]. TRM and LRMP make similar assumptions about loss detection and repair requests as SRM, so SRM can be discussed as an example for all three protocols. RMTP provides sequenced lossless delivery of bulk data (e.g. Multicast FTP), without regard to any real-time delivery restrictions. It uses a windowed flow control and ACKs for the received packets. This technique allows a reliable transmission, but if packets are lost, the data flow is interrupted because the lost packets are resent immediately by the sender which leads to a non-continuous data stream. So this protocol is not applicable for VoD applications.

SRM [2] is a reliable multicast framework for light-weight sessions and application level framing. It's main objective is to create a reliable multicast framework for various applications with similar needs of the underlying protocol. SRM does not distinguish senders from receivers. Whenever data is created, it is multicast to the group. Each member of the group is then responsible for loss detection and repair requests. The repair requests are multicast after waiting a random amount of time, in order to suppress requests from other members sharing that loss. Every member capable of sending a repair packet also sets a timer and if no repair packet is sent from another member it sends the repair packet. SRM's drawback for our scenario is that it needs a specific distribution infrastructure which is not widely available in the Internet at the moment.

A third class of reliable multicast protocols are the ones which include FEC (forward error correction) as a technique to achieve reliability [20]. Reliable multicast achieved through FEC is also applicable for VoD systems, since usually no retransmissions are necessary during the multicast transmission of the video stream. The major drawback of this approach is, that error correction information appropriate for the client with the worst connection must be included in each multicast packet. This will lead to a higher use of bandwidth thus leading to a reduced connection quality for the clients. In addition a completely new protocol must be built in the case of layered FEC since this model is not compatible with already existing protocols.

Our approach is to modify the most commonly used application-level protocol for streamed AV delivery, RTP, to address our specific case of reliable multicast. This variation, called LCRTP (LC for loss collection) is applicable for real-time audio and video data, does not require changes to the infrastructure except for cache servers and is compatible to standard Internet protocols. It uses central error recovery to allow a weighted retransmission (sections of the video that are listed in LC lists from multiple receivers are handled before sections that are reported missing from one receiver only).

LCRTP is used to identify the position of a transmitted packet in a complete movie, recognize packet losses and retransmit lost packets, thus allowing the cache server to store a perfectly correct copy of the movie on its local disks.

LCRTP sends a small amount of data in addition to the RTP header to determine exactly the amount of lost data and its position in the original file. This is achieved by the use of a byte count that represents the actual position in a file. On the receiver´s side in the case of a packet loss the byte count contained in each arriving packet is used to reserve space for the missing data that will be filled afterwards by retransmission.

LCRTP makes use of the standard mechanisms for RTP [22] extensions. The X flag (extension flag) is used in order to show that an extension header is following the standard RTP header. A standard RTP implementation ignores the X flag and the extension header and treats the packet as a standard RTP packet.

The following example shows in more detail how reliable multicast including the above described requirements is achieved with LCRTP. In Figure 2 the original video data is located on server S. Servers CS1 and CS2 are also video servers that operate as proxy and cache servers. Clients CL1 and CL4 are requesting the same video from server S through their proxies CS1 and

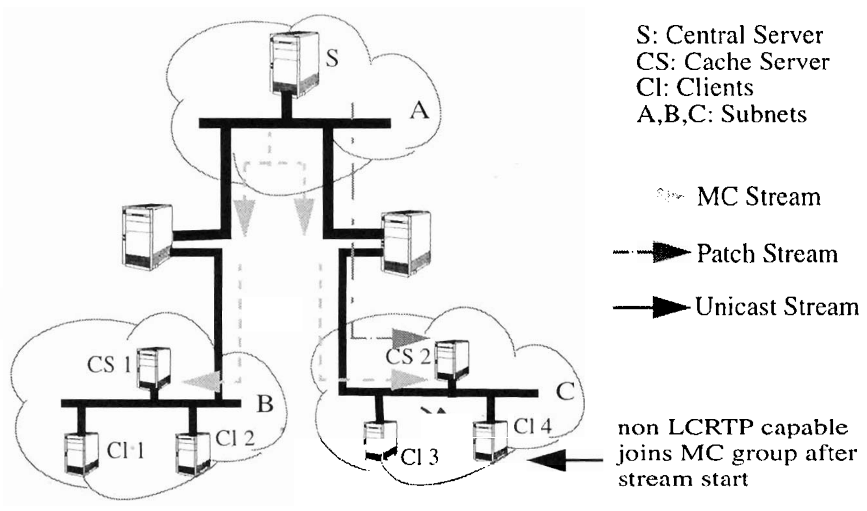

Figure 2: A caching scenario (LCRTP-Patching combination)

