Flexible Media Reflection for Collaborative Streaming Scenarios

Verena Kahmann, Jens Brandt, Lars Wolf
Technische Universität Braunschweig, Institute of Operating Systems and Computer Networks
Mühlenpfordtstr. 23
Braunschweig, Germany
{kahmann,brandt,wolf}@ibr.cs.tu-bs.de

Abstract

In scenarios where recorded presentations are streamed to a group of clients, users prefer to control streaming on demand and to get content in an individually adapted fashion. IP multicast, the obvious solution for group streaming, cannot provide such a level of flexibility. End-system multicast approaches alleviate the need for multicast routing, but require more powerful clients and are more complex regarding to reconfiguration and inter-client synchronization. The dynamic reflector we propose supports the latter issues by grouping clients with the same play-time position into one reflector session. Individual play-time position changes can be performed by opening new so-called reflector sessions or dynamically switching between them. We provide measurements to show that the latency introduced by the use of a dynamic reflector is low compared to other solutions. Furthermore, we provide scalability enhancements in terms of more efficient network resource usage and delay reduction.

1. Introduction

Collaborative streaming scenarios are group streaming scenarios that allow clients to interactively and individually control the streaming session, potentially supervised by a group management. For example, in a distance learning scenario, a group may stream a course presentation and a teacher may form subgroups of clients. Learners who want to advance more quickly may skip chapters on their own and resynchronize to a common group time-line later.

In our work on collaborative streaming [8] we have developed a control architecture based on RTSP and SIP. However, in this work unicast streaming connections are built from each client to the streaming server. Especially for groups that are concentrated in multiple LANs (e. g. learning centers) and reachable by the server via the Internet, this would lead to inefficient backbone resource usage.

Traditionally, backbone resource efficiency for group communication has been provided at network layer by IP multicast routers that form a distribution tree and copy packets. However, multicast routers have not been deployed widely because of scalability and complexity concerns and therefore, IP multicast is not available globally on the Internet backbone. Alternative proposals to overcome the limitations of multicast are overlay networks of end-systems (i.e. clients) that distribute the content. However, these algorithms must reconfigure the overlay tree each time a client leaves the group with a control method, and inter-client media synchronization is more complex to achieve because of a missing common point of synchronization.

Since we assume that specific intermediate systems will be available in our target environments (see section 2.1) we have decided to use a reflector that copies packets at the application layer. Since previous approaches using reflectors [1, 9] have not provided any presentation control functionalities, we have shown in [3] that such functionalities can be added quite easily by a separation of server-side and client-side data path.

In this paper we give measurement results showing that our flexible reflector does not introduce much latency compared to unicast client-server sessions. Although our target scenario is not intended for large-scale distributed groups, a certain number of clients should be handled as efficiently as possible. We will show by measurements that our flexible reflector is able to handle control events of a number of clients in a learning center with a low delay.

The rest of this paper is structured as follows. In the following section we define our target scenario of collaborative media streaming in subsection 2.1, followed by a short review of our flexible reflector approach in subsection 2.2 and concluded by the resulting requirements for the use of our reflector in the target scenario in subsection 2.3. In section 3 we discuss other group streaming solutions as related work. Section 4 gives experimental results concerning the introduced delay of our reflector implementation. In section 5 we provide scalability enhancements in terms of more effi-

cient local network resource usage and delay reduction for control events. A discussion of our proposed flexible reflector approach in the context of group streaming scenarios is given in section 6. Finally we conclude this paper in section 7.

2. Reflectors in Collaborative Streaming

Collaborative streaming is a group streaming architecture that enables users to watch a streamed presentation with others. Depending on group management policies, users may change the play-time position and thus form subgroups or synchronize to other subgroups. Thus, the architecture supports both collaboration and individual interactivity on demand. Group initiation is done by SIP, and a specific management entity maintains policies for group states [8]. Such an application is useful in several scenarios, amongst others distributed learning environments and home networks.

