Scalable TCP-friendly Video Distribution for Heterogeneous Clients

Michael Zink[1], Carsten Griwodz[2], Jens Schmitt[1], and Ralf Steinmetz[1]
[1] Multimedia Communications, Darmstadt University of Technology, Germany
[2]Department of Informatics, University of Oslo, Norway
Email: {Michael.Zink,Jens.Schmitt,Ralf.Steinmetz}@KOM.tu-darmstadt.de, griff@ifi.uio.no

Abstract - This paper describes the integration of a TCP-friendly mechanism into RTP. We discuss why the integration is favorable over a solution in which RTP and TCP-friendly mechanisms are separated from each other. During the design of the TFRC integration we took into account that caches should be integrated to create a TCP-friendly and scalable video distribution system. With the notion that caches are often placed very close to clients and that these clients will be in many cases connected to the caches via dedicated links (ADSL, cable modem) we present a solution where the connection between server and cache is TCP-friendly and the one between the cache and the client is not necessarily congestion-controlled. We further show how with the aid of an RTSP extension both peers of the link can negotiate the kind of transmission (congestion controlled or not), thus allowing the usage of standard and commercial clients and performing congestion control on the crucial part between server and client. In addition to the design of such an integration we have also implemented the TCP-friendly mechanism in our own streaming environment and present experimental resukts based on this implementation.

Keywords: VoD, Caching, Layered Video, TCP-friendliness

1. Introduction

In the last few years, the Internet has experienced an increasing amount of traffic stemming from the use of multimedia applications which use audio and video streaming [1]. This increase will continue and even be reinforced since access technologies like ADSL and cable modems enable residential users to receive high bandwidth multimedia streams.

The challenges of providing VoD in the Internet are manifold and require the orchestration of different technologies. Some of these technologies like video encoding are fairly well understood and established. Other technologies like the distribution and caching of video content and the adaptation of streaming mechanisms to the current network situation and user preferences are still under investigation.

Existing work on VoD has shown caches to be extremely important with respect to *scalability*, from network as well as from video servers' perspective [2]. Scalability, of course, is a premier issue if a VoD system is considered to be used in the global *Internet*. Yet, simply reusing concepts from normal Internet Web caching is not sufficient to suit the special needs of video content since, e.g., popularity life cycles can be very different [3].

Besides scalability, it is very important for an *Internet* VoD system to take into account the "social" rules implied by TCP's cooperative resource management model, i.e., to be *adaptive* in the face of an (incipient) network congestion. Therefore, the streaming mechanisms of an Internet VoD system need to incorporate end-to-end congestion control to prevent unfairness against TCP-based traffic and increase the overall utilization of the network. Adaptive streaming will also support a variety of clients (workstations, set-top boxes, handhelds) which characterize today's Internet.

2. Scalable Adaptive Streaming

In this section, we briefly describe the system in which the TCP-friendly mechanism should be integrated. We first present the architecture for Scalable Adaptive Streaming (SAS) and then present our implementation that serves as a basis for experimental work on this topic. A more detailed overview of the SAS architecture is given in [4].

2.1 Architecture

As caching method we employ so-called conditional write-through caching[1]. With conditional write-through caching a requested stream is forwarded "through" the proxy cache to the clients and the proxy cache stores the stream on its local cache if a positive decision is made by the cache replacement strategy. Subsequent clients can then be served from the proxy cache (see Figure 1). This technique reduces the overall network load in a VoD system compared to a method where the video is transported to the cache in a

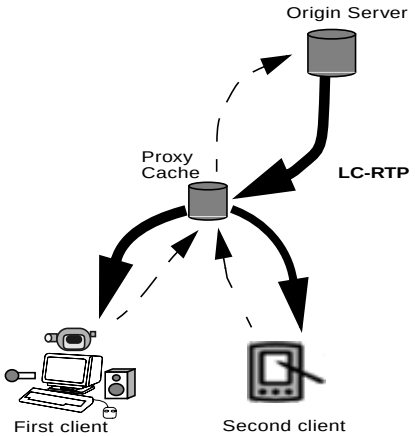

Figure 1. Video distribution.