CS2, respectively. Assume that the file is neither cached on CS1 nor on CS2 so far. Server S starts sending an LCRTP multicast stream which is received by the servers CS1 and CS2, which perform a write through caching. Thus, the received data is stored locally in a file while as well as being streamed at the same time further towards the requesting clients. At the client side a standard RTP receiver is able to process the incoming LCRTP packets since the extension header is ignored.

If data loss occurs upstream from one of the caching servers, space for the missing information can be allocated at the correct file position on disk due to the byte count that is included in the LC extension header. After the file is transmitted completely, the receivers send an application-specific RTCP packet containing the collection of losses to the sender. This report is sent after a random time to avoid flooding.

The sender then resends the missing data to the receivers. Retransmission also includes the LCRTP functionality to make sure that the whole file is received and stored correctly at the local cache. A timer at the receiver cancels the session if no additional data is received after a while in order to avoid an endless loop of retransmissions.

2 MOTIVATION FOR PATCHING IN CACHING HIERARCHIES

We motivate the integration of *patching* and caching by modeling analytically the necessary effort in an example hierarchical movie distribution scenario. First, we calculate cost functions for various approaches of serving movies to users in hierarchical distribution systems with the topology of binary trees. A binary tree as a model seems very restrictive. However, setting costs on some levels to zero allows for modelling of trees with levels of 2^n nodes, and with more complex operations our model can also be generalized for arbitrary cache trees, using sets of binary trees rather than a single one. If our tree model is used to identify good locations for movie placement, it is highly appropriate to identify the caches of several levels in the binary tree with each other. Starting with these considerations, we apply this analysis to an example system with somehow realistic features.

Figure 3 is a sketch of the base model topology central server *CS*, optional cache servers N_i^t with an index i at depth t in the binary tree, and network links E_i^t. Table 1 lists the symbols that are used in the formulas, and Table 2 presents the formulas for calculating the cost of the distribution systems. In this section, we provide terms, assumptions. We presented the detailled calculations in [14]. The most important limitations of the model are summarized below, but still, this analysis motivates us to realize caching with *patching*. We get a strong hint to combine caching with *patching* in the example below, for a VoD system with rather realistic characteristics, following the assumptions of the analysis.

The effort to set up the system is modeled as an abstract "cost" for basic server installations (including central server and cache servers), cost of server support for concurrent stream deliveries, the cost of concurrent streams support by each network link, and cost for the storage of movies in cache servers. As we assume all movie files to be optimally located in the caching hierarchy, there is no cost for transporting the movies to store and cache and for unnecessary copies. There are several noteworthy aspects to this assumption:

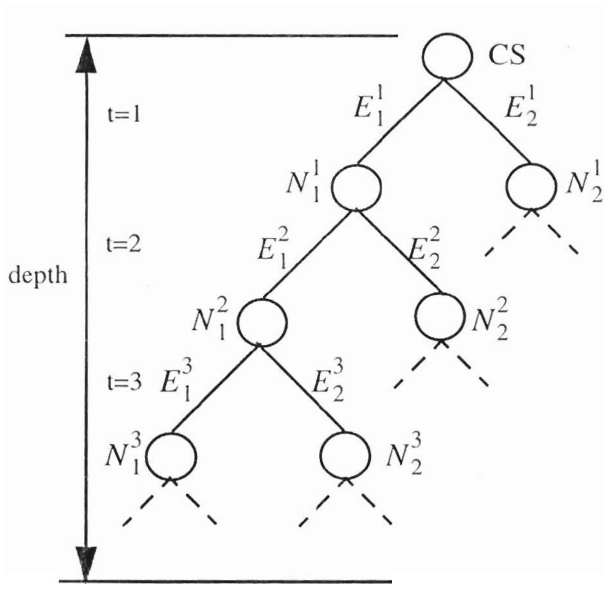

Figure 3: binary tree of analytical distribution system model

- assuming a perfect distribution of movies to cache servers according to their long-term relevance would also render movements due to relocation minimal
- for a downstream movement, caches that work according to our approach do not generate additional network load because they work in write-through mode - upstream movement is certainly missing
- if caching strategies are not sufficiently elaborate (or centrally controlled), they will react to short-term or at least to daytime variations in the request patterns, these calculations will be extremely optimistic

The numerical optimization assumes a distribution of movie hit probabilities according to the Zipf distribution. Although various papers state that the Zipf distribution describes the distribution of hit probabilities at any given time very well, a caching architecture is unable to achieve a distribution according to Zipf.

