[GrZiLiSt99] *Carsten Griwodz, Michael Zink, Michael Liepert, Ralf Steinmetz:* **Positioṅ Paper: Internet VoD Cache Server Design;** ACM Multimedia 99, Orlando

Position Paper: Internet VoD Cache Server Design

Carsten Griwodz[1], Michael Zink[1], Michael Liepert[1], Ralf Steinmetz[1,2]

[1]KOM - Industrial Process and System Communications
Darmstadt University of Technology
Merckstrasse 25
64283 Darmstadt, Germany
0049-6151-166151

[2]IPSI, German National Research Center for
Information Technology
Dolivostrasse 15
64293 Darmstadt, Germany
0049-6151-869869

{carsten.griwodz,michael.zink,michael.liepert,ralf.steinmetz}@kom.tu-darmstadt.de

1. ABSTRACT

We think that web caches will soon have to better support multimedia demands. In this paper we present a cache server design for internet video on demand (VoD) systems.

1.1 Keywords

Caching, Video on Demand, Internet

2. INTRODUCTION

Internet VoD today is dominated by systems like the Real G2 System [13] supporting various low bandwidth formats. 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 is the necessity to stream the video clip directly from a central server to each client individually, because re-distribution is not established yet. Intranet solutions have existed for a while, they use distributed systems but are typically managed from a central site.

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.

However, we expect that the growth of the Internet 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 that can and should be taken into account [5]
- The movement of full-sized high quality movies among caches is severely more expensive then that of video clips 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. This is unlike the current approach of the web caching commuties to efficiently handle co-operative caches: since the number of requests that reach these caches is large and requested files are mostly very small, these caches are mainly streamlined for simplicity and thus rapid request answers [1].
- Support the delivery of large bulk data files across networks with resource guarantees.
- Evaluate approaches like Hint-based caching [15] for applicability to movie caching as well.
- Support for untrusted caches [7].

We want to specifically handle movie distribution by means of caching, and a feasibility study from the the content providers' point of view. In the remainder of this paper we therefore introduce the combination of our reliable multicast protocol *LC-RTP* for caching hierarchies and our enhanced *Patching* technique [9] for bandwidth friendly True VoD.

3. LC-RTP

Current commercial video distribution systems [13] 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 alternative would be the use of reliable multicast. Approaches like SRM [4] require a specific distribution infrastructure in order to work. Nonnenmacher et al.'s approach [12] does not necessarily need such an infrastructure but without it, all clients receive an amount of error correcting information appropriate for the participant with the worst connection. Of course, this FEC information is also taking up bandwidth.

3.1 LC-RTP Concept

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 with the following requirements:

- 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

LC-RTP sends a small amount of data in addition to the RTP

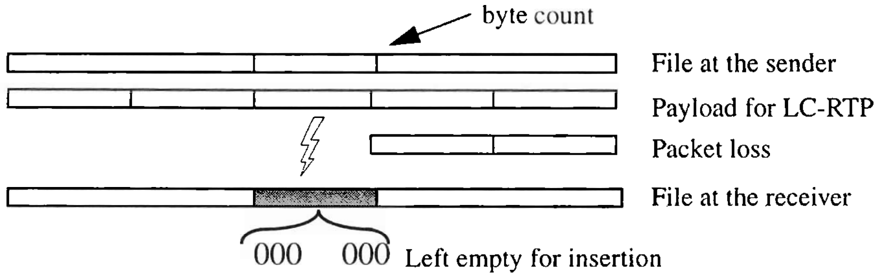

Figure 1. LC-RTP byte count supports retransmission

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 (Figure 1).

LC-RTP makes use of the standard mechanisms for RTP [14] 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.

In case that a sender is asked to send a specific movie via LC-RTP an extension header of 8 bytes is added to each packet. It includes the above described byte count. Assuming UDP packets with a maximum payload of 512 bytes this causes an additional overhead of 1,6%.

3.2 LC-RTP Scenario

The following example shows how reliable multicast including the above described requirements is achieved with LC-RTP. As an example, in Figure 2 the original video data is located on server A. Server B and C are also video servers that operate mainly as cache servers. Clients 1 and 4 are requesting the same video from server A. Assume that the file is neither cached on server B nor on C so far. Server A starts sending an LC-RTP multicast stream which is received by the servers B and C where a write through caching is performed. Thus, the received data is stored locally in a file while as well 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 LC-RTP packets since the extension header is ignored.

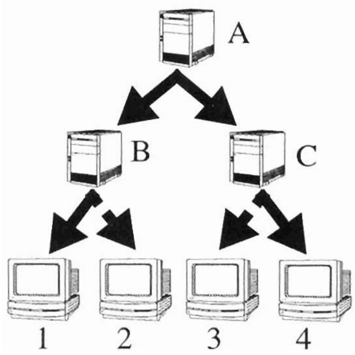

Figure 2. Example scenario

If a data loss occurs at one of the caching servers, space for the missing information can be allocated at the correct position in the file 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. If a report from another client has been observed covering the complete LC list, no report is sent.

The sender then resends the missing data to the receivers. Retransmission also includes the LC-RTP 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.

Without additional techniques, we must admit that we do not observe concurrent requests that allow such savings in real life frequently. To increase the savings we have looked for options to combine LC-RTP with other ideas.

