Scalable TCP-friendly Video Distribution for Heterogeneous Clients

Michael Zink[a], Carsten Griwodz[b], Jens Schmitt[a], Ralf Steinmetz[a]
[a]KOM, Darmstadt University of Technology, Merckstrasse 25, 64283 Darmstadt, Germany
[b]IFI, University of Oslo, Gaustadalléen 23, N-0371 Oslo, Norway

ABSTRACT

This paper investigates an architecture and implementation for the use of a TCP-friendly protocol in a scalable video distribution system for hierarchically encoded layered video. The design supports a variety of heterogeneous clients, because recent developments have shown that access network and client capabilities differ widely in today's Internet. The distribution system presented here consists of videos servers, proxy caches and clients that make use of a TCP-friendly rate control (TFRC) to perform congestion controlled streaming of layer encoded video. The data transfer protocol of the system is RTP compliant, yet it integrates protocol elements for congestion control with protocols elements for retransmission that is necessary for lossless transfer of contents into proxy caches. The control protocol RTSP is used to negotiate capabilities, such as support for congestion control or retransmission.
By tests performed with our experimental platform in a lab test and over the Internet, we show that congestion controlled streaming of layer encoded video through proxy caches is a valid means of supporting heterogeneous clients. We show that filtering of layers depending on a TFRC-controlled permissible bandwidth allows the preferred delivery of the most relevant layers to end-systems while additional layers can be delivered to the cache server. We experiment with uncontrolled delivery from the proxy cache to the client as well, which may result in random loss and bandwidth waste but also a higher goodput, and compare these two approaches.

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 constant bitrate, constant quality 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, the apparent lack of desire for dedicated distribution infrastructures or a resource reservation infrastructure makes it very important for an Internet video distribution system to adhere to the "social" rules implied by TCP's cooperative resource management model, i.e., they must 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.

An additional trend in the Internet is the increase of a variety of types of clients. Only a few years ago, the typical Internet client was a standard PC connected via LAN or modem, but today Internet clients are also set-top boxes, handhelds, mobile phones or even game consoles and the number of wireless clients is increasing immensely. Their characteristics in terms of computing power, memory space and access bandwidth vary greatly, thus leading to new challenges for a video streaming architecture.

Layer encoded video ideally supports such an architecture since it allows an adaptation to link bandwidth, client processing power and buffer size, which is not possible with monolithic formats like MPEG-1. A proxy cache server can act as a node in a distribution system and address several of the problems associated with this heterogeneity. It separates the long-distance network from the access network and their distinct characteristics concerning throughput, jitter and loss. A large number of features in a proxy cache can be investigated - in this paper we focus on the ability to apply, conditionally, congestion control mechanisms separately for the server-proxy and the proxy-client link. For example, it is possible to stream from the server to the proxy with higher bandwidth than from the cache to the client.

In Section 2, we introduce our architecture on scalable adaptive streaming and give an example that shows the advantages of a cache that can split one congestion controlled session into two separated ones. In Section 3, we survey the three categories of work that is related to the one presented in this paper: products, theoretical resp. simulative investigations, and prototype implementations. Next to the conceptual aspects we present the design of our implementation in Section 4. It was our goal to integrate an optional congestion control mechanism into our existing streaming system without loosing efficiency and compatibility. Based on this implementation we conducted several experiment to prove the applicability of the proposed architecture. The experimental results are presented in Section 5. First, we show how controlled packet dropping can result in a smooth transport of the most relevant layers of a layer encoded video to a client. Second, we show the effects of an uncontrolled access network, and compare the results with the controlled approach. Both experiments are performed in the lab and in the Internet. We will see that a sustainable number of layers can be determined easily by applying the congesting control mechanism TFRC on the access link but that this happens at the cost of one to two thirds of the average quality. In Section 6, we draw conclusions from these experiments and identify future work items.

2 TARGET SCENARIO AND ARCHITECTURE

In this section, we first present an example scenario that represents a scalable video distribution system based on the characteristics of today's Internet. Based on this scenario, we briefly describe our architecture for Scalable Adaptive Streaming (SAS) and its belonging mechanisms. A more detailed overview of the SAS architecture is given in [4].

2.1 Scenario

The scenario shown in Figure 1 depicts a heterogeneous distribution system consisting of two subnets that are connected to the Internet backbone. In each of these subnets a proxy cache is located at which client requests are directed, first. Subnet A has a wireless infrastructure in which only homogeneous clients (in terms of link bandwidth) are connected, while subnet B has a heterogeneous infrastructure. In the case of subnet A it becomes clear that the quality requested by the first client will be sufficient for all requests of other clients, which is not the case for subnet B. In subnet B the content might be first requested by a handheld with a lower quality. The next client requesting this content might be a high-end PC which would like to receive the content in a better quality. Thus, leading to additional transmissions from the server to the cache in order to improve the quality of the cached content. This can be omitted if, during the first client's request, the content is streamed in a higher quality from the server to the cache than requested by the client. The cache, in this case, has additional filtering functionality that allows to store the full quality stream but forward a lower quality stream to the client. If, e.g., a four layer video is assumed, all four layers would be streamed to the cache in the case of sufficient bandwidth between the server and the cache. From the cache to the client only two layers would be streamed due to the limitations of the client.

Sending faster than the required rate on the link between the server and the cache might also be useful in the case of subnet A, because this would allow to fill a buffer in the cache faster than it is emptied for transporting the stream to the client. Thus, assuming there is enough data buffered in the cache, periods with lesser available bandwidth on the link between server and cache would not harm the quality of the stream that is perceived by the client. A buffer of sufficient size at the client might also solve this problem, but in this case the maximum available bandwidth on the link between client and cache would be the limiting factor. Streaming formats already consume most of the bandwidth on this link (e.g., the cable modem downlink in our experiment (Section 5) has a minimum guaranteed bandwidth of 384kB/s) and therefore, prevents much faster streaming than the required minimum rate.

With a cache that includes the functionality described above also standard RTP/UDP clients can be used. Since the congestion controlled connection is already subdivided in two parts and the transmission speed in both parts can be different, the second part (between cache and client) can be replaced by a standard RTP/UDP streaming session. Thus, it is possible to perform congestion control in the most critical part of the path between server and client and use commercial clients that are available today. Congestion control on the link between cache and client must not necessarily be performed due to the fact that clients are in many cases connected to the caches via links with minimum bandwidth guarantees (ADSL, cable modem).

