Tune to Lambda Patching

Carsten Griwodz[1], Michael Liepert[1], Michael Zink[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.liepert,michael.zink,ralf.steinmetz}@kom.tudarmstadt.de

1. ABSTRACT

A recent paper by Hua, Cai and Sheu [7] describes *Patching* as a technique for reducing server load in a true video-on-demand (TVoD) system. It is a scheme for multicast video transmissions, which outperforms techniques such as Batching in response time and Piggybacking in bandwidth savings for titles of medium popularity, and probably in user satisfaction as well. It achieves TVoD performance by buffering part of the requested video in the receiving end-system.

In a further study, the authors give analytical and simulation details on optimized patching windows under the assumptions of the Grace and Greedy patching techniques. In our view, this does not exploit fully the calculation that was performed in that study. We state that temporal distance between two multicast streams for one movie should not be determined by a client policy or simulation. Rather, it can be calculated by the server on a per video basis, since the server is aware of the average request interarrival time for each video. Since we model the request arrivals as a Poisson process, which is defined by a single variable that is historically called λ, we call this variation "λ Patching". Furthermore, we present an optimization option "Multistream Patching" that reduces the server load further. We accept that some near video-on-demand-like traffic is generated with additional patch streams, and achieve additional gains in server load.

1.1 Keywords

Streaming Server, Video on Demand, Multicast, Adaptive

2. INTRODUCTION

Several approaches have been presented for lowering server load by joining subsequent user requests in VoD systems. [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. [2] modifies this approach towards dynamic batching, which services requests as soon as a stream becomes available. [5] 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. [10] and [8] 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.

For the exploitation of multicast in TVoD systems, Hua et.al. invented *patching*. The basic approach, presented in [7], 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.

We work on wide-area distribution systems without central control and have been looking at various options for caching and prefetching of continuous media data in such a system. While the Patching technique [7] seems to be designed for a central server system, this is not necessarily the only way of using it. Some initial cost calculations are hinting at a joint applicability with a caching architecture. As a prerequisite of those investigations, several tuning options for variations of Patching were considered and documented in this paper.

In the following chapter we provide the calculation of 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. Chapter 4 extends the

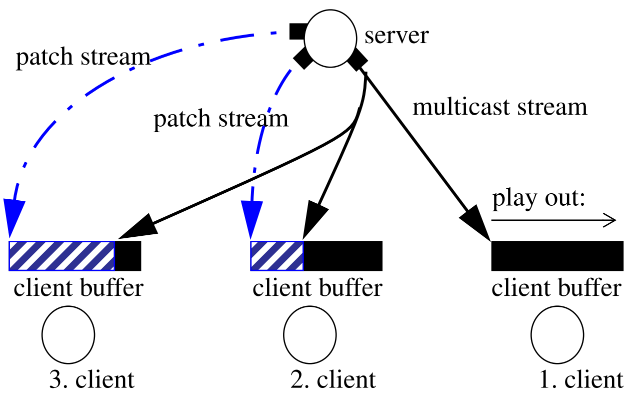

Figure 1. Buffer usage in patching

considerations by adding and optimizing the use of multicast patches, Chapter 5 concludes the paper.

F	length of movie	sec
Δ_M	time interval between multicast starts	sec
$\Delta_U = 1/\lambda$	expected time interval between video demands (unicast starts), according negative exponential distribution	sec
B	buffer length at the client	sec
W	maximum number of streams received by client (receiving load)	number of streams
C_U	cost of unicast stream at server	EUR/sec
C_M	cost of multicast stream at server	EUR/sec
S_U	unicast stream setup cost at server	EUR
S_M	multicast stream setup cost at server	EUR

Table 1: Terms and definitions of the calculations

3. λ PATCHING

Figure 2 demonstrates the starting point of the optimizations: the number of concurrent multicast and unicast streams has a non-trivial minimal value.

