A Flexible Reflector for Media Streams

Jens Brandt, Verena Kahmann, Lars Wolf
Institut für Betriebssysteme und Rechnerverbund
Technische Universität Braunschweig
{brandt, kahmann, wolf}@ibr.cs.tu-bs.de

Abstract: Reflectors mapping multicast to unicast sessions have been proposed to enable multicast media distribution for clients in non-multicast capable networks. However, existing implementations do not allow clients to modify the session by using VCR-like commands in a flexible way. In this paper we describe the design and implementation of a reflector mapping a multicast or unicast session to several downstream unicast sessions. We show that by applying a separation of control and data path such a flexible reflector can handle true media-on-demand functionality for several scenarios, even for group communication, more efficiently than several unicast sessions.

1 Introduction

Distribution of multimedia data over the Internet requires a high level of server and network resources, which may be efficiently reduced using IP multicast. Despite the research effort of many years, multicast routing has not found widespread acceptance on the Internet. As one of the proposals for alternative group communication services so-called reflectors have been designed for media streaming applications to support clients in non-multicast capable subnets. Those reflectors act as a proxy between server and client, mapping multicast to unicast sessions. One example of such a reflector is the broadcast reflector in Apple's QuickTime streaming system [App98].

However, existing reflectors neither support different quality requirements and capabilities of clients nor true media-on-demand facilities like pausing or position changes in the stream. Thus, we propose the implementation of a flexible and modular reflector, which can map a multicast or unicast session on the server side to several unicast sessions on the client side. By using this tool, each client of a group can use a dedicated control session allowing flexible data path reconfiguration according to its demands. The server-side data path on the reflector will normally be shared, however, due to control requests of a client a different server-side data path may be chosen or established. This separation of control and data path on the one hand as well as the separation of server-side and client-side data path on the other hand, leads to the ability of our reflector to manage the cases of a true media-on-demand system. To support different quality requirements and capabilities of clients, transcoding modules can easily be used at the reflector by adding them to the client-side data path.

Thus, our reflector provides – in contrast to multicast – individual flexibility while it is more efficient than several unicast sessions from the server to each client. The rest of the paper is structured as follows: After discussing related work in section 2 we introduce scenarios for the use of a flexible reflector and elaborate on use cases for handling client's requests in section 3. The design of our reflector is presented in section 4, where we provide details on the data processing architecture, the internal data path management for joining clients and repositioning requests as well as the necessary RTSP signaling. We present a short qualitative evaluation of our approach in section 5 before we conclude the paper in section 6.

2 Related Work

Multicast to unicast reflectors are no new concept in media streaming. For example, reflector tools for MBone sessions have been proposed, such as the MURS reflector, a part of the MECCANO project [KB00]. A commercially successful implementation of a reflector is Apple's QuickTime proxy which acts as a reflector for clients which are not capable of participating in multicast sessions. Both projects, however, are not intended to handle VCR-like control requests of clients to enable true media-on-demand sessions.

Besides reflectors, there are several other proposals to implement alternative services for group communication [ESRM03]. Overlay networks of unicast connections are currently discussed for end-system multicast. However, these approaches make greater demands on the clients, since clients should relay data to others. The REUNITE approach [SNZ01] uses a concept of reflection similar to ours, but packets are copied on network layer by REUNITE-capable routers. Though this is an interesting concept, it is not useful in our target scenarios, since it is important to control the data path to be able to use transcoding modules and the REUNITE approach cannot provide this.

3 Reflector Use in Media Streaming

The potential advantages of a flexible reflector can be seen by considering several scenarios. Additionally, the flexibility of our reflector, distinguishing it from existing approaches, will be shown presenting different use cases of handling VCR-like repositioning requests of a client. At this point we introduce the notion of a *reflector session*, which can be seen as a group of streaming sessions which share the same server-side data path. Two clients which are participating in one reflector session will get the same media content at the same time (possibly tailored to their needs). In other words all clients of one reflector session share the same time-line. Thus the participation of a client in a reflector session will change if its position in the stream changes, i.e. by repositioning requests.

3.1 Scenarios

Reflectors are useful in many scenarios where the transport of multimedia data cannot be done by using multicast communication (or only on parts of the network path), but should still be handled as efficiently as possible. For example, several clients may wish to view the same content but have different quality requirements. Layered coding using multicast would be a solution in this situation, but those techniques are not very widespread and affect existing media servers. Moreover, layered video coding supports only a limited set of different quality steps and therefore cannot provide custom-tailored streams. Thus, the use of transcoding techniques on intermediate systems can mediate the gap between the server and the clients. In our case, a server may deliver streams using multicast, while the reflector maps those sessions to unicast sessions and hands the data to a transcoding module where appropriate.