2.2 Architecture

Like most other scalable distribution architectures ours consists of servers, caches and (heterogeneous) clients. In this paper, it is not our goal to optimize video streaming by investigating several caching concepts (hierarchical, cooperative). We are rather interested in transport issues related with TCP-friendly streaming and caching which can be used by either hierarchical or cooperative caching. This also means that we do not investigate cache replacement strategies like it was, e.g., done in [2] which is orthogonal to our work.

In our SAS architecture, the process of storing and forwarding in the cache is performed as conditional write-through caching. With conditional write-through caching[1] 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 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.

To be adaptive in the case of network congestions, requires a congestion controlled streaming that overcomes the shortcomings of TCP in the case of video streaming. In recent years, several TCP-friendly mechanisms have been proposed. In Section 4, we show how one of this proposals can be integrated in our streaming architecture and in Section 5 results from experiments in which this mechanism was used for congestion control are presented.

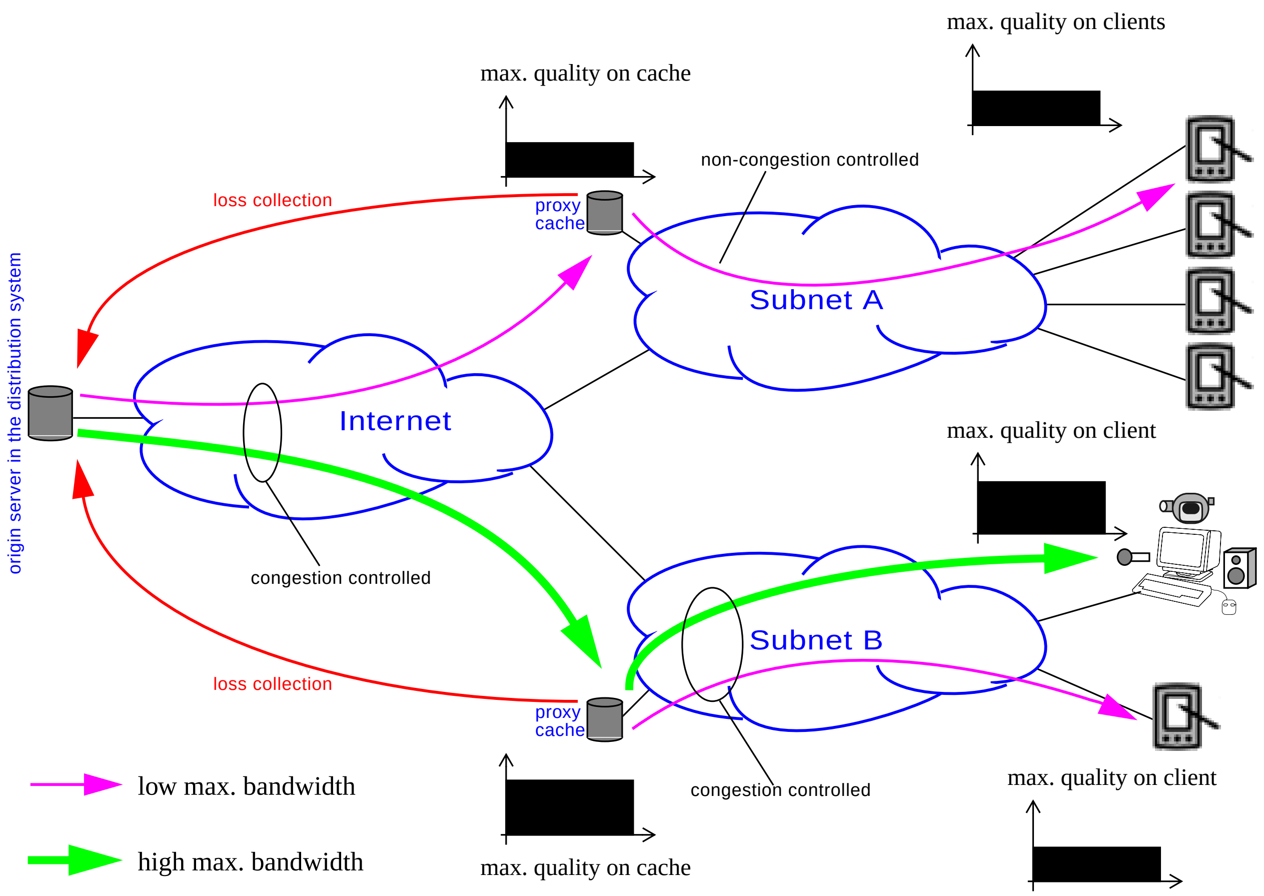


Figure 1: Scalable video distribution. system for heterogeneous clients

1. Adopted terminology from memory hierarchies.

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. If, on the other hand, a variety of heterogeneous clients is served by a cache, it can be more efficient to obtain a cached object with the highest possible quality, even though the requesting client can only deal with a lower quality stream (due to network or processing power conditions).

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.

[7, 8, 9] 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 [8] makes use of the prefix caching mechanism [10].

For the second category, we only mention very closely related work since a large amount of research has been performed in this area. An architecture of video servers, caches and clients for layer encoded video is proposed in the work of Paknikar et al [11]. 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 [12] 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, which is in contrast to our architecture that is intended for wide-area networks. Based on simulations the performance of this approach depending on different replacement policies is examined.

In subsequent work, which is the first example for the third category, Rejaie et al. [13] mainly focus 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 [13] 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 [14]. No proxy caches are envisioned in this architecture.

In [15], 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.

4 IMPLEMENTATION

In the following, we present the design and implementation of our SAS architecture that is capable of supporting heterogeneous clients. The implementation described here is based on our KOMSSYS[2] streaming systems which was built during our research on wide-area distribution systems. In a basic version [16], it consists of server, proxy cache, and client and is based on the standard protocols RTP/RTCP, RTSP, and SDP. To verify our new ideas for scalable distribution systems it is constantly extended by new functionality and used for experiments. LC-RTP [6] and *Gleaning* [2] are two examples for new functionality that has been integrated in to KOMSSYS.