For our calculations, we assume Poisson-distributed request arrivals with an interarrival time $1/\lambda$ that depends on the current popularity of the video. We simplify the Patching model by starting multicast streams in cycles of length Δ_M rather than on-demand. This implies a near video-on-demand (NVoD) model for the multicast transmissions. It

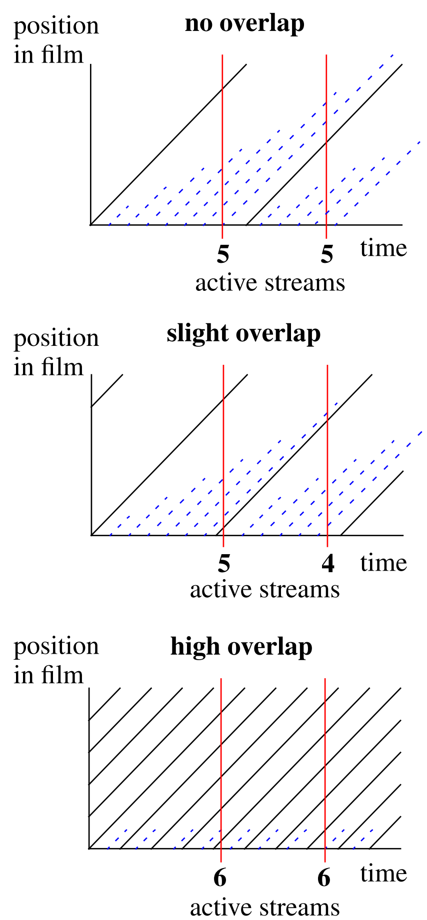

Figure 2. Hints that Δ_M may have an optimum

provides several convenient simplifications to computations, e.g. that the expected value for the number of concurrent streams is time-independent.

We agree with the inventors of the patching technique that the interarrival time varies comparatively quickly during each day. We ignore this issue on the basis that the server's decisions that we propose can be made whenever a request for a video arrives, based on knowledge that has sufficient short-term validity.

3.1 Expected Patch Stream Length

The expected value of the number of unicast streams that are started in each interval of length Δ_M between two multicast stream starts is Δ_M / Δ_U. Assuming that one full multicast stream starts at time 0, the length of each unicast transmission can be calculated as follows:

$$\forall t \in [n\Delta_M, (n+1)\Delta_M): \qquad length(t) = t \bmod \Delta_M$$

If we compute the expected value of the patch stream length, we find that it is $(1/2)\Delta_M$.

3.2 Expected Number of Active Patch Streams

The expected interarrival time of streams is Δ_U. It is clear that the average number of streams that are concurrently active is $\Delta_M/(2\Delta_U)$. The expected value of the number of streams that are concurrently active at a given time t is less intuitive (although the result is the same).

We examine the interval of possible starting times for streams that can still be active at the given time t.

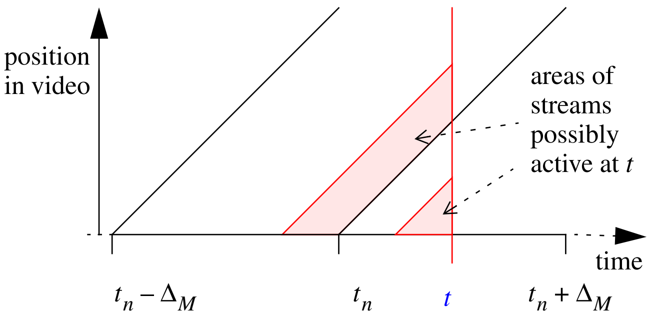

Figure 3. Expected start time intervals for active streams at time t

This interval is defined by two sub-intervals. One includes the streams that are started in the same interval $[t_n, t_n + \Delta_M)$ where t_0 is that latest multicast stream starting time before t and still active at time t. The other includes the streams that have been started in the interval $[t_n - \Delta_M, t_n)$ and that are still active at time t. With earlier definitions, this provides the following set of starting points U_t:

$$U_t = \left[\frac{t + t_n - \Delta_M}{2}, t_n\right) \cup \left[\frac{t + t_n}{2}, t\right)$$

These intervals are always disjoint, and their combined length is $|U_t| = 1/2 \cdot \Delta_M$

Since the Poisson distribution defines that the expected number of arrivals in any interval T is T/Δ_U, this provides the expected number of active streams at time *t,* i.e., the number of streams that are started in U_t, which is $|U_t|/\Delta_U$. This results in equation (1), calculating the expected number of unicast streams active for any *t,*

$$N_u(t) = \frac{\Delta_M/2}{\Delta_U} = \frac{\Delta_M \lambda}{2} \qquad (1)$$

equal to the average number of concurrent unicast streams.