2.1. Scenario

For the work presented in this paper, the target scenario is a distributed learning environment where users participate in a common presentation, e. g. a streamed pre-recorded lecture. Users may execute repositioning requests to jump to different chapters of a course. We distinguish the impact which repositioning requests have on the group: either the whole group follows (so-called *shared repositioning*) or only the user executes this jump and the group keeps the old play-time position (*individual repositioning*). Learners may also dynamically join or leave the presentation, or synchronize to other subgroups streaming the presentation at different points in time. A teacher may control the allowed level of individuality and interactivity by help of a group management entity. In our system policies to work at a certain collaboration and interaction level can be defined, although the system in general will not force any collaboration.

Learners are grouped in several classrooms at possibly distributed network places. The networking environment consists therefore of local area networks (classrooms or learning centers) which may be connected to other learning centers by a wide-area network. Intermediate entities can be easily deployed near clients in such an environment. Individual mobile learners, e. g. in a train, can participate also by the use of wireless Internet access. The latter would receive transcoded content tailored to their needs. Therefore, a transcoding module could be loaded at the reflector and plugged into the client-side data path. Such a module could adapt the stream to the client's requirements such as a reduced temporal or spacial resolution. For signaling purpose of such requirements the optional RTSP method SET_PARAMETER can be used. For all environments we assume that standard Internet protocols like RTSP and RTP are available.

2.2. Flexible Reflector Approach

We have already presented the approach of a flexible reflector in [3] so we give only a short review of the approach here. A reflector has to map a unicast server session to several unicast client sessions. The client sessions that share the same play-time position, and the server session for that play-time position, form a so-called *reflector session*. At RTSP level the reflector acts as a usual RTSP proxy with separated proxy-server and proxy-client parts. We have modified the RTSP proxy state machine for SETUP requests in the case of joining existing sessions, because no session setup to the server is necessary in this case. We have also modified the behavior of the state machine for PAUSE/PLAY requests in the case of individual repositioning if a reflector session does not yet exist. In this case the proxy has to issue a SETUP request to the server before receiving media at that play-time position.

Besides, we separated the data path into a server-side and client-side part. We employed the concept of *stream handlers*, which are entities that implement one specific data processing functionality like copying or transmitting over the net. These stream handlers are composed to paths by a *graph manager* which provides an interface between control and data path. This concept allows us to setup and change data handling of a system in a very easy and flexible way. In our case, the server-side graph manager controls the copying while the client-side graph managers may include transcoding functionality and at last send the data to the clients. We show an overview of this architecture in figure 1.

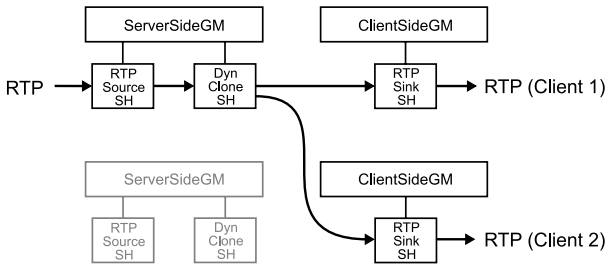

Figure 1. Flexible Reflector Architecture

The reflector session serves as a shared point of synchronization. Data with the same play-time position is sent to clients that are grouped into one reflector session at nearly the same point in time. By using the possibility of joining such a reflector session it is easy to synchronize to a group of clients. For repositioning the whole group only the play-time position of the reflector session has to be modified.

This is done at RTSP level and therefore, nothing has to be changed at the graph manager or stream handlers. Repositioning to an individual position, on the other hand, needs such a change, because the client-side graph manager has to be released from the reflector session and either be added to an existing one or to a newly created one.

2.3. Requirements

Derived from our target scenario of distributed learning environments, we determine the following requirements for a flexible reflector. We suppose that about 30 clients may reside in a local learning center, thus the reflector must support at least 30 parallel media streams and also a number of up to 30 server sessions, since repositioning could lead to one single session setup per client in the worst case.

In our scenario, stored presentations are streamed on demand, thus the actual media delivery delay is not considered as important as in interactive conferencing applications. However, the start-up delay as well as the latency introduced by control events are important. The reaction on such events should follow almost immediately, though users have to tolerate a relatively high latency due to the relatively large play-out buffer delays in streaming clients. If we assume a play-out buffer delay of a second, the reaction on a control event should not take longer than 250 ms.