In the remainder of this section, we at first present the modifications and new functionality for the data path related part of the streaming system, do the same for the control path, and show how all the new pieces are orchestrated in the cache to offer the new functionality.

2. http://komssys.sourceforge.net

4.1 Data Path

4.1.1 TCP-friendly Congestion Control Mechanism

In [17], we already justified our decision to use TCP-friendly rate control (TFRC) as congestion control algorithm for streaming in our architecture. We also presented how the integration of TFRC in RTP can result in a congestion controlled streaming mechanism. For that reason, only a short overview of the TFRC functionality in our implementation is given. For the interested reader we refer to [17].

To enable TFRC functionality in RTP some new header information is needed (see Figure 2) 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 2 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. [18] 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 on the right hand side of Figure 2.

RTP/Sender	RTCP/Receiver
seq num*	
time stamp*	
round trip time	seq num
bitrate	time stamp
round	b_rep
mode	b_exp

*already included in the RTP header

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      defined by profile      |            length              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        round trip time                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                            bitrate                            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     loss measurement round                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                              mode                             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                   byte count (used for LC-RTP)                |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                   byte count (used for LC-RTP)                |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 2: Additional TFRC header fields (left) and RTP extension header (right)

In our implementation TFRC enforces bandwidth limitations on RTP by making the timing decisions for the RTP packaging stream handler, named RTPEncoderSH. Usually, this stream handler chooses the delays between packet generation, because it is the only component in the system that can make payload-dependent decisions including the appropriate time between packets. With TFRC, the frequency of packet generation is determined by TFRC and the encoder for each payload that is supposed to operate correctly with TFRC must be modified to cope with unexpected read operations.

4.1.2 Layer Encoded Video Format (Layer Dummy)

Before we describe the modifications to the lossless transmission into proxy caches that have to be made if a layered video encoding is used, we introduce the *layer dummy* (LD) format that was used for the experiments presented in this paper. We have used it because we first wanted to investigate whether the modifications on the data and control path proposed here meet our expectations. In a second step, we plan to integrate SPEG [19] as layer encoded video format. We intend to use SPEG as a future layer encoded video format for several reasons. SPEG is designed for a QoS-adaptive video-on-demand (VoD) approach, i.e., the data rate streamed to the client is controlled by feedback from the network (e.g., congestion control information). In addition, the developers of SPEG also implemented a join function that transcodes SPEG back to MPEG-1 [20] and therefore, allows the use of standard MPEG-1 players, e.g., the mpeglib[3] used in our client. Because of our integration plans, the LD format is designed with properties similar to SPEG. We assume that the format is hierarchically coded, i.e., a segment of higher layer data is worthless if the corresponding lower layer data segment has been lost. We assume a constant bit rate format, and that all layers have equal segment sizes and

3. http://mpeglib.sourceforge.net/

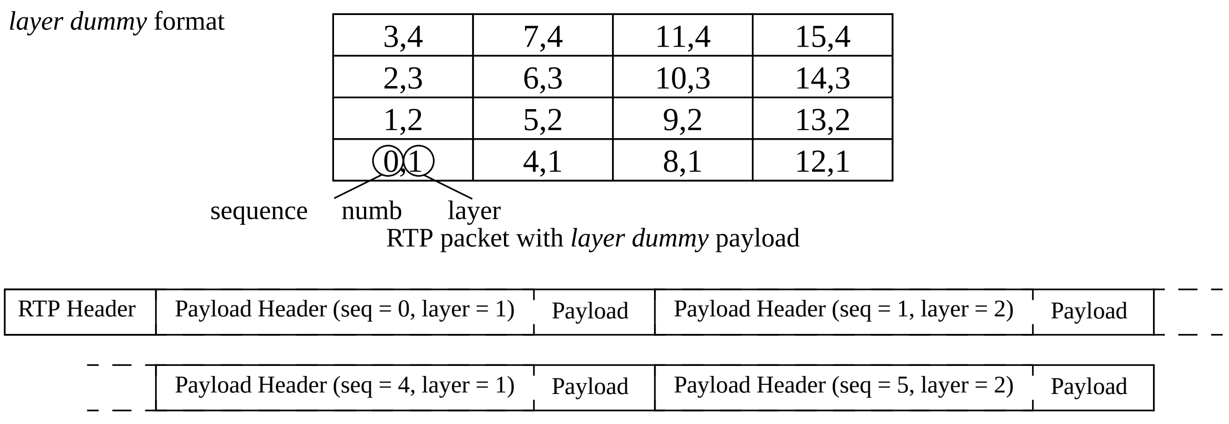

layer dummy format
3,4
7,4
11,4
15,4
2,3
6,3
10,3
14,3
1,2
5,2
9,2
13,2
0,1
4,1
8,1
12,1
sequence numb
layer
RTP packet with *layer dummy* payload
RTP Header
Payload Header (seq = 0, layer = 1)
Payload
Payload Header (seq = 1, layer = 2)
Payload
Payload Header (seq = 4, layer = 1)
Payload
Payload Header (seq = 5, layer = 2)
Payload

Figure 3: Layer dummy format and RTP packet

equal bandwidths. The RTP payload for this payload format includes a header as shown at the bottom of Figure 3, which includes a sequence number and a layer field. The latter specifies the layer of the video data that follows the payload header. The MTU size chosen by RTP can differ from the segment size, therefore, the payload of one RTP packet can contain several segments.

A payload format has to be specified for every payload type that is carried by RTP, including our *layer dummy* payload type. Since the media processing units of our implementation are based on a stream handler architecture [21] the integration of this new packetizer for our *LD* format was straight forward. We have introduced new codecs into the RTP encoder and RTP decoder stream handlers by adding codec modules for encoding and decoding that can be instantiated by a codec factory (see Figure 4). In contrast to our other codecs, the layered codecs can not assume that ordered, linear delivery of segments is the general case. For our experiments, however, we require just the conversion from and to an internal structure without the dummy payload. The DummyLayerSourceSH simulates reading from file. It generates segments at playout speed with increasing segment numbers and a cyclic assignment of layer numbers. If packets are requested faster than they are generated, the source yields data at most at playout speed. When packets are sent slower than they are generated and added to a queue with limited length. If the limit is exceeded, packets are discarded from the queue, starting with the highest layer number. On demand, the DummyLayerSinkSH writes arrival time, segment number, layer number, and segment length to a log file. The existing PushPullSH, which implements a queue that separates an active producer SH from an active consumer SH, was extended to allow limiting the queue length and controlled dropping of packets with LD payload.