[1] Adopted terminology from memory hierarchies.

separate stream using a reliable transmission protocol (e.g., TCP) [5]. On the other hand, conditional write-through caching requires a reliable transport protocol to recover from transmission losses. In [6], we present the design and implementation of such a protocol, called *Loss Collection RTP* (LC-RTP), which fits particularly well in a VoD architecture.

Enabling congestion control for streaming applications requires quality adaptation in contrast to elastic applications that allow a reduced transmission speed. However, this quality adaptation does not solely serve congestion control purposes but also satisfies the needs of the large variety of heterogeneous clients that exist in the Internet. Layered video represents a suitable method to allow for this quality adaptation. This offers also the ability to cache a stream in an appropriate quality, i.e., in a scenario where only handheld devices might be using a cache it is not necessary to cache the video in its best quality thus using less storage space and increasing the cache's efficiency.

2.2 KOMSSYS

KOMSSYS[2] is our own implementation of a streaming system that consists of server, proxy cache, and client. It is based on the standard protocols RTP/RTCP, RTSP, and SDP. With our ongoing work on wide-area distribution systems for AV content we decided to build our own AV streaming and distribution platform to perform further investigations. This decision is based on the experiences we made, trying to integrate our new mechanisms in already existing streaming platforms. When we started our implementation, there was only one implementation of RTSP available in open source[3]. We found out that this implementation preceded the RFC [27] and was not easily updated and reused. After some unsatisfactory experiences in adapting existing RTP/RTCP implementations for our goals, we decided to integrate our own implementation [ZGJ+00] into the system. We checked whether JMF [Car99] fulfils our needs but MPEG-1 decoders e.g. were only available for Solaris and Windows. RTP was integrated in some open source projects like vic, but a closer look at this implementations showed us that RTP is highly intergrated and was therefore not usable. In the MASH project a scalable multimedia architecture for distributed multimedia collaboration in heterogeneous environments [MBK+97] was developed. Streaming in MASH is realized by the MBone videoconferencing tools and therefore bears the same problems a described for vic.

We are also aware of a stand-alone RTP library [Lie00], but this project started after we decided to implement our own RTP. The "Darwin" project [Dar00] was published by Apple after the start of our work as well. It is concerned exclusively with the server side and supports only the QuickTime file format [Tow99]

Further examples that exploit our implementation are given in [ZGS01].In this paper we present an extension to our streaming platform by a TCP-friendly mechanism.

2.3 TCP-friendliness

It is not our purpose to develop new TCP-friendly mechanisms for streaming. In recent years, several protocols for the transport of non-TCP traffic with TCP-friendly congestion control were developed. They can be separated in mainly two classes: window-based [7, 8] and equation-based [9, 10]. Widmer et al. have published an overview of some of these approaches [22]. To be applicable for streaming over the Internet these protocols have to meet the following requirements: a) rate oscillations should be kept to a minimum and b) modification to the network infrastructure should be prevented (e.g., the protocol stack in the routers can stay as it is).

From our observations and the classification presented by Widmer et al. [22], TCP-friendly Rate Control (TFRC) is very promising as a TCP-friendly protocol for unicast streaming [2]. Recent work by Widmer and Handley [11] introduces a single-rate multicast version of TFRC but we were not able to integrate this multicast extension into our work, so far. TFRC is a rate-based congestion control protocol with good TCP-friendliness. The main advantage in combination with A/V streaming is that the rate is smooth in the steady-state case and therefore applications that rely on a fairly constant sending rate are supported.