- The relevance of movies is changing with respect to other movies, which implies that their index value in the Zipf distribution is changing,
- Hit rates do not typically conform perfectly to the Zipf distribution because of user behavior. The divergence is greater for small user populations, which means that distribution systems without an exchange of hit rate information will estimate a movies popularity less exact than a centrally coordinated system.
- Movies must be relocated between cache servers according to their estimated relevance. This may be done predictively (which reduced accurateness of the estimation), so the optimal location for each movie is achieved timely, but such relocations do still incur additional network and server load.
- Homogenous distribution systems are unrealistic.
- Not all movies have equal length and data rate.

Note, that a non-hierarchical approach will probably result in additional savings but for hierarchies, any algorithm should be unable to reach the optimum that can be computed numerically from the formulas in Table 2.

To verify the effects of these computations, we present an example that demonstrates the vast options for savings. This example is simplified from the reality that we envision with the combination of *patching* and caching. For example, we assume that *patching* is implemented in the clients, which is not realistic in a widely distributed network of heterogeneous clients.

Table 1: Elements used in folrmulas

Symbol	**Meaning**	**Symbol**	**Meaning**
S_0	Basic cost of a server/cache server installation.	S_1	Cost for one supported stream of a server.

Table 1: Elements used in folrmulas

Symbol	Meaning	Symbol	Meaning
C_t^E	Cost for one supported stream on a network link at level t.	C_t^N	Cost for the storage needed to store one movie in a cache server.
M	Number of available movies.		Hit probability of movie m.
t(m)	Optimal tree level for caching movie m.	r(m)	Optimal patching window for movie m.

Table 2: Analysis of cost effects of patching on caching hierarchies, cf. [14]

Distribution Method	Calculated Cost Formula
unicast directly from central server	$S_0 + 2^d \cdot S_1 + 2^d \sum_{t=1}^{d} C_t^E$
unicast with caches	$\left[\sum_{t=1}^{d-1}\left(2^t \bigcup_{m \in M} \delta_{t(m)}(t)\right)\right] \cdot S_0 + 2^d \cdot S_1 + \sum_{m \in M}\left[P(m)2^d \sum_{t=t(m)+1}^{d} C_t^E + 2^{t(m)} C_{t(m)}^N\right]$
greedy patch-ing from cen-tral server	$S_0 + \left[2^{d-1} + \sum_{m \in M}(1-(1-\eta_m)^{2^d})\right] \cdot S_1 + \sum_{m \in M}\sum_{t=1}^{d}\left[\left(2^t - 2^t \cdot (1-\eta_m)^{2^{d-t}} + 2^{d-1} \quad \eta_m\right) \cdot C_t^E\right]$,where $\eta_m = P(m)$
patching with limited buffer from central server	$S_0 + \left[\sum_{m \in M}(2^{d-1} \cdot \eta_m + r(m)(1-(1-\eta_m)^{2^d}))\right] \cdot S_1$ $+ \sum_{m \in M}\sum_{t=1}^{d}[\eta_m \cdot 2^{d-1} + (2^t \cdot r(m) \cdot (1-(1-\eta_m)^{2^{d-t}})) \cdot C_t^E]$,where $\eta_m = \frac{1}{r(m)} \cdot P(m)$
patching with caches	$\left[\sum_{t=0}^{d-1}\left(2^t \cdot \delta(\bigcup_{m \in M}(t = t(m)))\right)\right] \cdot S_0 + \sum_{m \in M}(2^{t(m)} C_{t(m)}^N)$ $+\left[\sum_{m \in M}(2^{d-1} \cdot \eta_m + 2^{t(m)} \cdot r(m)(1-(1-\eta_m)^{2^{d-t(m)}}))\right] \cdot S_1$ $+ \sum_{m \in M}\sum_{k=1}^{d-t(m)}[(\eta_m \cdot 2^{d-1} + 2^{t(m)} \cdot 2^k \cdot r(m) \cdot (1-(1-\eta_m)^{2^{d-t(m)-k}})) \cdot C_k^E]$,where $\eta_m = \frac{1}{r(m)} P(m_m)$