For our experiments we decided to use a four-layer format as shown at the top of Figure 3, where the first digit specifies the sequence number and the second the layer of the segment. The maximum queue lengths in DummyLayerSourceSH and in PushPullSH were set to 10.

4.1.3 LC-RTP Extensions

In preceding work [6], 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

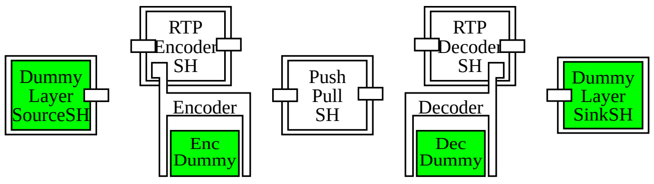


Figure 4: New building blocks for the layer dummy format

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. In the case of layered video streaming the byte count information is not sufficient to identify lost layers of a frame, but in our case the *payload header* (see *Section 4.1.2)* carries the relevant information. To keep track of the lost segments the loss-list must slightly be modified to store information about the sequence and layer numbers. With this information the receiver can, during the retransmission phase, exactly request segments that were not received during the initial streaming phase.

In addition, we extended LC-RTP such that two different types of transmission of missing segments are possible. Depending on different factors like popularity of the video and kind of client it might be necessary to transmit 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 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 transmitted and which not. The client sends requests for the transmission of missing segments as long as all segments from its list of missing segments are received, a maximum number of retries is reached, or a BYE message from the server is received.
- When the server notices that the client request contains only unsent 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 and stops sending requests for retransmission.

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. In order to distinguish between the two retransmission methods we subdivide the *subtype* field of the application-defined RTCP packet as follows: There first bit is used to indicate which retransmission method should be applied (0 = only losses, 1 = losses and not initially sent segments) and the remaining four bits are used to identify the type of the packet. In Figure 5, an example transmission from the server to the cache is shown. Due to bandwidth limitations only three of the four layers are sent to the cache. During the transmission some of the segments are lost. These losses are recognized by the cache and requested for retransmission. Since the cache is not aware of the fact that the server did not sent the segments belonging to the fourth layer, those segments are also requested. But the server has kept track of the segments that have been sent to the client initially and retransmits only those segments. If the client requests only packets that have not been sent the server sends an RTCP BYE Message and the retransmission phase is terminated.

4.2 Control Path

The feedback of a TFRC receiver can be transported in an *application defined* RTCP packet which is intended for experimental use [18]. Using this kind of RTCP packet, as depicted in Figure 6, 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 to a minimum. The timing for the receiver reports must be changed in a way that is based on the RTT information instead of the algorithm proposed in [18]. 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. Modifying the rules for RTCP feedback messages is currently also discussed by the IETF [22].