Since we want to extend the scenario to learners with mobile devices, it must also be possible to include transcoders into the data path. Thus, clients must be able to connect to a reflector independently from other clients' data paths.

We expect inter-client synchronization to be bounded such that a discussion may take place between users in a learning center, because the reflector is located nearby the clients.

3. Related Work

IP multicast is an obvious choice in the case of group streaming scenarios, but as mentioned above IP multicast is not widely deployed on the Internet backbone. The use of IP multicast together with RTSP streaming is only targeted at either live broadcasts or the inclusion of a streaming server into an existing multicast conference. True media-on-demand functionalities as required in our scenarios are not intended for multicast and are therefore not supported by existing streaming server implementations.

Reflectors are no new concept and have been implemented to support streaming for clients without IP multicast access [1, 9]. However, these implementations do not consider presentation control functionality. In the last few years, other approaches for application-level multicast have been elaborated. Approaches like End-system Multicast (Narada) [5] or SRMS [2] form an overlay of end-systems which deliver the traffic to the next end-system in the tree. The latter system uses an interesting probabilistic packet-forwarding algorithm to deal with dynamic membership changes. Scattercast [4] uses specific scattercast proxies (SCX) to copy media to clients, which could be used like a reflector. However, optimization of the overlay tree means possible media data loss, which requires a more complex implementation to receive media on both the old and the optimized link for a certain time. We discuss these application-level multicast approaches in section 6, particularly considering our requirements of a flexible data path reconfiguration.

4. Measurements

Based on our implementation of the proposed flexible reflector we performed several tests to analyze the delay introduced by the use of the reflector. Therefore we measured the time between sending a PLAY request and the arrival of the first RTP packet at the requesting client in different situations. The hardware environment we used for our tests consists of standard components. We ran the server and the proxy each on an IBM x336 server equipped with two Intel Xeon 3.2 GHz processors and 2GB RAM. The client we used for the measurements has a PentiumM 1.7 GHz processor and 512 MB RAM. The operating system of each component was a Debian Linux with a kernel version 2.6.10. Our flexible reflector implementation, the streaming server and the client are commonly based on the komssys RTSP/RTP streaming system originally developed at the Technical University Darmstadt [10]. All measurements were performed in a 100 Mbit/s local area network. Therefore, the network delays between client, reflector and server are negligible, except for the third test series where

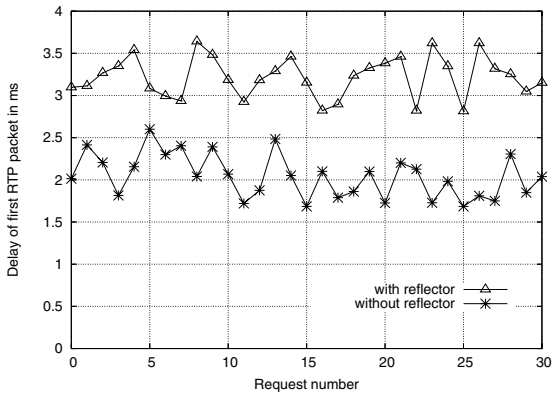

Figure 2. Delay introduced by reflector use

we introduced an artificial server delay by using the netem [6] module for the Linux kernel.

Figure 2 shows the delay introduced by using our proposed reflector compared to the situation of direct client server streaming. Both the server and the reflector were idle and served no other clients at the moment of measurement. Obviously, the maximum delay introduced by the use of our reflector is lower than 2 ms and therefore negligible.

The next series of measurements which is depicted in figure 3 shows the delay of our reflector when a new client wants to join an existing reflector session depending on the number of clients in this session. For ease of measurement we started the joining client on a system like the one mentioned above and ran the clients which are already in the session on the server computer. This shows that apart from the high increase between 0 and 1 connected clients the increase of delay per 10 connected clients is on average about 1 ms. The high increase between zero and the first connected client can be traced back to our prototypic implementation of the reflector which needs further optimization.