In our example, the movie probabilities are distributed according to the Zipf distribution: $P(drawm_m) = z(m) = \frac{C}{m}, C = \sum_{m \in M} \frac{1}{index(m)}$

Besides the predefinitions from the analytical model, we define

- 500 different movies
- 2^{20} active users (i.e. a binary distribution depth of 10, where most nodes do not contain a server)
- a cost of 25000 $ for a basic server installation

- a cost of 100 $ for each concurrent high quality movie stream supported by a server
- a cost of 350 $ for each concurrent high quality movie stream supported on a network link
- a cost of 1000 $ for storage to hold one high quality movie

The location of the caches in the distribution hierarchy for examples 2 and 5 was not optimized. Rather, the caches were moved heuristically upstream until no immediate gain was perceived any more. For the example 2, "unicast with caches", the approach "installed" caches at levels t=12, 10, 8, 6 and 4 in the order to decreasing movie popularity. For the example 5, "patching with caching", the approach "installed" caches at levels t=9, 7, 3, 5 and 1. The heuristic prohibited to choose the level 0 for the least popular movies which would have been roughly three quarters of all movies

Table 3: Example for theoretical effect of the various methods

Modeled Distribution Method	**Calculated System Cost**
1. unicast from central server	7,445 Mio $
2. unicast with caches	4,664 Mio $
3. greedy patching from central server	3,722 Mio $
4. patching with limited buffer from central server	375 Mio $
5. patching with caching	276 Mio $

These numbers indicate, that there are scenarios with a large potential for savings in the joint use of the *patching* and caching techniques. When (costly) caches are introduced in a *patching* distribution system, savings are made with much less expensive necessary system links and storage space (cf. the last two rows in Table 3).

Although this model and these numbers are quite illusionary, and we can not expect clients that to implement *patching* buffers and *patching*-capable protocols, this potential for savings demonstrates that:

- the use of cache servers generates savings that make up for their installation cost
- *patching* with optimized window sizes is the major advancement in savings
- The most important issue for our architecture is:
  The installation of caches in conjunction with *patching* does not eliminate the effect of *patching*. With an appropriately dimensioned cache server, it will even increase the savings by keeping the most popular titles in the cache. Thus, we can proceed to build a wide-area caching architecture that relies on *patching* for wide-area distribution of the videos to cache servers that act of proxies for clients without these specific features.

3 REALIZATION OF PATCHING IN CACHING HIERARCHIES

Our architecture assumes caching proxy servers, clients that receive video stream using RTP streaming and the *patching* technique for information exchange between the central server and the caches.

The choice for the *patching* and file streaming techniques was due to efficiency and bargain availability respectively. To integrate *patching* into the cache server communication, we use LCRTP. This choice fixes the major drawback of *patching* that it can only reasonably be used with specific receivers, while current commercial video distribution systems [21] depend on unicast connections between original server and cache server and also among cache servers in order to guarantee the integrity of the copies in their caches.

Another possibility would be to enhance the cache servers to use *patching* in a way that the stream is reassembled correctly at the cache and transmitted (as a unicast stream) to the client. But using only standard *patching* for the transmission of videos to video caches which belong to one multicast group would lead to a poorer quality of the videos on the cache since losses during the transmission can not be repaired. Therefore we introduce LCRTP as a protocol to repair losses on the caches while maintaining compatibility to standard RTP clients.

In the case that *patching* is used in combination with LCRTP losses can be detected and repaired because a retransmission is possible, which makes sense for caches where the data is stored and not presented immediately. The bytecount that is included in every LCRTP packet allows the cache servers to write incoming data directly to the disk instead of buffering it in their main memory like it would be necessary by a non-LCRTP transmission. The position of each packet in relation to the complete movie is specified by the bytecount and therefore it can be written to its exact position on the hard disk, assuming a file system supporting this behavior.