2.4 TFRC in RTP

In this section, we motivate why we favor an integration of TFRC into RTP[4] to allow congestion controlled streaming. It was one of our goals to keep the deviation from standard RTP and RTCP as small as possible and therefore use conformant extensions if possible. However, that has not always been realizable which is mainly due to the fact that the proposed TCP-friendly mechanisms requires some significant changes to the protocol behavior. Our decision to integrate TFRC into RTP is based on the following considerations: Some functionality like the time stamping and the sequence numbering of the packets would be performed twice in a non-integrated scenario of RTP over TFRC. The amount of feedback message would increase since TFRC ACKs and RTCP message are sent separately. In addition, with separated RTP and TFRC headers some information in the headers would be redundant. Integrating RTP and TFRC would (i) reduce redundant functionality in the sender and receiver, (ii) reduce the amount of feedback messages, and (iii) minimize the overhead in the protocol

[2] http://www.kom.e-technik.tu-darmstadt.de/komssys/

[3] http://www.realnetworks.com/devzone/library/rtsp/index.html

[4] For reasons of simplicity we call this integration of TFRC in RTP RTP/TFRC in the remainder of this paper.

headers. The main disadvantage of this proposed integration of TFRC in RTP is that is can only be used by applications that already use RTP. But most of the applications TFRC is well suited for do already use RTP, like most of today's audio- and video-streaming applications. Another disadvantage is the context-change that is necessary to pass all TFRC messages into user space rather than being handled in the kernel. A new protocol (like DCP [KFH+01]) could evaluate this feedback in the kernel. In our application domain, this is probably not a fatal problem because video server performance is currently not CPU-bound

3. Related Work

The related work for scalable streaming in the Internet that involves proxy caches can be split in three categories: commercial products, theoretical resp. simulative investigations, and prototype implementations in the academic world.

[12, 13, 14] represent the first category. Unfortunately, there is only little or no technical information about these products available. One exception is Kasenna who reveal that their caching product [15] makes use of the prefix caching mechanism [16].

For the second category, we only mention very closely related work since a large amount of research has been performed in this area. In [16], the authors propose an interesting scheme of caching only the beginning (prefix) of video streams. While this allows to decrease the setup latency for clients and to accommodate variable bit rate transmission channels it does not address the scalability and adaptiveness issues. An architecture of video servers, caches and clients for layer encoded video is proposed in the work of Paknikar et al [17]. In contrast to our SAS proposal a single broker exists that handles all client requests and redirects them to the corresponding cache. Even though the usage of a broker allows to reduce the complexity of the caches it has the disadvantage that in the case of a broker failure the clients are not able to request content. The MiddleMan [18] approach differs from the others in a way that a cooperative caching mechanism is introduced where a single video stream can be stored across multiple caches. The single caches are connected via a LAN. Based on simulations the performance of this approach depending on different replacement policies is examined. [19] were among the first that proposed a proxy caching mechanism for adaptive streaming and evaluated this mechanisms via simulation.

In subsequent work, which is the first example for the third category, Rejaie et al. also implemented that mechanism in a prototype [20] and performed experiments to present some of its features. In this work their focus is mainly on the design and the implementation of a proxy cache and the goal to adaptively adjust the quality of a cached stream based on popularity and available bandwidth. The main difference to our approach is the fact that clients in their architecture always have to be able to perform congestion control. In addition, the congestion control mechanism in [20] is not integrated in RTP but set on top of it. Another implementation of a TCP-friendly partially reliable video streaming approach is presented in [21]. No proxy caches are envisioned in this architecture.

In [22], Race et al. present the implementation of a RAM based video cache which is designed for the caching of MPEG-2 streams. The usage of RAM instead of a hard disk circumvents a bottleneck on the disk's channel. DSM-CC is used for stream control and the streaming is performed in traditional, non-congestion-controlled manner via UDP. In [23], Gruber et al. present the design an implementation of a prefix cache [16] based on the standard protocols RTP/ RTCP and RTSP and necessary extensions of those protocols.

4. Implementation

In the following, we present the design and implementation of our SAS architecture that is capable of supporting heterogeneous clients.