4. PATCHING

We have also found in the past that the Patching approach by Hua et al. [10] has provided an interesting means for saving server capacity in centralized systems. We have presented one approach for dynamical optimization of the server load depending on a specific movie's recent hit probability in [6].

Patching works by delivering a full movie from start to end to the first client that requests this movie.

Subsequent requests in a temporal interval after each multicasted 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. There is an optimal time mostly depending on the frequency of requests for a movie. After that time, a further request is answered by repeating the complete movie multicast stream, instead of sending more patches in parallel (cf. calculations in [6, 10]).

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.

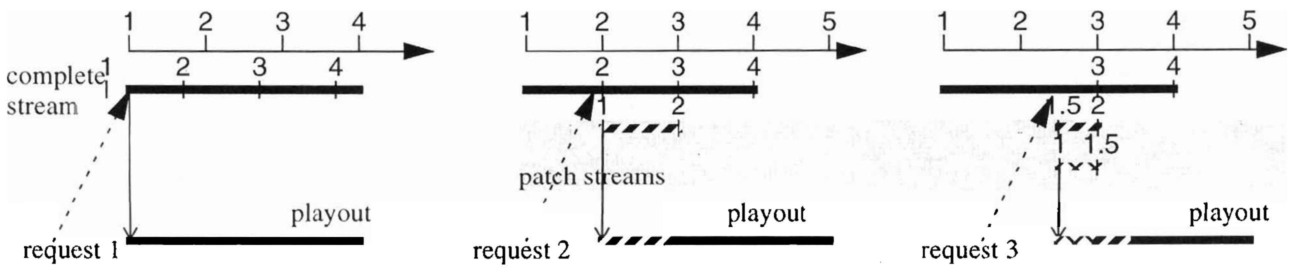

Figure 3. Patching

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 [8] approach can be used without recognizable service degradation for the user.

The use of patch streams is depicted in Figure 3.

In extension to the server load optimization approach of [9], we have made some calculations for long-term savings, but the results have been unstable under different assumptions concerning the network cost in a distribution system and it seems impossible to validate them at this time.

We think that a model in which the VoD clients have a large enough buffer to preload the complete set of existing movies is not feasible even if today it would be possible to equip the clients with a sufficient amount of RAM. Clients should be held as simple as possible in order to manage them easily and also to keep them inexpensive since they may require replacement every few years because of newly available technologies.

Under these assumptions, 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.

5. STREAMING INCOMPLETE FILES

Various multimedia file systems exist that are able to handle the reading from an incomplete file that is still open for writing. Examples of file systems that are capable of this operation with performance guarantees are SGI's XFS, IBM's TigerShark [9] and the Fellini [11] filesystem. It is unclear whether these file systems are able to handle multiple concurrent writes to their files, but other means, e.g. TigerShark's play lists allows the concatenated delivery of subsequent pieces of a movie to a client; storing each patch in a seperate file and delivering the stream from these files one after each other in the correct order circumvents this potential problem.

It would be possible to implement the reception mechanism of LC-RTP in such a way that files are closed after each write operation for one packet. While this is horribly expensive in most file systems, some optimistic implementations such as the Ext2 [2] file system are able to handle several of these streams at once; obviously without performance guarantees.

A second option is inspired by the handling of M-JPEG movies: each packet can be stored as an individual file, and the packet sequence numbers are used to name these seperate files and to guarantee ordered forwarding. This approach, as well, is very resource intensive for the cache server but probably the simplest to implement. If such an approach is taken, it is appropriate to store movies in this way at all times.

Our current approach is to extend Ext2 to fully support admission controlled, concurrent scattered read and write operations.

6. VoD CACHE SERVER DESIGN

To combine the LC-RTP and the *patching* approaches in a *patch caching* system, we need to modify the retransmission approach a little bit. A specific use of the cache server file system is also required, and can be addressed in several ways.

First of all, both the original server and the cache servers of the *patch caching* distribution hierarchy need to be aware of the modification from the original LC-RTP usage, or client requests will not be answered in any expected way.

The client, which is a regular RTP-receiving application, contacts the cache 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. Some more information concerning this issue will be presented later.

If the video is not present at all, the cache server contacts the next upstream server (without loss of generality called the original 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 that 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 the movie on disk as the data arrives from the original server, and to open a seperate unicast connection from the cache server to the client to transmit a stream from that file.

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 LC-RTP extensions; however, the concurrent bandwidth needs are much higher at the beginning of a patch transmission than is usual for the regular LC-RTP case.

As mentioned above, a dedicated cache server file system has to support the store and forward operation for partially received files.

7. INTERMEDIATE RESULTS

We have presented an approach to the implementation of a wide-area caching architecture. It exploits Patching to decrease the number of concurrent transmissions for movies.

We will continue to examine optimizations in multimedia distribution system, with a specific focus on caching.

8. REFERENCES

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

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

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

[4] 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

[5] 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

[6] C. Griwodz, M. Liepert, M. Zink, R. Steinmetz. Tune to Lamda Patching. WISP 1999, Atlanta, GA, USA, May 1999

[7] Carsten Griwodz, Oliver Merkel, Jana Dittmann, and Ralf Steinmetz. Protecting VoD the Easier Way. In Proc. of ACM Multimedia 1998, pages 21-28, September 1998.

[8] 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

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

[10] 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

[11] 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

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

[13] Real Networks, Realserver Administration Guide, RealNetworks Inc. 1998

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

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