3.3 Optimizing Δ_M

Since all complete multicast streams have length *F,* $N_m(t) = F/\Delta_M$ multicast streams are concurrently active at each time. Together with equation (1), we have the overall number of concurrent streams,

$$N(t) = N_m(t) + N_u(t) = \frac{F}{\Delta_M} + \frac{\Delta_M}{2\Delta_U} \qquad (2)$$

By adding server stream maintenance costs and server stream setup costs for multicast and unicast streams, we get

$$\mathrm{Cost}_{\lambda\text{-patching}} = \frac{S_M}{\Delta_M} + \frac{S_U}{\Delta_U} + C_M \cdot \frac{F}{\Delta_M} + C_U \cdot \frac{\Delta_M}{2\Delta_U} \qquad (3)$$

the overall server streaming cost

We can now use the expected cost by computing an optimal value for Δ_M. It depends on the current popularity of the video, which is expressed by $\Delta_U = 1/\lambda$. We get

$$0 = \frac{\delta}{\delta\Delta_M}(\mathrm{Cost}_{\lambda\text{-patching}}) = -\frac{S_M + C_M F}{\Delta_M^2} + \frac{C_U}{2\Delta_U}$$

(4)

$$\Leftrightarrow \Delta_M = \sqrt{2 \cdot \frac{S_M + C_M F}{C_U} \cdot \Delta_U}$$

By neglecting setup costs and assuming $C_M = C_U$, this can be simplified for an approximation of the optimal value of the client buffer's size as a time B_λ. It depends on popularity and length of a video:

$$B_\lambda = \Delta_M = \sqrt{2 \cdot F \cdot \Delta_U} = \sqrt{\frac{2F}{\lambda}}$$

(under the condition that the client can receive 2 concurrent streams).

We derive Δ_M directly from given figures, so that a video server can recalculate Δ_M for every given film or change in request rate or even bandwidth costs. This approach is more easily applied in the real-world than simulations.

To demonstrate the use of these equations, consider the following example: let multicast and unicast streaming costs be equal, multicast stream setup costs be $C_M \cdot 0.5\mathrm{sec}$ (i.e., setup is worth half a second of streaming) and unicast stream setup costs be $C_U \cdot 5\mathrm{sec}$. Let the film be a popular movie of 4200 seconds with an average request interarrival time Δ_U of 3 sec. This results with equation (4) in an optimal temporal distance Δ_M between multicast restarts of about 159 seconds (equation (5) calculates the same). The server streaming cost for this Δ_M is equivalent to about 53.11 concurrent streams (equation (3)), with multicast streams cost equivalent to 26.3 concurrent unicast streams, including multicast setup costs.

3.4 Given Limits

As every client eventually has to buffer Δ_M of video, the VoD-systems minimum client buffer size is an upper bound to Δ_M.

There is obviously a lower limit to the frequency with which streams need to be started even under very high loads: since there is a limit to the user perception of lag in stream acquisition, it is acceptable to delay the stream start for a few seconds without giving the user the impression of an NVoD system. This imposes a lower limit to Δ_U we did not exploit in our calculations.

4. MULTISTREAM PATCHING

In this section we extend the patching algorithm by additional multicast patch streams. This extension of patching we call Multistream Patching. We demonstrate that the server load can be traded for client network bandwidth.

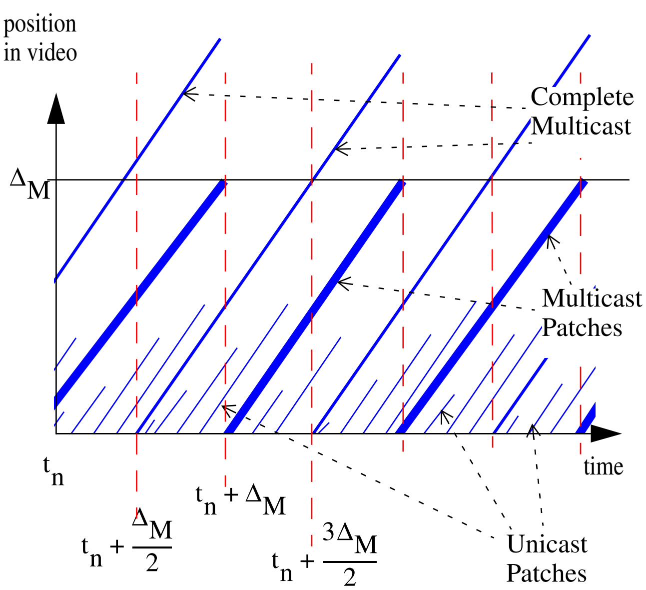

Figure 4. Stream setup example with first multicast patch

4.1 First Multicast Patch Stream

We assume that a client is able to receive up to three streams in parallel. Then, we extend the patching algorithm for the server by the rule: "in every interval $T_n = [t_n, t_n + \Delta_M/2)$ between the starts of two complete multicast streams multicast an additional patch stream at $t_n + \Delta_M/2$, and play it for a length of Δ_M".