4.1 Data Path

4.1.1 Stream Handler Extensions

The media-processing part of KOMSSYS is based on the stream handler architecture that is presented in more detail in [GZ01-1]. In this section, we will define the extensions that must be made in the stream handler architecture to support the RTP/TFRC technique. In our experimental VoD platform we developed server, proxy cache and a client which make use of the stream handler architecture. Therefore, a great deal of stream handler modules have already been designed, implemented and tested. To support RTP/TFRC in our platform we reuse these elements and if necessary extend them. Figure 4 gives an overall overview of the SHs that are involved in an RTP/TFRC streaming session. The TFRC functionality is not implemented as a new SH but as an extension of the PacketizerSH at the sender and the DepacketizerSH at the receiver. At the packetizers the aplication level framing and deframing is performed. The PacketizerSH is also responsible for the timing, i.e., it determines when the packet is sent onto the network. This extension of the packetizers was caused by the fact that the functionality of the packetizers and TFRC are very similar. At the sender e.g. in both cases the time when a new packet will be generated and sent out on the network is determined here. The only difference with TFRC is that this timing information is based on the feedback the TFRC sender received from the client rather than the encoding format that would determine the timing in an RTP-only case.

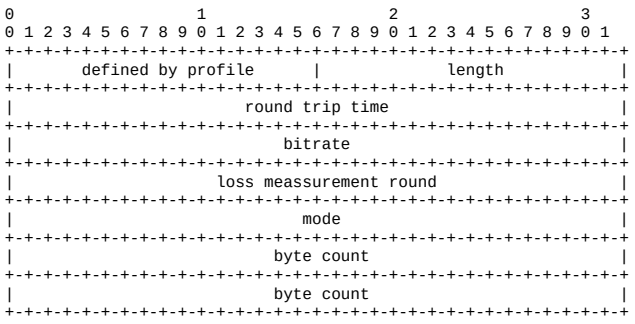

Figure 2. : RTP Header Extension

4.1.2 Packetizer and depacketizer

Several profiles for the transport of standardized audio and video formats in RTP exist [Sch96]. So far no profile for the transport of layer encoded video is available. Our experience with the development of (de-)packetizers for several audio and video formats [ZGS01] have shown that building new SH for this purpose is a rather simple task. Depending on what layer encoded video techniques should be supported it might be necessary to build more than one SH. One of our future tasks is to investigate layer encoded video proposals with respect to the needs of our system. If this work is done we will be able to decide which of the proposals should be supported and develop the appropriate packetizer and depacketizer SHs. At the moment we do onlys send dummy-content in our system.

Whenever an RTP packet should be sent out on the network the Packetizer SH will pull data from the FileSourceSH and push it to the RTPSink SH. With the TFRC extension the Packetizer SH determines the time when the next send event should occur, based on the feedback from the TFRC receiver, to achieve an appropriate transmission rate. For the layer encoded video the rate information, that TFRC provides, will also determine the number of layers that should be transmitted. The TFRC extension at the Depacketizer SH retrieves the TFRC specific information from the RTP extension header an generates a feedback message based on this information.

4.1.3 TFRC Functionality

To enable TFRC functionality in RTP some new header information is needed (see Figure 3) and part of the overall protocol behavior must be changed. Two of the additionally required header fields are already contained in the RTP header, *sequence number* and *time stamp*, respectively. We propose that the additional fields shown in Figure 3 should be put in the RTP extension header. This allows clients that have no extended (TFRC) functionality to also receive RTP packets that include TFRC information, since they simply ignore the extension header. [24] states that the extension header must be ignored by other interpreting implementations that have not been extended. As already mentioned in Section 2.2, we have, in preceding work, made an extension to RTP in order to make it reliable, called LC-RTP. This mechanism makes also use of RTP's extension header. To be able to use both, LC-RTP and integrated TFRC, we propose an extension header as shown in Figure 2.

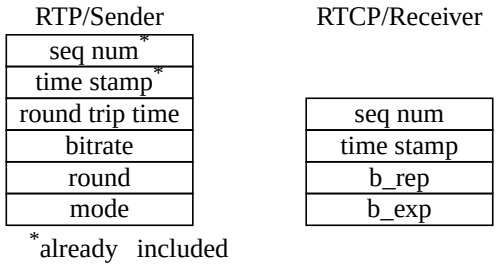

Figure 3. : Additional TFRC header fields

We also made an extension to the LC-RTP mechanism that allows two different kinds of retransmission for lost segments. This extensions will be presented in the following section.