4.3 Signaling

4.3.1 Transmission type negotiation

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 a congestion

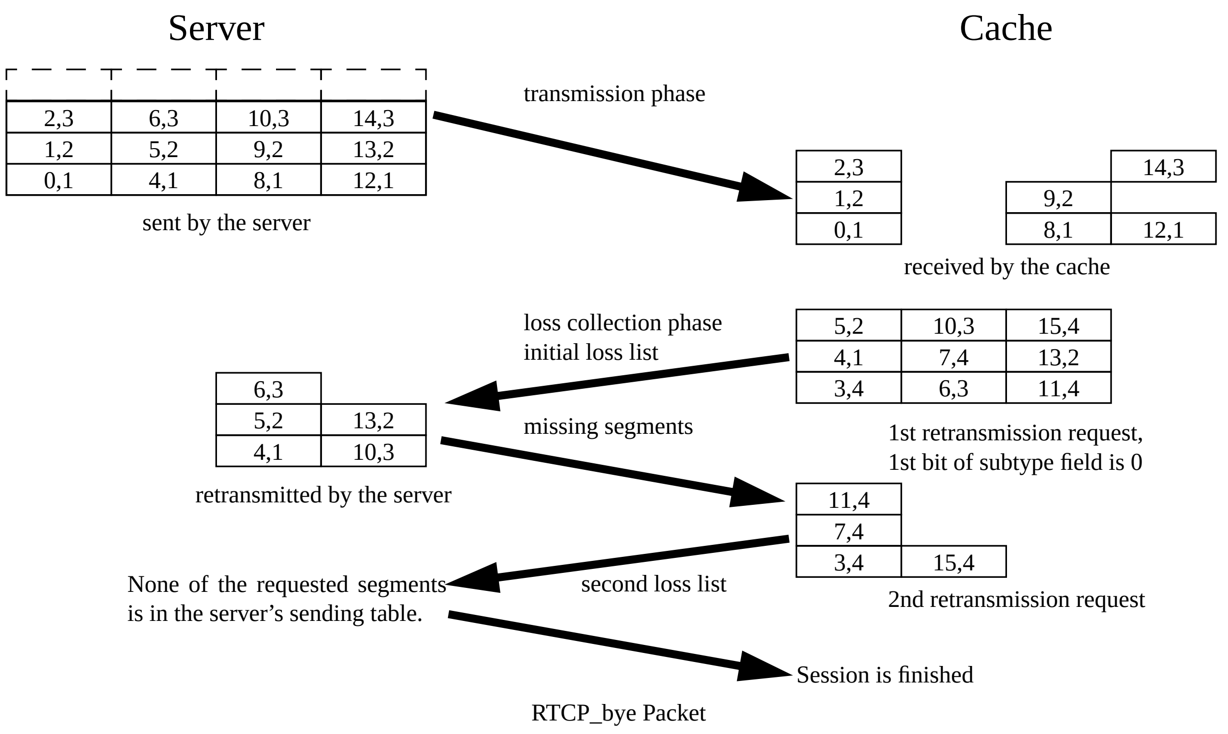


Figure 5: LC-RTP for layered video

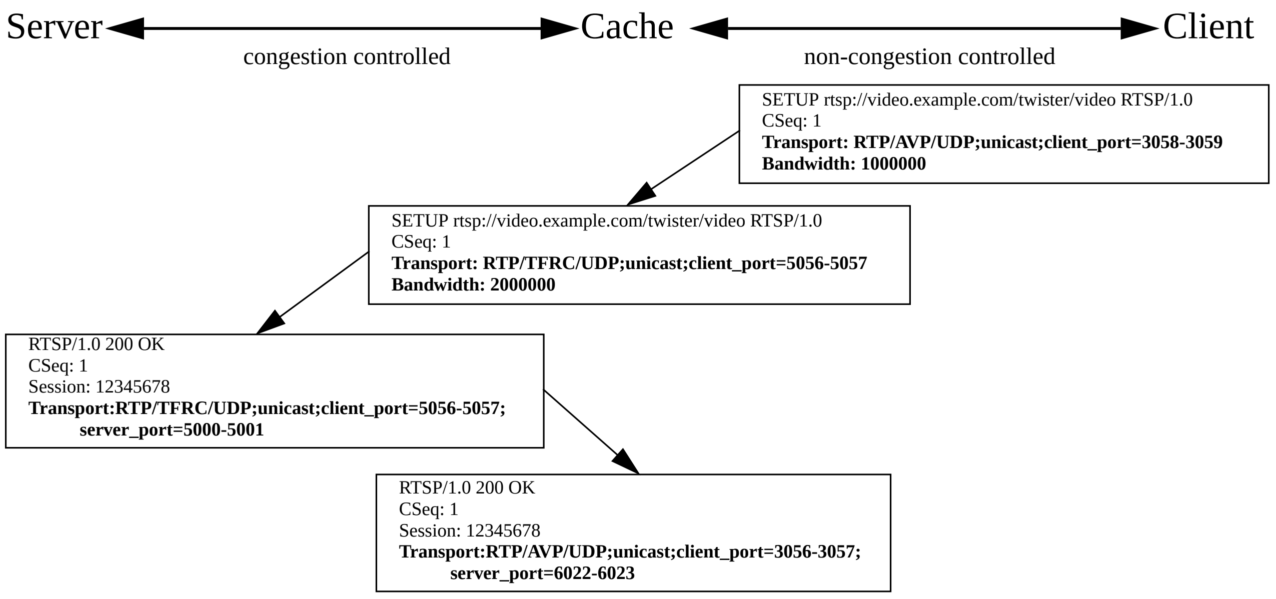
```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|V=2|P| subtype |   PT=APP=204  |             length            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           SSRC/CSRC                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       name (ASCII) = TFRC                     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                            seq num                            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           timestamp                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             b_rep                             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             b_exp                             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 6: : Application-defined RTCP packet

controlled streaming session or not. To achieve this capability we introduce a new *transport-protocol* identifier in the *Transport* header, called **RTP/TFRC/UDP**. This would, e.g., allow an non-RTP/TFRC-capable client to send a *SETUP* message as shown in Figure 7 and therefore, initiate a non-congestion controlled session between itself and the cache. The cache on the other hand can modify the *transport-protocol* identifier and therefore initiate a congestion controlled session between itself and the server.

If the cache is capable of establishing congestion controlled sessions towards the server and the client, the tag will be kept unchanged and forwarded to the server. An equally extended server replies with the appropriate *transport-protocol* identifier depending on whether it is RTP/TFRC capable or not. The cache modifies the identifier according to its own capabilities and the clients initial request and forwards the message towards the client. A complete communication between server, cache and client regarding the RTSP *Setup* message is shown in *Figure 7*.

Since the cache should also support clients that are not capable of congestion control (see Section 2.1) the cache can automatically insert the *transport-protocol* identifier in the setup message that is forwarded to the server and remove it



```
SETUP rtsp://video.example.com/twister/video RTSP/1.0
CSeq: 1
Transport: RTP/AVP/UDP;unicast;client_port=3058-3059
Bandwidth: 1000000
```

```
SETUP rtsp://video.example.com/twister/video RTSP/1.0
CSeq: 1
Transport: RTP/TFRC/UDP;unicast;client_port=5056-5057
Bandwidth: 2000000
```

```
RTSP/1.0 200 OK
CSeq: 1
Session: 12345678
Transport:RTP/TFRC/UDP;unicast;client_port=5056-5057;
        server_port=5000-5001
```