Another scenario where a flexible reflector can be useful is collaborative media streaming. Users participating in a shared media session may wish to take individual control of the stream time-line. We have already presented a flexible proxy architecture for the management of signaling between different clients in [KW02]. To deliver multimedia contents efficiently, multicast transport may be chosen at the server side. Thus, a reflector will be useful to combine individual signaling with efficient media data transport.

3.2 Use Cases

For the design of our flexible reflector we have figured out several use cases. These can be distinguished mainly by the handling of VCR requests such as changing the position or pausing a stream. Both requests affect the position in the time-line of the active session and therefore, they are called *repositioning* requests in the rest of this paper.

Three different situations of position changes need to be distinguished:

1. Repositioning is neither possible nor allowed.
2. Repositioning requests of one client affect all other clients of the reflector session.
3. Individual position changes are allowed.

An example for the first situation is the use of normal multicast to unicast reflection as done by already existing tools such as Apple's QuickTime proxy [App98]. Streaming a unicast live presentation to a number of unicast clients is another example where position changes are not possible. In both cases, our flexible reflector will act as a normal reflector: Joining clients just have to be connected to the initial reflector session, which has been established by the first requesting client. A similar simple handling of VCR requests is appropriate in the second situation where any change of a reflector session affects all participating clients. An example for such a situation is a learning environment or a reflector session comprising only one client.

Things get different when individual repositioning or pausing is allowed, e. g. in a situation where several clients share the same unicast path between reflector and server. In this situation the requesting client leaves the active reflector session and needs to join another existing session matching its request, a so called *matching session*. Depending on the number of clients participating in the current session and the existence of matching sessions we can distinguish four more cases:

3.a The current session contains only one client and there is no matching session.

3.b The current session contains only one client and there is a matching session.

3.c The current session contains more than one client and there is no matching session.

3.d The current session contains more than one client and there is a matching session.

The first of these cases can be handled similarly to case 2 because the current session of the client can be changed according to the client's request. In the second case the current session can be deallocated and the client can join the existing reflector session which matches its request. In the last two cases the client can leave the current reflector session and a new reflector session needs to be created or the client joins a matching session, respectively. But due to different granularities of the clients' time-lines it is very unlikely to find an exact matching session. Therefore, we define a session S_i with its current time t_i to 'match a request' of a client with time t_R if and only if its temporal deviation $\Delta t = |t_i - t_R|$ is less than a threshold T. Now the best matching session can be determined by calculating the minimal temporal deviation. Depending on the presentation's nature this method can also be adjusted, e.g. by considering only those sessions S_j with a corresponding time t_j less than the time of the client's request t_R.

4 Flexible Reflector Architecture

To build our reflector we used a data processing architecture based on *Stream Handlers* (SHs) [GZ01]. SHs are small data processing units which can be connected by a controlling unit into an acyclic directed data path. A SH can either consume or generate data units or transmit data units from its input to its output. Additionally, each SH can manipulate the data units and, when connected to a data path, a stream which is pipelined through these SHs can be processed or manipulated in a certain manner. The controlling unit which connects several SHs to form a data path is called *Graph Manager* (GM). The GM is responsible for the configuration of each SH when the data path is built as well as for its control and reconfiguration during transmission of the stream (Figure 1).

The minimal useful data path for media streaming with RTP in this architecture consists of two SHs: *RTPSourceSH* and *RTPSinkSH*. The former SH receives RTP packets from a streaming server and the latter one sends them to a client. The GM plus this data path forms a streaming session which can serve exactly one client, which would be an appropriate architecture for a proxy. To manipulate the streaming data before it is sent to the client, other intermediate SHs could be plugged into the data path. A caching proxy, for example, could use a SH which can copy and store the stream into a file.

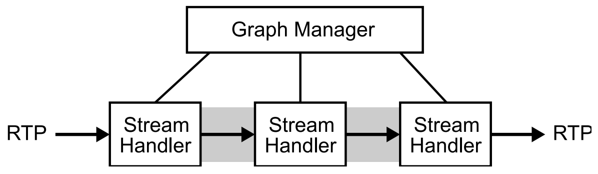

Figure 1: Several SHs controlled by one GM

4.1 Data Path Management

When more than one client should receive the same stream from the same proxy, the data packets need to be copied for each client. However, the number of clients which should receive the same stream is not known in advance and could change dynamically during media transmission. Therefore, we designed the *DynCloneSH* which initially, when only one client uses the data path, just forwards the incoming data packets to its output. When supplemental clients should be served by this data path, the number of output ports of the *DynCloneSH* can be extended dynamically. For each new client a *RTPSinkSH* can be connected to the newly created ports and hereby extends the existing data path. Inside the *DynCloneSH* the data packets will not be copied, but the reference to the corresponding data buffer will be duplicated by the number of output ports.