In our implementation, which is based on a stream handler architecture, the integration of TFRC in RTP actually saves an additional timer and a buffer (see Figure 4). If TFRC and RTP would be separated, data would be retrieved from the file system by the RTP internal timer. This timer is defined by the profile of the streamed data. Since in combination with TFRC, packets are not sent by the RTP profile determined time, a buffer is needed to temporarily store the data before TFRC's own timer fires and the data is sent out on the network. The rate information provided by TFRC also determines the number of actual layers that are transmitted.

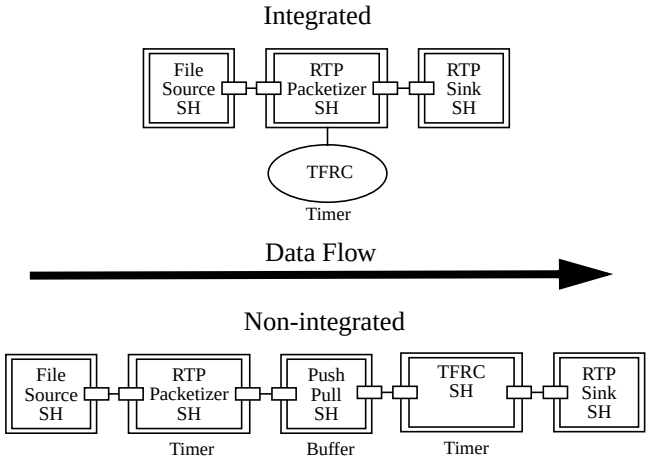

Figure 4. SH graph for the integrated and non-integrated case

4.1.4 LC-RTP Extensions

In preceding work [ZGJ+00], we proposed an extension to RTP that provides lossless transmission of AV content into proxy caches and concurrently, lossy real-time delivery to end-users. It achieves reliability by retransmitting lost

segments of a stream after the original streaming phase. This RTP extension is called loss collection RTP (LC-RTP). Due to a byte count in the LC-RTP packet header, the cache recognizes packet losses. It maintains a loss-list and after the initial transmission the missing segments are requested from the server. Thus, an exact copy of the video can be created on the proxy cache. The traffic increase is minimal because the transmission of the AV content and any caching will take place while the end-user is served.

We made an extension of LC-RTP that allows two different kinds of retransmission of missing segments. Depending on different factors like popularity of the video and kind of client it might be necessary to retransmit only the losses that occurred during the transmission or in addition transmit the segments of the layered video that were not transmitted at all (e.g .in the case of congestion). For the second case no modifications must be made to LC-RTP, since all missing segments will be (re)transmitted as with the original LC-RTP. In the first case two modifications to LC-RTP must be made:

- The sender stores a list of the segments that are really sent onto the network.
- With the aid of this list the server can identify which segments have to be retransmitted and which not. The client sends requests for retransmissions as long as all segments from its list of missing segments are transmitted or a maximum number of retries is reached. When the server notices that the client request contains only not sent segments it will send the client a BYE message to stop it from sending further requests. Since the client is already in the loss collection phase it will interpret this message differently from a BYE that is sent at the end of the initial transmission.

We decided to modify LC-RTP in this way because only minor changes are needed to enhance its functionality and no new messages must be introduced.

4.2 Control Path

The feedback of a TFRC receiver can be transported in an *application defined* RTCP packets which are intended for experimental use [24]. Using this kind of RTCP packet, as depicted in Figure 5, takes also advantage of the fact that several standard RTCP messages (e.g., receiver report) can be transmitted together with the TFRC feedback in one packet, since RTCP allows the concatenation of several RTCP packets to one compound packet. Although the usage of “stacked” RTCP packets reduces the amount of payload rather insignificantly, the I/O load at sender and receiver can be reduced considerably since the overall amount of feedback packets that must be transmitted, received and processed decreases. Thus, the additional load that is introduced by the congestion control mechanism is kept minimal. The timing for the receiver reports must be changed in a way that is based on the RTT information

```
SETUP rtsp://video.example.com/twister/video RTSP/1.0
CSeq: 1
Transport: RTPTFRC/AVP/UDP;unicast;client_port=3058-3059
```

instead of the algorithm proposed in [24]. Since we envision only unicast transmission so far in our architecture the higher amount of reports should neither restrict the raw data transmission nor cause an ACK implosion.