```
RTSP/1.0 200 OK
CSeq: 1
Session: 12345678
Transport:RTP/AVP/UDP;unicast;client_port=3056-3057;
        server_port=6022-6023
```

Figure 7: Modified *Setup* messages

from the server's reply before it is forwarded towards the client. Thus, allowing a congestion controlled stream between server and cache and a standard RTP/UDP stream between cache and client.

It must be mentioned that the mechanism to negotiate a congestion controlled session between two instances based on TFRC does not exclude the use of other TCP-friendly mechanisms. E.g., in the case of TCP emulation at receivers (TEAR) [23], which is another proposal for TCP-friendly streaming, the *transport-protocol* identifier **RTP/TEAR/UDP** could be used. This would, in addition, allow two peers to negotiate the TCP-friendly mechanism that should be used, if they are capable of more than one, since the transport request header field may contain a list of transport options acceptable to the client [24].

4.3.2 Bandwidth negotiation

An additional information that is signaled by RTSP is the maximum bandwidth at which a stream can be sent to the client or the cache. For this purpose, RTSP provides the *Bandwidth* request header field, that describes the estimated bandwidth available to the client, expressed as a positive integer and measured in bits per second [24]. In the case of our implementation this header field is also added to the *SETUP* header and used by either the server or the cache to limit the maximum bandwidth TFRC uses to stream the content. In Section 2.1 we have described the functionality that the stream between the server and the cache is streamed faster than its default rate and the cache serves as a buffer for the client. To achieve this, the information in the *Bandwidth* request header field is modified by the cache. The *Setup* signaling for this case is shown in *Figure 7*.

4.4 Putting the pieces together: Proxy Cache

Figure 8 depicts the streaming graph at the cache including both alternatives for congestion controlled and standard RTP/UDP-based streaming towards the client. Conditional write-through caching is enabled by the use of the Packet-Multiplier-SH that creates a copy of each received segment, which is then stored at the local disk. This part of the streaming path is only activated if a positive decision to store the video in the cache is made by the caching strategy. The two alternative paths that handle the data forwarding towards the client are created based on the RTSP information that is received from the client. If the client RTSP *SETUP* message contains the RTP/TFRC transport protocol identifier, the upper path in Figure 10 is created, if not (in the case of RTP/AVP) the lower path will be created. Recall from Section 4.1.2 that the PushPullSH in the upper path implements a queue with limited length that allows controlled dropping. In the lower path, all data is forwarded as it arrives at the cache.

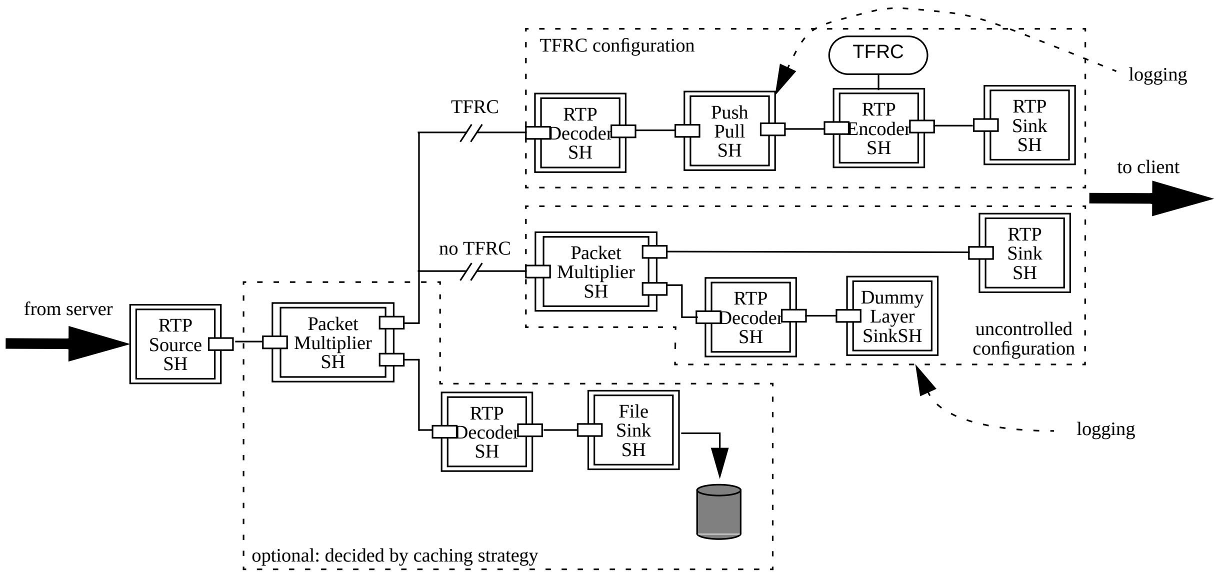


Figure 8: Proxy configuration for TFRC downlink and for uncontrolled downlink

Both paths have additional functionality that is only needed for the measurement in our experiments. For this special case we added logging functionality in both paths. If logging is requested, a log entry is generated in the respective trace files for each layer of a frame that is either received at the cache or the client.

5 EXPERIMENTS

5.1 Setup

The goal of the experiments is to verify that the functionality described in Section 2.1 can be realized by an implementation. Therefore we extended our streaming platform KOMSSYS by the functionality described in Section 4. We conducted two sets of experiments. The lab test environment is shown in Figure 9. The five systems used were standard Pentium-III PCs (850 MHz) with 256 MB of RAM and Linux 2.4 as operating system. During the experiment, KOMSSYS server, client and cache were used. The use of the two NISTNet[4] network emulators allowed us to reduce the link conditions between server and cache and between cache and client from the original capacity of a switched 10 MBit/s Ethernet. The measurements described in the following section were executed in an environment without additional network traffic, since the testbed was not connected to any other network. For the experiments we chose a bandwidth of 1 MBit/s for the server-cache link, and 512 kbit/s for the cache-client link. The bandwidth of the layer dummy video at full bit rate was 1 Mbit/s, with each layer making up an equal share of 256 kbit/s.