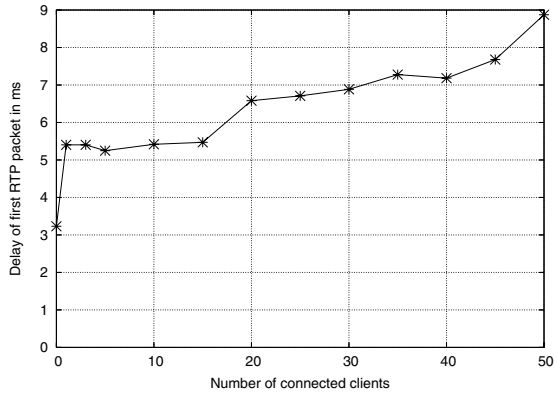

Figure 3. Delay of joining a reflector session

According to our requirements outlined in section 2.3 the reflector should support 30 clients and the delay for a session setup should not be higher than 250 ms. In the situation of 30 clients sharing the same reflector session the delay is about 6.9 ms. Thus the requirements are fulfilled.

In our third test series which is shown in figure 4 we investigated the delay of different types of repositioning requests subject to different server delays, representing servers in a WAN. In the case that a client joins an existing reflector session, i. e. the requested play-time position is the same as that of this existing session, the delay corresponds to the delay between the client and the reflector because no control communication to the server is needed. In the case that a client jumps within a reflector session, i. e. the request changes the temporal position of the session, some control communication to the server is needed and the delay corresponds to the server delay. If the client's request leads to the creation of a new session, i. e. individual repositioning mode is used, the server delay has much more influence on the delay of the first RTP packet.

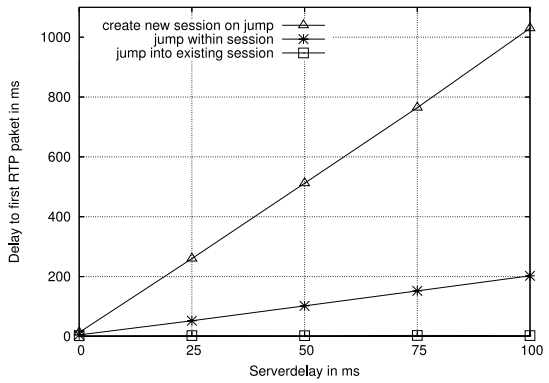

Figure 4. Delay of different control events

Thus, on the one hand our flexible reflector approach leads to lower delays compared to traditional client server streaming if a client joins an existing session. On the other hand it leads to higher delays if a client creates a new reflector session. To alleviate the delay in the latter case we introduced a threshold for finding a matching reflector session in order to prevent the setup of a new session. Further improvements can be achieved by using caching functionality at the reflector which is described in section 5.2.

5. Enhancements for Scalability

We find our measurement results encouraging to use a reflector for collaborative streaming scenarios. Joining an existing reflector session can be done in very short time, and repositioning for a whole group only needs one server interaction, which can be done almost as fast as for a normal client-server session without proxy intervention. Besides, we have seen that a certain number of clients can be supported without introducing higher delays.

However, two problems are still relevant to reflector usage: First, the problem of efficient local network resource usage and second, reducing the delay for new session setup in case of individual repositioning, which is also relevant for the efficient resource usage in wide-area networks. We discuss each problem and solutions to it in the following subsections.

5.1. Efficient Local Network Resource Usage

Since the reflector on a standard server can support more than 50 clients with unicast connections, this would be enough to deliver media to all clients in a classroom. However, if all clients mostly execute shared repositioning, and

individual repositioning occurs quite seldom, maintaining unicast connections leads to high physical link stress in a local area network. Unlike on the global Internet multicast is available in local-area networks, where only a limited number of multicast flows must be handled. We thus propose to use IP multicast alternatively for the downstream link from the reflector to the clients. Because the traffic will be mapped to layer-2 multicast, the physical links will be used very efficiently. Note that we still use unicast connections from the server to the reflector and thus do not rely on global multicast availability. Since multicast usage may be alternatively signaled in RTSP, it is also easily possible to decide on an administrative basis whether the proposed enhancement can be supported or not.