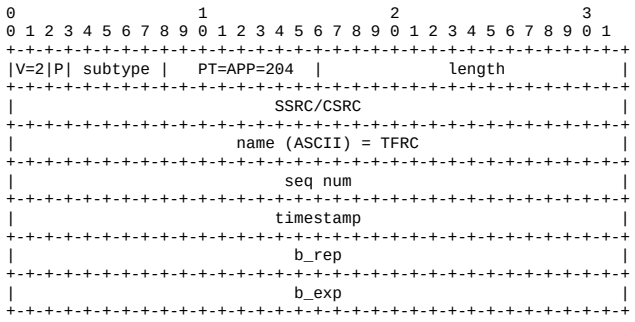

Figure 5. : Application-defined RTCP packet

4.3 Signaling

In our streaming environment RTSP is used as application signalling protocol. In this section, we show how a small extension of RTSP can be used to allow members of a streaming session to negotiate if they are capable of RTP/TFRC. To achieve this capability we introduce a new *transport-protocol* identifier in the *Transport* header, called **RTPTFRC**. This would, e.g., allow an RTP/TFRC capable client to send the following *SETUP* message and therefore initiate a TCP-friendly session.

Afterwards, the server replies with the appropriate *transport-protocol* identifier depending if it is RTP/TFRC capable or not. For the first case the answer would look as follows:

```
RTSP/1.0 200 OK
CSeq: 1
Session: 12345678
Transport:RTPTFRC/AVP/UDP;unicast;client_port=3056-3057;
        server_port=5000-5001
```

In the following section, we motivate why the capability to negotiate congestion control capability is necessary.

4.4 Putting the pieces together: Proxy Cache

Figure 8 depicts the streaming graph at the cache. The shown graph allows us to perform conditional write-through caching (Figure 2.1) due to the Packet-Multiplier-SH that creates a copy of each received segment which is stored at the local disk. Besides, this graph also allows TCP-friendly transmission between server and cache and non-TCP-friendly transmission between cache and client since the path between server and client is separated into two RTP sessions. Thus, allowing the usage of commercial clients and also providing a TCP-friendly transmission in the

backbone. Although their might also be bandwidth variations on the link between client and cache, this transmission need not always be TCP-friendly.

Based on the RTSP SETUP message (Section 4.3) the cache can determine if a client can perform congestion control or not. If congestion control is possible no feedback messages are sent from the cache but rather from the client and passed through the cache to the server. In the opposite case, feedback messages are created at the cache and forwarded to the server. In order to avoid a client overload (the client does not give any TFRC feedback in this case) the client can initially signal the kind of link it is connected to via the *Bandwidth* header field in RTSP [27], as it is, e.g., done by the RealPlayer. The information contained in the *Bandwidth* field is forwarded from the cache to the server and is used there to set the MAX_BITRATE value for TFRC which defines an upper threshold for the sender's rate. During the session RTCP feedback messages from the client are used at the cache to detect a possible client overload.

With the ability not only to send data along the defined path between the stream handlers but also to exchange signalling information in a bidirectional manner between the stream handlers, information obtained from the RTCP receiver reports can be forwarded to the TFRC receiver in order to influence the value of the expected bitrate (b_exp).

The expected bitrate is transmitted via TFRC ACKs to the TFRC sender and is used there for the calculation of the new transmission bandwidth. In future investigations we have to find a reasonable heuristic that transforms the loss information of the RTCP receiver report into a reasonable value for the expected bitrate at the TFRC receiver. To sustain the TCP-friendliness of TFRC the heuristic should not be able to increase the expected bitrate.