The extension requires the client to listen to a complete multicast stream, potentially one unicast patch and potentially one additional multicast patch. This increases peak receiving load on the client up to three concurrent streams, demanding for higher bandwidth between client and server and higher client computation power. The buffer requirements do not change, as the received amount of data to be buffered is still a maximum Δ_M, although eventually written concurrently in two portions).

4.1.1 *Chosen Position of First Multicast Patch*

Unicast patches deliver only the amount of data not available from the last multicast stream (including complete MC streams and MC patch streams). Their average length and with that the average number of concurrent unicast streams is proportional to the gap between multicast streams. We therefore start a multicast patch in the middle of two multicast stream starts to decrease the average required length of unicast patches.

With a multicast patch halfway in between two complete streams, unicast patches only patch a maximum gap of $\Delta_M/2$. In the same way as seen above, this gives us an expected number of $(\Delta_M/2)\lambda/2 = \Delta_M/4\Delta_U$. The average number of concurrent unicast streams over an arbitrary interval with one multicast patch is halved.

4.1.2 *Chosen Length of First Multicast Patch*

There are two cases, depending on the position of the client's request time in the interval between two complete multicast streams.

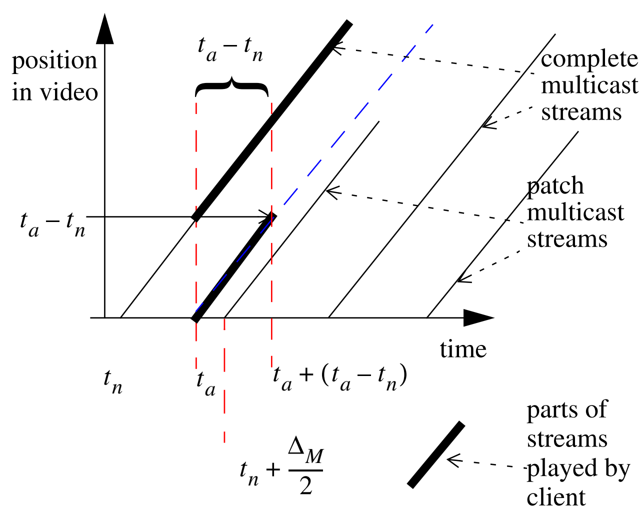

Figure 5. Request at time $t_a \in \left[t_n, t_n + \frac{\Delta_M}{2}\right)$

- If the client requests a video at a time t_a in the first half of an interval between two complete multicast streams (Figure 5), it listens to the unicast patch stream and to the complete multicast stream, immediately playing the unicast. The multicast stream is buffered and played with a delay of $t_a - t_n$.

  These clients do not use the multicast patches the server provides.

- If the client requests a video at time t_b in the second half of an interval between two complete multicast streams

(Figure 6), it listens to the unicast patch stream, to the last multicast patch stream and to the last complete multicast stream. It immediately plays the unicast stream, the two multicast streams are buffered and played with a delay of $t_b - (t_n + \Delta_M / 2)$ for the multicast patch respective $t_b - t_n$ for the complete multicast.

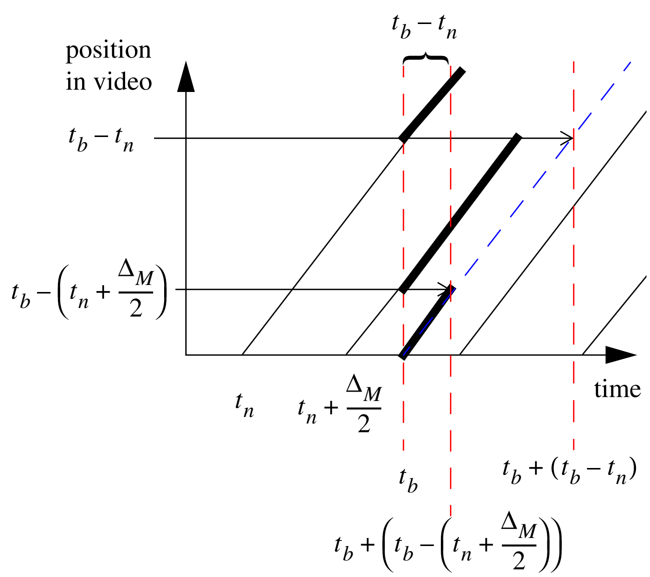

Figure 6. Request at time $t_b \in \left[t_n + \frac{\Delta_M}{2}, t_n + \Delta_M\right)$:

Figure 6 shows that the multicast patch at $t_n + \Delta_M / 2$ eventually has to patch the video data of the interval $[t_b - (t_n + \Delta_M / 2), t_b - t_n)$ with $t_b \in [t_n + \Delta_M / 2, t_n + \Delta_M)$, which gives that the latest video data possibly to be patched are at $t_n + \Delta_M - t_n = \Delta_M$.