Changing the play-time position means joining a new multicast group, i. e. transport parameters have to be changed within a streaming presentation. The RTSP standard allows such a change only within the SETUP method but not within the PLAY method, which, however, is needed for repositioning. Thus, the clients' implementations need to be modified such that they send a SETUP request before an individual PLAY request. Such a SETUP request can be used for both opening new sessions and synchronizing to a different sub-group. The reflector will then convey the multicast address and the port of the group that should be joined by the client.

For the implementation of our reflector multicast can easily be added. The only differences are that only one ClientSideGM is needed for the whole multicast group and that the RTPSinkSH controlled by this graph manager would send to the corresponding multicast address and port. Further joining clients would just keep their RTSP sessions, while each client has to join the multicast group.

We show an overview of this in figure 5. It can be seen that multicast as well as unicast clients may be supported by our flexible reflector architecture concurrently.

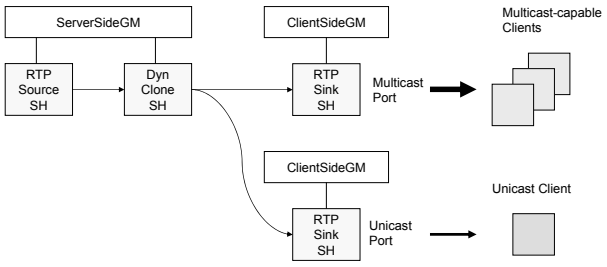

Figure 5. Shared ClientSideGM for Multicast Clients

5.2. Reduction of Individual Repositioning Delay

As already depicted in section 4 one disadvantage of using our reflector is the higher delay in the situation of individual repositioning, i. e. when a client requests to seek to a specific position and therefore must leave its current reflector session. In this situation a new RTSP session between the reflector and the server needs to be set up which introduces this delay. This setup of a new RTSP session whenever one client leaves its current session can also lead to many sessions with very close but distinct viewing positions in time. To alleviate the creation of new RTSP sessions we have introduced a threshold for very slight differences in time of two reflector sessions. When searching a matching reflector session we also consider those sessions with a temporal deviation to the client's request lower than the given threshold. For differences greater than the threshold, such as 500 ms, we can use the Sliding Interval Caching method [11] for our reflector. If a client requests to jump to a temporal position before the position of an existing reflector session (i. e. jumping back in time) this client is served from the cache and no further RTSP session needs to be set up between the reflector and the server. If a client wants to seek to a future position in time the reflector sets up a new RTSP session to the server but can now cache the stream for clients being at an earlier temporal position. Thus, the use of such a caching functionality leads to the avoidance of any control communication to the server and therefore reduces the delay to the same value as in the case of a jump into an existing session (see fig. 4).

An additional reduction of the number of reflector sessions can be achieved if special clients are used which are capable of patching [7]. In this situation they can join an existing reflector session with a later temporal position as requested and request the missing portion of the stream from the reflector separately.

6. Discussion

In terms of scalability end-system multicast is an interesting approach, because no specific host that has to deal with a large number of sessions is required in the network. Intelligent overlay tree building algorithms manage to keep the degree of hosts small. This also saves bandwidth. Besides, reconfiguration allows hosts to leave the group, which is also important in the case of a host failure.

However, end-system multicast approaches are not built for frequent reconfigurations which may be the case if repositioning in a presentation is allowed like in our scenarios. Besides, approaches that rely on end-system data forwarding need considerable complexity at end-systems for measurements, monitoring and tree building. Since in our scenarios gateways are likely to be used, such gateways can

deal with these tasks.

A further advantage of a reflector is a common synchronization point for the group. At the reflector all packets are sent at the same time except for queuing delays. In end-system multicast some receivers will experience a higher delay than others related to the server. Thus, artificial delays must be introduced for inter-client synchronization, which would have to be adapted for each tree reconfiguration.