5. Experiments

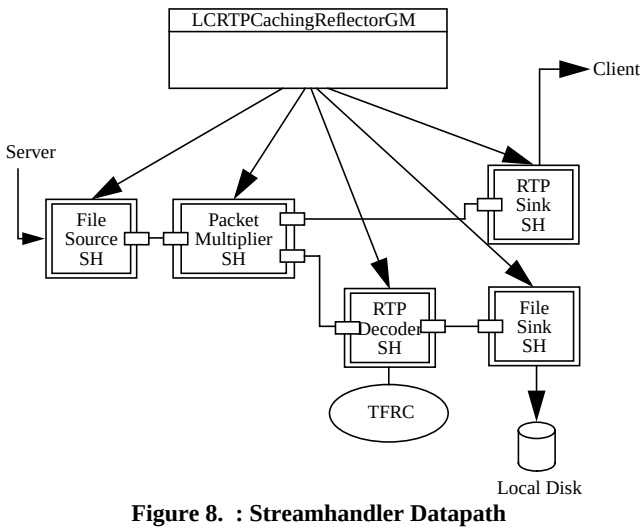

Figure 8. : Streamhandler Datapath

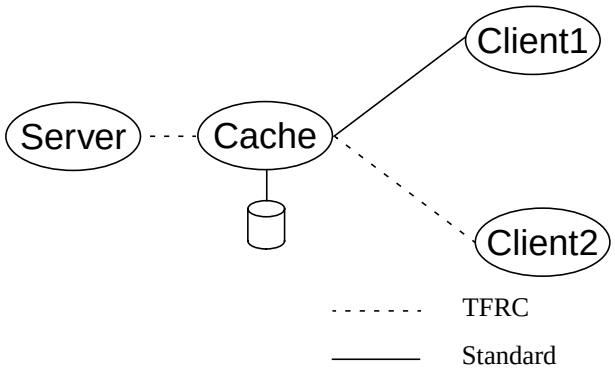

Figure 9. : Testbed

- Linux based implementation

I guess we can show that the RealPlayer would work in this scenario. (Shows that commercial clients can be used). We should show a TFRC trace between server and cache and a trace between cache and client. The both should be identical!!

6.

We present experiments and their results in the full paper.

7. Conclusions

In this paper, we presented the design and implementation of a scalable, TCP-friendly video distribution system for heterogeneous clients. The presented system allows the usage of clients with and without congestion control while a congestion controlled transmission will always be performed between server and cache. Congestion control in our system is achieved by the integration of TFRC in RTP. Next to the integration itself we also presented the benefits of such an integration in comparison to an RTP over TFRC approach.

In future work we plan to make use of real layered video instead of dummy-content which depends on the availability of an adequate codec.

8. References

[1] S. McCreary and K. Claffy. Trends in Wide Area IP Traffic Patterns. In *Proceedings of 13th ITC Specialist Seminar on Internet Traffic Measurement and Modelling*, September 2000. http://www.caida.org/outreach/papers/AIX0005.

[2] C. Griwodz. *Wide-area True Video-on-Demand by a Decentralized Cache-based Distribution Infrastructure*. PhD thesis, Darmstadt University of Technology, Darmstadt, Germany, April 2000.

[3] C. Griwodz, M. Bär, and L. C. Wolf. Long-term Movie Popularity in Video-on-Demand Systems. In *Proceedings of ACM Multimedia'97*, pages 340–357, November 1997.

[4] M. Zink, J. Schmitt, and R. Steinmetz. Retransmission Scheduling in Layered Video Caches. In *to appear at ICC 2002, New York, NY, USA*, April 2002.

[5] R. Frederick, J. Geagan, M. Kellner, and A. Periyannan. Caching Support in RTSP/RTP Servers. Internet Draft, March 2000. Work in Progress.

[6] M. Zink, C. Griwodz, A. Jonas, and R. Steinmetz. LC-RTP (Loss Collection RTP): Reliability for Video Caching in the Internet. In *Proceedings of the Seventh International Conference on Parallel and*

Distributed Systems: Workshops, pages 281–286, July 2000.

[7] S. Jin, L. Guo, I. Matta, and A. Bestavros. TCP-friendly SIMD Congestion Control and Its Convergence Behavior. In *Proceedings of ICNP'2001: The 9th IEEE International Conference on Network Protocols, Riverside, CA*, 2001.

[8] I. Rhee, V. Ozdemir, and Y. Yi. TEAR: TCP emulation at receivers - flow control for multimedia streaming. Technical report, North Carolina State University, April 2000.