In addition to this special SH, a data path which can serve more than one client is also needed. Therefore, we split the data path into two parts: the server-side part, managed by a GM called *ServerSideGM* and the client-side part, managed by a GM called *ClientSideGM*. The former GM holds an *RTPSourceSH* and a *DynCloneSH* whereas the latter one only holds an *RTPSinkSH*. Both GMs are created at the time when a new client requests to setup a media session. This split up design makes it possible to serve more than one client by one data path and to substitute the server-side data path if necessary (i.e. due to repositioning requests).

If a new client wants to join an existing media session the newly created *RTPSinkSH* of its *ClientSideGM* is connected to the *DynCloneSH* of the existing *ServerSideGM* as illustrated in figure 2. The newly created *ServerSideGM* is not used in this case but is needed for potential position changes of a client. Otherwise, when a connecting client does not want to join an existing session, a new reflector session is built up by its *ServerSideGM*.

In cases where no position changing of any client is allowed or each repositioning should apply to all clients the creation of the unneeded *ServerSideGM* could be avoided, but for simplicity we have not implemented this feature yet. In all other cases this GM is needed for potential position changes in the future.

If the joining client has different quality requirements, an intermediate transcoding SH could be used between *DynCloneSH* and *RTPSinkSH* which could tailor the stream according to the requirements of the client. For signaling the requirements the optional RTSP-methods `GET_PARAMETER` and `SET_PARAMETER` can be used.

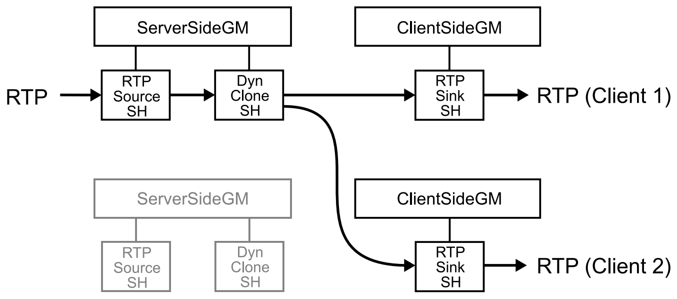

Figure 2: A second client has joined an existing session.

4.2 Handling of Position Changes

We have already distinguished four cases for repositioning in 3.2. Let us first consider case 3.b where the request of a client being in a reflector session with other clients cannot be fulfilled by a matching reflector session already established. The client has to be disconnected from its current session and a new reflector session needs to be set up. This will be done by referencing a free *ServerSideGM* on a repositioning request. The reference to the old *ServerSideGM* must be deleted as well as the link to the old *DynCloneSH*. Instead, a link from the *DynCloneSH* in the new *ServerSideGM* to the *RTPSinkSH* in the repositioning *ClientSideGM* is created.

We show the procedure of repositioning a client exemplary in figure 3. Here, client 2 asks for repositioning. The proxy reflector will break up the link to the current *ServerSideGM* and call the function *getFreeServerSideGM()* giving back a reference to an available *ServerSideGM* where the *DynCloneSH* can be connected to the *RTPSourceSH* of the client (dashed in fig. 3).

Considering the case there is a matching reflector session for a client's request, the streamer interface just needs to call *getServerSideGM(clientID)* which can deliver the reference to a *ServerSideGM* a certain client depends on. The client's old *ServerSideGM* may be marked as free if no other client uses this Graph Manager (case 3.b in 3.2), otherwise (case 3.d) only the link to the *ClientSideGM* is dereferenced.

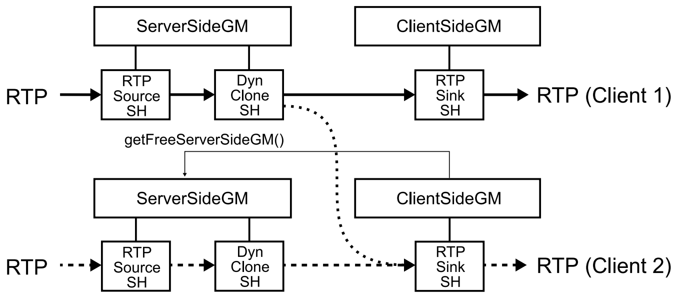

Figure 3: Repositioning of a Client

4.3 RTSP Signaling

As already mentioned we have separated the control and data path in our flexible reflector design. RTSP signaling requests are used on the control path to instantiate Graph Managers and invoke their functionalities as shown before.