The combination of *patching* and caching allows clients and video cache servers to join a multicast stream that has already started (see Figure 2). The missing part of the movie is sent in a separate stream to the receivers the way it is done for *patching*. When the patched part of the movie is sent as a unicast or multicast stream should be case dependent, since the amount of receivers can vary. If a client in a subnet decides to request a movie that is already being multicast, the cache server of this subnet (which acts as a VoD proxy) joins the multicast group. Depending on the decision of the implemented caching algorithm the received movie is cached on the video cache or only assembled in right order and then forwarded to the requesting client. This decision should be driven by the popularity of the movie and therefore further possible requests in the subnet [12].

It would also be possible to send multicast streams and additional patch streams directly to the clients, e.g. in case the video cache does not decide to cache the requested movie, but this would require a modification of the clients. If *patching* is only handled by the cache servers which forward the received stream as a unicast stream to the clients, they can exist without any modification. We think that this solution is more reasonable since it is more costly to modify all possible clients instead of the video caches which must be able to handle patch streams in anyway.

To combine the LCRTP and the *patching* approaches in a *patching with caching* system, we need to modify the retransmission approach a little bit. First of all, both the original server and the cache servers of the distribution hierarchy need to be aware of the modification from the original LCRTP usage, or client requests will not be answered in any expected way.

A regular RTP-receiving client contacts the cache server as its proxy server with a request for a certain movie title; products demonstrate that this can be done in the same way for video as it is done for web pages. A cache server that receives this request and has the movie or at least the initial portion of the movie already stored locally assumes that the movie can be streamed to the client in an individual unicast transmission (fine-grained batching could be used to collect a couple of requests in this case as well). Even if the movie is not completely available, some other client is currently requesting it from the server and the remaining part will arrive at the cache server before it is required by the new client. Obviously, this requires that the cache server is able to send data from partially available movies.

If the video is not present at all, the cache server contacts the next upstream server in the distribution hierarchy and requests the title. If the movie title is not in transmission to any other cache server or client at that time, the original server starts a transmission of this movie to the cache server as a multicast stream. The cache server stores this title, and at the same time, forwards the data to the requesting client. A special approach in this case is the possibility to permit the client to listen to the multicast stream, which is also listened to by the cache server. This handling of the special case permits the implementations of conditional overwrite caching strategies that do not store movies just because they are requested for the first time, but do this only if sufficient information is available to deduce that the title will probably be requested soon again. The more generic approach, which does not work with conditional overwrite strategies is to start storing as the data arrives from the original server, and to open a separate unicast connection from the cache server to the client to transmit a stream.

If the video, which is not present in the cache server, is already being sent by the original server to other cache servers or to clients, the original server will decide according to the estimated optimal restart frequency of the requested movie, whether the cache server receives a new complete stream or not.

In the latter case, *patching* is applied and the client receives an individual patch stream in addition to information that is necessary to join the complete stream and the patch streams which are already active for that movie. This process demands to keep and order the individual parts. The position on disk of each RTP packet arriving from the different patches at the cache server is uniquely identified by the byte count in the extended header. The receiving application is responsible for storing the packet at this position in the file. Since all elements are transmitted and received with regular streaming speed, the transmission thread or process, which reads the data from the partial file and forwards it to the client, will experience a lost packet only if the link between the cache server and the original server is congested or the packet is lost for any other reason. The receiving thread or process does not require any special operation beyond the normal LCRTP extensions; however, the concurrent bandwidth needs are much higher at the beginning of a patch transmission than is usual for the regular LCRTP case.

4 CONCLUSION

We have analytically motivated and then presented an approach to the implementation of a wide-area caching architecture. It exploits *patching* to decrease the number of concurrent transmissions for movies in the distribution system. We have shown that the savings in terms of investment in a distribution system can be huge when *patching* is applied. We have also shown that these savings are not lost when *patching* is combined with legacy clients that require linear stream delivery by means of caching proxies. Rather, our examples demonstrates that the savings can increase if the location of such cache servers is chosen appropriately.

We presented the LCRTP protocol and the *patching* technique that we have designed or enhanced for our needs, respectively. Next steps for a full implementation include a selection of a control protocol that can be used or enhanced to receive multiple parts of a patched transmission from different sources at the same time. The analytic model will be enhanced to examine the recursive application of *patching* as demonstrated in [13].

We will continue to enhance multimedia distribution systems, with a specific focus on caching without central control.