[9] S. Floyd, M. Handley, J. Padhye, and J. Widmer. Equation-Based Congestion Control for Unicast Applications. In *Proceedings of the ACM SIGCOMM '00 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication 2000, Stockholm, Sweden*, pages 43–56, August 2000.

[10] R. Rejaie, M. Handley, and D. Estrin. RAP: An End-to-End Rate-based Congestion Control Mechanism for Realtime Streams in the Internet. In *Proceedings of the Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies 1999, New York, NY, USA*, pages 395–399, March 1999.

[11] J. Widmer and M. Handley. Extending Equation-Based Congestion Control to Multicast Applications . In *Proceedings of the ACM SIGCOMM '01 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication 2001, San Diego, CA*, pages 275–285, August 2001.

[12] I. Infolibbria. Media Servers and Media Proxies - the Critical Differences, 2001. http://www.infolibria.com/products/collateral/Application_Briefs/ds_ab_strea% m_media_v7e.pdf.

[13] I. Inktomi. Inktomi Traffic Server - Media Cache Option, 1999. http://www.inktomi.com/products/cns/resources/technical.html.

[14] I. Realsystems. Realsystem Proxy8 Overview, 2000. http://service.real.com/help/library/.

[15] I. Kasenna. Technical White Paper: Video Content Distribution, 2000. http://www.kasenna.com.

[16] S. Sen, J. Rexford, and D. Towsley. Proxy Prefix Caxching for Multimedia Streams. In *Proceedings of the Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies 1999, New York, NY, USA*, pages 1310–1319, March 1999.

[17] S. Paknikar, M. Kankanhalli, K. Ramakrishnan, S. Srinivasan, and L. H. Ngoh. A Caching and Streaming Framework for Multimedia. In *Proceedings of the ACM Multimedia Conference 2000, Los Angeles, CA, USA*, pages 13–20, October 2000.

[18] S. Acharya and B. Smith. MiddleMan: A Video Caching Proxy Server. In *Proceedings of NOSSDAV 2000, Chapel Hill, North Carolina, USA*, June 2000.

[19] R. Rejaie, H. Yu, M. Handley, and D. Estrin. Multimedia Proxy Caching for Quality Adaptive Streaming Applications in the Internet. In *Proceedings of the Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies 2000, Tel-Aviv, Israel*, pages 980–989, March 2000.

[20] R. Rejaie and J. Kangasharju. Mocha: A Quality Adaptive Multimedia Proxy Cache for Internet Streaming. In *Proceedings of NOSSDAV 2001, Port Jefferson, New York, USA*, June 2001.

[21] N. Feamster, D. Bansal, and H. Balakrishnan. On the Interactions Between Layered Quality Adaptation and Congestion Control for Streaming Video. In *11th International Packet Video Workshop (PV2001), Kyongju, Korea*, April 2001.

[22] N. J. P. Race, D. G. Waddington, and D. Shepherd. An Experimental Dynamic RAM Video Cache. In *Proceedings of NOSSDAV 2000, Chapel Hill, North Carolina, USA*, June 2000.

[23] S. Gruber, J. Rexford, and A. Basso. Protocol Considerations for a Prefix-Caching Proxy for Multimedia Streams. In *Proceedings of the Ninth International World Wide Web Conference 2000, Amsterdam, The Netherlands*, May 2000.

[24] H. Schulzrinne, S. L. Casner, R. Frederick, and V. Jacobson. RFC 1889 - RTP: A Transport Protocol for Real-Time Applications. Standards Track RFC, January 1996.

[25] M. Zink, J. Schmitt, and R. Steinmetz. Scalable TCP-friendly Video Distribution for Heterogeneous Clients. Technical Report TR-KOM-2002-01, Darmstadt University of Technology, January 2002.

[26] C. Griwodz and M. Zink. Dynamic Data Path Reconfiguration. In *International Workshop on Multimedia Middleware 2001, Ottawa, Canada*, pages 72–75, October 2001.

[27] H. Schulzrinne, A. Rao, and R. Lanphier. RFC 2326 - Real Time Streaming Protocol (RTSP). Standards Track RFC, April 1998.