In order to know when a reflector session should be established, the proxy must examine the session description delivered from the origin server. If this session description offers a multicast address in the connection field, but the client requests unicast transport in RTSP SETUP, the proxy will setup a reflector session with the according ReflectorGMs as described above. If the client is able to stream multicast, nothing needs to be done by the proxy and it could just forward the SETUP request containing multicast transport parameter.

In the unicast case, a reflector session is always established for a client streaming a new presentation from a server, thus, the server-side and client-side Graph Managers are instantiated. This is done to handle the case of (1) live sessions that are unicast to the proxy but should be delivered to multiple clients and (2) collaborative streaming sessions where other clients may join the session later.

If other clients join an existing reflector session, again, the session description must be examined. The information which reflector session is to be joined can be derived from the offered multicast address in the description. In case of collaborative streaming, a collaborative service component may be queried to return information on the client already streaming at that certain position.

5 Comparison with Multicast

Using alternatives to IP multicast routing always comes with a decrease in performance, since the distribution tree may not always be optimal. If the reflector proxy is not placed optimally, several parallel data sessions may be sent over the same link. Besides that, the reflector may be a hot spot in the network, if it has to handle many clients and sessions.

However, the possible performance decrease comes with a qualitative enhancement. By using the flexible reflector, it is possible to enable scenarios like transcoding and collaborative streaming. True video-on-demand architectures like patching (which need dynamic data-path reconfiguration) can also profit in the sense that clients which are not capable of multicast may participate Since we can share the path from server to proxy reflector, data transport is still more efficient than in the case of several unicast sessions.

In order to overcome the performance and scalability issues, a two-level hierarchy of reflectors could be used, where the upper-level reflector is placed in the backbone near a natural branching point, whereas local reflectors control a set of clients in the same domain. The second reflector class could even be implemented as unicast to multicast reflectors if the domain is capable of routing IP multicast. A different way to go would be the creation of an overlay network of reflectors by exploiting well-known overlay topology building algorithms. Automatic placement algorithms are thus an issue for future work.

6 Conclusion and Future Work

In this paper we have presented the design of a flexible reflector following the stream handler concept. A reflector used for mapping multicast to unicast streaming sessions has proven to be useful for clients not being able to receive multicast data. Another advantage is the possibility for the proxy to control the data path, thus, transcoding for resource-limited clients can be done.

Our reflector can also be used for collaborative streaming scenarios, since unicast to unicast mapping can also be handled. Signalling of such sessions may have to be enhanced in the way that a tag must be delivered indicating a joining client demands to see the presentation at the same position. The reflector we designed is flexible in the sense that each client may request position changes corresponding to the establishment of a new data path in the reflector. For providing the data path the reflector may have to set up a new session to the origin server.

The flexible reflector thus provides for individual client sessions similar to unicast with a more efficient data transport from server to proxy. Future work includes testing the flexible reflector with transcoding modules. For handling many collaborative streaming scenarios, the flexible reflector may be enhanced such that easy switching of data paths in a collaborative group will be possible. Possible placement strategies and hierarchies of reflectors will be further issues of research.

References

[App98] Apple Developer Connection: QuickTime Streaming. Published at http://developer.apple.com/documentation/QuickTime/PDF/QTStreaming.pdf, 1998.

[ESRM03] Ayman El-Syed, Vincent Roca, and Laurent Mathy. A Survey of Proposals for an Alternative Group Communication Service. *IEEE Network*, 17(1):46–51, January 2003.

[GZ01] Carsten Griwodz and Michael Zink. Dynamic Data Path Reconfiguration. In *International Workshop on Multimedia Middleware 2001 at ACM Multimedia*, pages 72–75, October 2001.

[KB00] Peter T. Kirstein and Roy Bennett. Multimedia Education and Conferencing Collaboration over ATM Networks and Others (MECCANO). Final Report, RE 4007, June 2000.

[KW02] Verena Kahmann and Lars Wolf. A Proxy Architecture for Collaborative Media Streaming. *ACM/Springer Multimedia Systems Journal*, 8(5):397–405, 2002.

[SNZ01] Ion Stoica, T.S. Eugene Ng, and Hui Zhang. REUNITE: A Recursive Unicast Approach to Multicast. In *INFOCOM 2001 (3)*, pages 1644–1653, Tel Aviv, Israel, March 2001.