To complement the artificial network setup with real world data, we have also performed the measurements over the Internet between Oslo and Darmstadt. The setup for those measurements is shown in Figure 10. The test machines were different Pentium-III Linux 2.4 machines. The network was fairly unloaded during the test.

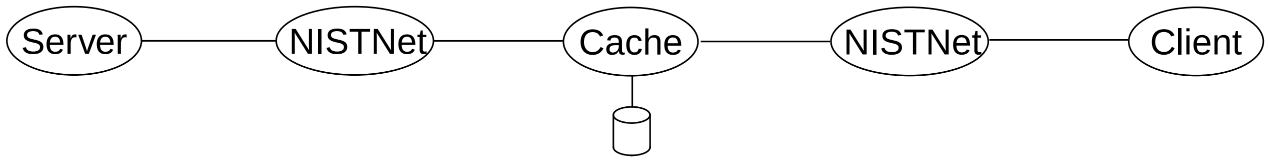


Figure 9: : Testbed setup

4. http://snad.ncsl.nist.gov/itg/nistnet/

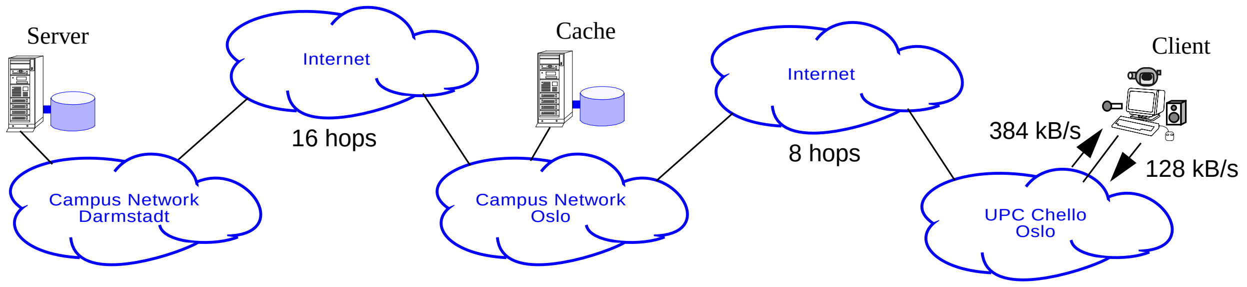


Figure 10: Internet-based measurement

5.2 Measurements

In the following we present measurement results from both test setups. The results are shown in graphs that depict the development of received quality at the proxy cache server and at the client, respectively. We measure the quality in layers although this does probably not reflect a linear relevance of equally sized layers. Since we consider only hierarchically layer codecs, the loss of a segment with a lower layer number means implies that all arriving segments with higher layer number that belong to the missing segment, have to be considered lost as well. Under this consideration, we create one graph that shows the valid, arrived layers for every individual frame in the video. The frequency of quality changes can make it impossible to visualize the average number of layers that are received. To provide this intuition, we present a second graph, which present the same quality in terms of layers once for each frame in the movie, but instead of just showing the quality for the individual frame, the average of the frame and the previous 100 frames is shown. This presentation hides the activity of short-term quality changes, but it makes it easier to identify mid- and long-term changes in the quality development. For all measurements the streaming between the server and the cache was congestion controlled.

5.2.1 TFRC in the access network

In this experiment, the client demands less bandwidth from the proxy than the proxy demands from the server. This allows the proxy cache to be filled more completely, and clients that request the same content later will be able to receive higher qualities as well. In Figure 11 we present the lab test based results for this experiment.

The detailed observation of layers received at the client shows a very low loss rate for packets that contain layer 1 and layer 2 data. This implies that TFRC chooses a sustainable bit rate, which can in this case sustain a load of 2 layers. Additional packets of layer 3 are inserted into the stream for bandwidth probing, and have a considerable probability of getting lost.

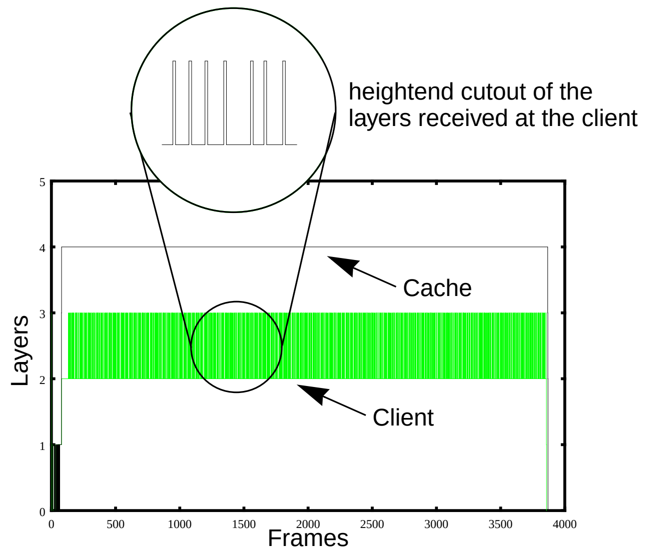
The use of TFRC allows us to use this additional bandwidth above the sustained bandwidth in any application-defined way. It is inappropriate to use it for predistribution of mandatory packets. The high loss probability would make the additional use of retransmission necessary. For a hierarchically encoded layered video, however, this additional layer, that is too unstable to be used for transmitting another layer, may be used to implement retransmission (as proposed for fair share claiming [17]) or forward error correction for the base layer.

In examining all scenarios of TFRC use, we observe a strong instability in the early phase of the connection, compared to the stable operation when the appropriate bandwidth has been found. Since this start phase is unexpectedly long, it seems reasonable to enter the initial slowstart phase with the bandwidth of a previous connection rather than a full slowstart [25].

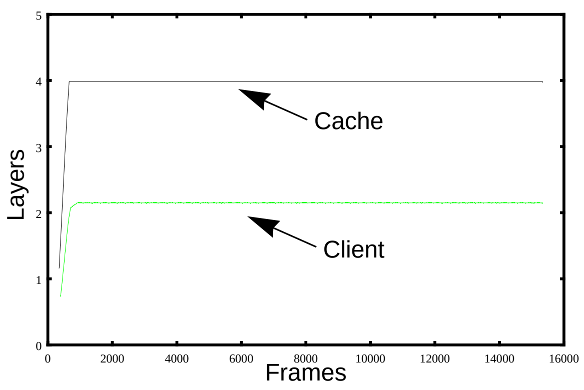
5.2.2 UDP in the access network

Second, we want to investigate the effects of using UDP in the access network combined with TFRC in the backbone. This question is legitimate for several reasons.