Thus, the multicast patch has to cover an interval of data to be patched of $[0, \Delta_M)$, being twice as long as a unicast patch starting at the same time would have to be.

4.1.3 Evaluation of First Multicast Patch

With a fixed client buffer, but with $3/2$ of peak receiving load compared to original patching, we introduced multistream patching with one intermediate multicast patch. With the halved unicast load and with one additional multicast patch of length Δ_M starting every $\Delta_M / 2$, the required bandwidth cost at the server is

$$C_M \cdot \frac{F}{\Delta_M} + C_M \cdot 1 + C_U \cdot \frac{\Delta_M}{4\Delta_U}$$

The gain over non-multistream patching on the server is as below.

$$C_U \cdot \frac{\Delta_M}{4\Delta_U} - C_M$$

This will be a positive value for large Δ_M / Δ_U. In our example, we get 27.4 multicast streams and 13.25 unicast streams concurrently on the server.

Including the stream setup costs for multicast and unicast streams at the server, the cost for multistream patching is:

$$\text{Cost}_{\text{1st mc-patch}} = \frac{2S_M}{\Delta_M} + \frac{S_U}{\Delta_U} + C_M \cdot \frac{F}{\Delta_M} + C_M + C_U \cdot \frac{\Delta_M}{4\Delta_U} \quad (6)$$

With equation (3), this is a gain of:

$$\text{Cost}_{\lambda\text{-patching}} - \text{Cost}_{\text{1st mc-patch}} = -\frac{S_M}{\Delta_M} - C_M + C_U \cdot \frac{\Delta_M}{4\Delta_U} \quad (3)\text{-}(6)$$

This again will be a positive value for large Δ_M / Δ_U.

For or example above, equation (6) gets server costs for λ patching with a first multicast patch as an equivalent to 40.89 concurrent streams, saving in this example an equivalent of more than 12 streams from non-multicast patching.

4.2 n-th Multicast Patch

To introduce the first multicast patch for multistream patching, we had to extend the available maximum client bandwidth to $2 + 1 = 3$ streams, which has to be fully available during a short time immediately after requests. But if clients can receive $W > 3$ concurrent streams, we can introduce $W - 2$ multicast patch streams by applying the multicast patch recursively. The resulting characteristics of multistream patching with n multicast patches are:

- peak receiving load: $W = n + 2$
- a time interval of $\Delta_M / 2$ between multicasts, resulting in an average number of concurrent unicast streams on the server of
$$\frac{\Delta_M}{(2^{n+1}\Delta_U)}$$
- Server bandwidth cost of
$$C_M^S \cdot \frac{F}{\Delta_M} + nC_M + C_U \cdot \frac{\Delta_M}{2^{n+1}\Delta_U}$$
- Server bandwidth and stream setup cost of

$$\text{Cost}_{\text{nth mc-patch}} = \tag{7}$$
$$\frac{(n+1)S_M}{\Delta_M} + \frac{S_U}{\Delta_U} + C_M \cdot \frac{F}{\Delta_M} + nC_M + C_U \cdot \frac{\Delta_M}{2^{n+1}\Delta_U}$$

- With a gain over non-multicast patching of

$$\text{Cost}_{\lambda\text{-patching}} - \text{Cost}_{\text{nth mc-patch}} = \tag{(3)-(7)}$$
$$-\frac{S_M}{\Delta_M} + C_U \cdot \left(1 - \frac{1}{2^n}\right) \cdot \frac{\Delta_M}{\Delta_U} - n \cdot C_M$$

Again, these formulae are valid only for large Δ_M / Δ_U. Also, saved unicast bandwidth soon will be outweighed by additional expenses in multicast path tree setup and bandwidth. But if we consider the equations, we get a theoretical optimum of savings over non-multicast patching

$$0 = \frac{\delta}{\delta n}\left(C_U \cdot \left(1 - \frac{1}{2^n}\right) \cdot \frac{\Delta_M}{\Delta_U} - n \cdot C_M\right) \tag{8}$$
$$\Leftrightarrow n = log_2\left(\frac{C_U \cdot \Delta_M}{\Delta_U(S_M / \Delta_M + C_M)}\right) - 1 .$$

The optimum for n here is computed for a fixed Δ_M, as for now we do not optimize the two-dimensional tupel (Δ_M, n).