REFERENCES

The 4th International Web Caching Workshop, San Diego, California, March 31 - April 2, 1999; URL: wwwcache.ja.net/events/workshop

M. Beck et al., "Linux Kernel Internals", second edition. Addison-Wesley Longman 1998

A. Dan, D. Sitaram, P. Shahabuddin. "Dynamic Batching Policies for an On-Demand Video Server". Multimedia Systems. 1994.

A. Dan, P. Shahbuddin, D. Sitaram, D. Towsley, "Channel Allocation under Batching and VCR Control on Video-on-Demand Systems", IBM Research Report, RC 19588, September 1994

S. Floyd, V. Jacobson, C. Liu, S. McCanne, and L. Zhang. "A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing". IEEE/ACM Transactions on Networking, Volume 5, Number 6, pp. 784-803, December 1997

D. Venkatesh, T.D.C. Little, "Dynamic Service Aggregation for Efficient Use of Resources in Interactive Video Delivery", Proceedings of the 5th NOSSDAV Conference, Nov. 1995, pp. 113-116

R. Krishnan, D. Venkatesh, T.D.C. Little, "A Failure and Overload Tolerance Mechanism for Continuous Media Servers", Proceedings of the ACM MM 97 Conference, 1997, pp. 131-142

J. Nonnenmacher, M. Lacher, M. Jung, E. W. Biersack, G. Carle, "How Bad is Multicast Without Local Recovery?", INFOCOM 1998

B. Sabata, M. J. Brown, B. A. Denny, "Transport Protocol for Reliable Multicast: TRM", IASTED International Conference on Networks, 1996

John C. Lin, Sanjoy Paul, "RMTP: A Reliable Multicast Transport Protocol", INFOCOM 1996

Tie Liao. "Light-weight Reliable Multicast Protocol" Technical Report, INRIA, Le Chesnay Cedex, France, 1998

C. Griwodz, M. Bär, L.C. Wolf. "Long-term Movie Popularity Models in Video-on-Demand Systems or The Life of an on-Demand Movie" ACM Multimedia 1997, November, Seattle, WA, USA, November 1997

C. Griwodz, M. Liepert, M. Zink, R. Steinmetz: "Tune to Lamda Patching" WISP 1999, Atlanta, GA, USA, May 1999

C. Griwodz, M. Zink, M. Liepert, G. On: "Analytical Model for Patching VoD Cache Hierarchies" Technical Report tr-1999-03, KOM TU Darmstadt, Darmstadt, Germany, 1999

C. Griwodz, O. Merkel, J. Dittmann, R. Steinmetz. "Protecting VoD the Easier Way" In Proc. of ACM Multimedia 1998, pages 21-28, September 1998

L. Golubchik, J. C. S. Lui, R. R. Muntz. "Adaptive Piggybacking: A Novel Technique for Data Sharing in Video-on-Demand Storage Servers". Multimedia Systems 4, pp. 140-155, 1996

R. Haskin and F. Schmuck. "The Tiger Shark File System". Proceedings of IEEE 1996 Spring COMPCON, Santa Clara, CA, USA, February 1996

K. A. Hua, Y. Cai, S. Sheu: "Patching: A Multicast Technique for True Video-on-Demand Services", Proc. of ACM Multimedia 1998, pp. 191-200, Bristol, England, September 1998

C. Martin, P.S. Narayan, B. Özden, R. Rastogi and A. Silberschatz. "The Fellini Multimedia Storage Server" appears in Chung Multimedia Information Storage and Management, Kluwer Academic Publishers, 1994

J. Nonnenmacher, E. Biersack, D. Towsley. "Parity-Based Loss Recovery for Reliable Multicast Transmission". ACM SIGCOMM 1997, Cannes, France, September 1997

Real Networks, "Realserver Administration Guide", RealNetworks Inc. 1998

H. Schulzrinne, S. Casner, R. Frederick. "RTP:A Transport Protocol for Real-Time Applications". IETF Audio/Video Transport Working Group, January 1996, RFC1889

23 R. Tewari. "Architecture and Algorithms for Scalable Wide-area Information Systems". Dissertation, Universuty of Texas, Austin, TX, USA, August 1998

24 K. Nichols, S. Blake, F. Baker, D. Black. "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers". RFC 2474, IETF, December 1998