Particularly for mobile participants the feature that each client is only dependent on its own network conditions is important. A change in network conditions does not mean any reconfiguration, instead an optional transcoding module can react on it. Moreover, clients may change their play-time position individually without affecting data delivery to other clients.

From this comparison of reflectors and end-system multicast related to collaborative streaming scenarios, we find it more important to solve the problem of individual repositioning by adding cache strategies to our reflector. However, we are aware that one reflector for a whole wide-area learning scenario will not be enough in future. Thus, we intend to establish a hierarchy of reflectors with a repositioning-aware service composition approach for discovery and tree-building. This composition algorithm may benefit from the overlay-tree building algorithms like Scattercast [4] that were developed for end-system multicast.

7. Conclusion and Further Work

As we have shown reflectors are an alternative approach to multicast streaming in group streaming scenarios that can be deployed with little effort. We have presented an enhancement of usual reflectors which allows changes of the play-time position. Participants of a collaborative streaming scenario can be grouped in a reflector session according to their play-time position. Members of one reflector session thus share a common server-side data path. A change of the play-time position is thus the same as joining and leaving a reflector session.

We have shown by our measurements that repositioning requests can be served without much delay if shared repositioning is used or if an already existing reflector session can be joined. The fact that a setup of a new reflector session introduces higher delay leads us to the inclusion of caching strategies. We have also proposed enhancements for scalability to a larger number of clients in a local network. This needs a slight change of protocol processing in the clients, because transport parameters must be changed with a SETUP request, but has the advantage that less networking resources are used.

We also compared our approach to existing end-system multicast approaches, which are made to scale for larger groups, but do not consider frequent data path reconfigurations. However, overlay tree building algorithms of these approaches can also be applied to build a hierarchy of reflectors to be used in larger collaborative scenarios. This will be an issue for further investigation.

For the inclusion of caching many strategies already exist. We plan to examine some more of them and derive cost functions to optimize the use of our reflector with these strategies.

8. Acknowledgement

This work has been partly supported by the European Union under the E-Next Project FP6-506869.

References

[1] Apple developer connection: Quicktime streaming. Published at `http://developer.apple.com/documentation/QuickTime/PDF/QTStreaming.pdf`, 1998.

[2] S. Banerjee, S. Lee, R. Braud, B. Bhattacharjee, and A. Srinivasan. Scalable Resilient Media Streaming. In *14th International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, pages 4–9, Cork, Ireland, June 2004. ACM.

[3] J. Brandt, V. Kahmann, and L. Wolf. A Flexible Reflector for Media Streams. In *Kommunikation in Verteilten Systemen, Kurzbeiträge und Workshop der 14. GI/ITG-Fachtagung*, volume PI-61 of *Lecture Notes in Informatics*, pages 41–48. Gesellschaft für Informatik, Bonn, March 2005.

[4] Y. Chawathe. Scattercast: an adaptable broadcast distribution framework. *Multimedia Systems*, 9(1):104–118, July 2003.

[5] Y.-H. Chu, S. G. Rao, S. Seshan, and H. Zhang. A Case for End-System Multicast. *IEEE Journal on Selected Areas in Communication*, 20(8):1456–1471, October 2002.

[6] S. Hemminger. netem Network Emulator. Published at `http://developer.osdl.org/shemminger/netem/`.

[7] K. A. Hua, Y. Cai, and S. Sheu. Patching : A multicast technique for true video-on-demand services. In *ACM Multimedia*, pages 191–200, 1998.

[8] V. Kahmann and L. Wolf. A Proxy Architecture for Collaborative Media Streaming. *ACM/Springer Multimedia Systems Journal*, 8(5):397–405, 2002.

[9] P. T. Kirstein and R. Bennett. Multimedia Education and Conferencing Collaboration over ATM Networks and Others (MECCANO). Final Report, RE 4007, June 2000.

[10] KOM(S) Streaming System. Published at `http://komssys.sourceforge.net/`.

[11] R. Tewari, H. M. Vin, A. Dan, and D. Sitaram. Resource-based caching for Web servers. In *Proceedings of ACM/SPIE Multimedia Computing and Networking (MMCN'98)*, 1998.