The multistream patching scheme could easily be extended to chose n according to a client's buffer and available bandwidth, as existing streaming approaches like MPEG-4 [5] support dynamic setup for multi-stream connections. This would allow for a scheme to individually set up multistream-patching for each client, dynamically calculating the appropriate length of patches.

For our example movie above, equation (8) gives an advice to use the fourth (or fifth) multicast patch:

$$n = log_2\left(\frac{C_U \cdot 159}{3\left(\frac{5C_U}{159} + C_U\right)}\right) - 1 = log_2\frac{159 \cdot 159}{3 \cdot 164} - 1 \approx 4.7$$

This would result in a multicast patch every 9.9 seconds (resp. 5 seconds). Using the fourth (fifth) multicast patch on our example, we get server streaming and stream setup costs equivalent to 32.4 (32.6) concurrent streams, which means further savings of 8.4 concurrent streams over first multicast patching. The video server with n-th multistream patching in this theoretical example could provide TVoD while being only about ten streams more expensive than NVoD at a granularity of 159 seconds (26.4 concurrent multicast streams). As stated above, this is in trade-off to the expense of 159 seconds buffer and the triple(resp. $7/2$) required burst bandwidth on every client.

5. CONCLUSION

In this paper, we have presented two modifications of the patching technique. The first variation *λ–patching* is based on dynamic buffer calculations that can be performed by a video server at request time for each video depending on its length F and popularity, which must be expressed in interarrival times 1/λ. With this information, with respect to server load the optimal temporal distance between complete multicast streams can be approximated as

$$\Delta_M = \sqrt{2 \cdot F / \lambda}$$

The second modification *multistream patching* provides a means of starting streams cyclically, from which end-systems can buffer video data while they receive patch streams for the initial portions of a video. In contrast to the original technique, these cyclically started streams need not be complete video streams, but they can end when sufficient data from a running complete video stream has been received. This approach can be re-iterated. We have provided a formula based on server cost computations that allows to find the optimal number of iteration steps, again depending on a video's current popularity. Some example computations show that this approach can provide remarkable reduction of server load for popular videos in conjunction with the dynamic buffer size selection of the first part.

In future work, we intend to extend cost calculations to the network and to identify an applicable combination of patching with caching techniques.

6. REFERENCES

[1] C. Aggarwal, J.Wolf, P. Yu. On Optimal Batching Policies for Video-on-Demand Servers. IEEE Multimedia Computing and Systems Conference, Hiroshima, Japan, 1999, pp. 253-258

[2] Asit Dan, Perwez Shahabuddin, Dinkar Sitaram, Don Towsley. Channel Allocation under Batching and VCR Control in Video-On-Demand Systems, IBM Research Report, RC 19588, Sept. 1994.

[3] Asit Dan, Dinkar Sitaram, Perwez Shahabuddin. Dynamic Batching Policies for an On-Demand Video Server. Multimedia Systems. 1994.

[4] A. Dan, D. Sitaram, P. Shahabuddin. Scheduling Policies for On-Demand Video Server with Batching. ACM Multimedia Conference, San Francisco, USA, 1994, pp. 15-24

[5] Leana Golubchik, John C. S. Lui, Richard R. Muntz. Adaptive Piggybacking: A Novel Technique for Data Sharing in Video-on-Demand Storage Servers. Multimedia Systems 4, 1996, pp. 140-155

[6] L. Golubchik, J. Lui, R. Muntz. Reducing I/O Demand in Video-On-Demand Storage Servers. ACM Sigmetrics, Ottwawa, Canada, 1995, pp. 25-36

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

[8] Rajesh Krishnan, Dinesh Venkatesh, Thomas D. C. Little. A Failure and Overload Tolerance Mechanism for Continuous Media Servers. Proceedings of the ACM MM 97 Conference, 1997, pp. 131-142

[9] Moving Pictures Expert Group: Text for ISO/IEC FCD 14496-6, ISO/IEC JTC 1/SC 29/WG 11/N2206, 1998

[10] 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