First, bandwidth in access networks that use DSL or cable modems is frequently restricted by shaping but provides a guaranteed minimal throughput in the provider's network. If proxy servers are deployed at the ISPs'site, the customers should be able to exploit this guaranteed bandwidth: an additional consideration of TCP friendliness in the access

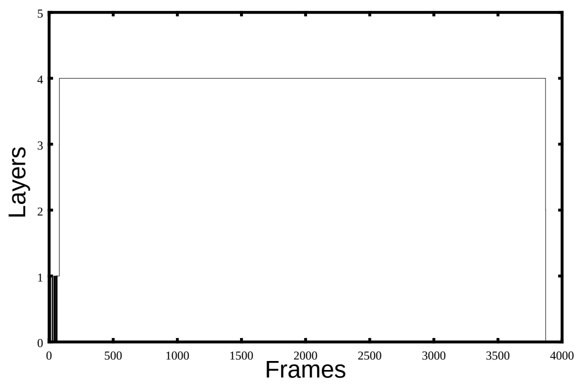


Received layers at cache and client

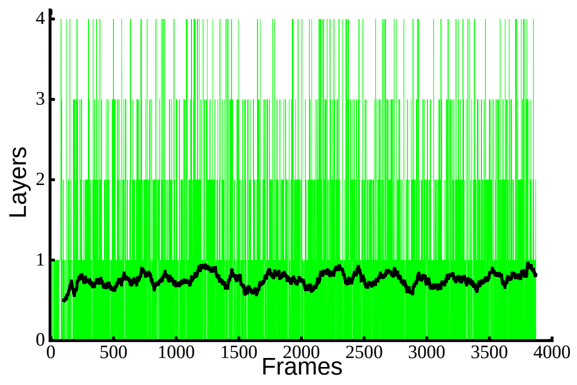


Average number of layers per 100 frame interval

Figure 11: Congestion controlled by TFRC in the access network (testbed)



Received layers at the cache



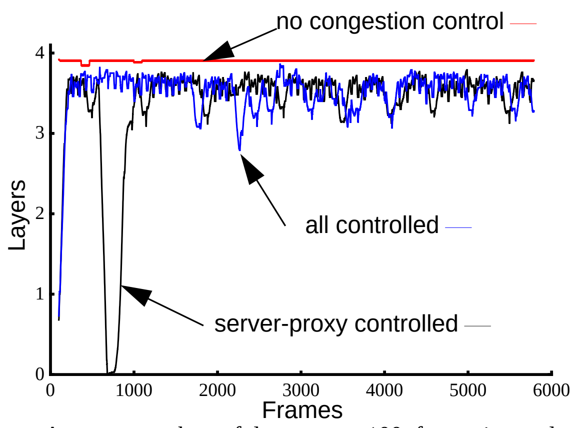
Received layers at the client:
individual number of layers received for each frame, overlaid with 100 frame average

Figure 12: Uncontrolled UDP in the access network (testbed)

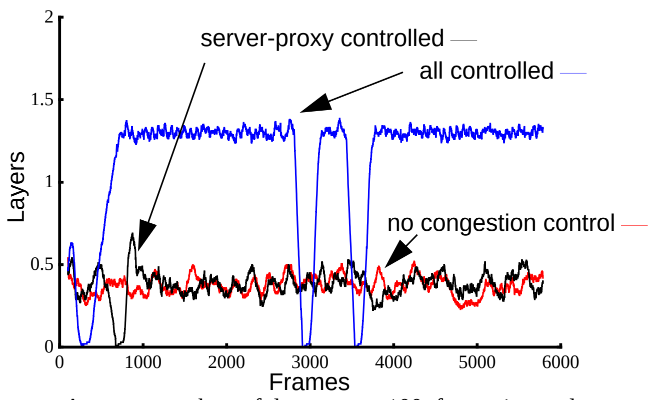
network is not required. Second, standard-compliant clients don't use TFRC. Thus, it is important to understand the interplay of using TFRC in the backbone and regular RTP/UDP in the access network.

The results of our experiments showed that when the access link becomes the bottleneck link, random loss occurs in case of UDP. This should be compared to the controlled loss that is possible with TFRC. It becomes apparent from the detailed graphs in Figure 11 and Figure 12 that the UDP approach suffers from a problem. The rate of valid packets is not sustained, which results in frequent quality changes. Since the access link can support approximately two layers and the cache forwards approximately 4 layers, the probability of a successful arrival of a layer 1 block is 0.5 and a layer 1 and a layer 2 block 0.33. This yields an estimated average number of 0.66 valid layers. The results in Figure 12 support this estimation.

When we compare the lab results with the real world traces in Figure 13, the difference in TFRC performance between the lab test and this specific kind of access network becomes visible. Whereas the congestion controlled approach yields about 2/3 of the uncontrolled throughput on the access link, it is just 1/3 in the real-world scenario. This limited throughput is likely due to artificial limitations of the TCP throughput to 384 kbit/s, which is defined in the service

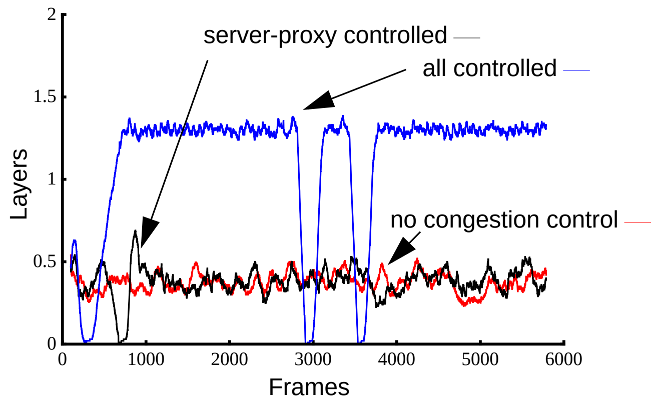


Average number of layers per 100 frame interval received at the proxy

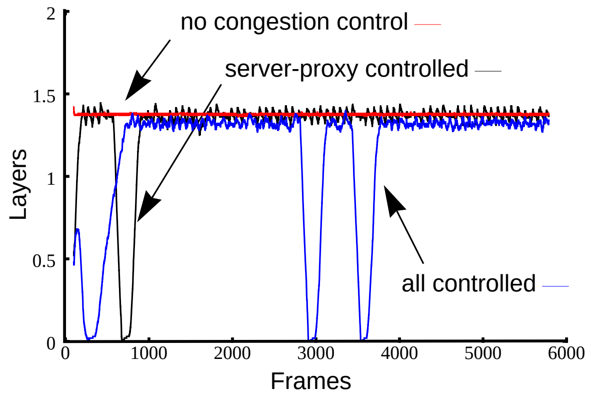


Average number of layers per 100 frame interval received at the client

Figure 13: Results for different control approach (real-world traces)



Average number of layers per 100 frame interval received at the client for hierarchical layered video



Average number of layers per 100 frame interval received at the client for non-hierarchical layered video

Figure 14: Hierarchical vs. non-hierarchical

agreement. However, we observe in both scenarios that the short-term quality changes with congestion control are considerably lower than without. In the observed case, it does not seem viable to use the bandwidth available between the server and the proxy cache for a complete transfer of titles into the cache, because of the contractual limitations of the downlink. However, since a bandwidth very close to the demand bandwidth is made available, it is appropriate to allocate it for faster-than-real-time transfer and retransmission.

5.2.3 Hierarchical and non-hierarchical codecs

Since these drops are entirely random, it is important to notice the disadvantage of a hierarchical layered video codec: a codec that uses independent layers would have considerably more stable results. We make the comparison with the results that a non-hierarchical codec would yield in Figure 14.

The two graphs in the figure show two alternative interpretations of the same trace files, a hierarchical interpretation on the left and a non-hierarchical interpretation on the right. Since no filtering takes place in the TFRC case unless the forwarding queue in the cache is full, there is no need to consider the preference of forwarding packets with lower layer number also in the non-hierarchical case, the number of forwarded packets would be identical even if the queue would not prefer selected packets. Obviously, the congestion controlled approach behaves identical for both kinds of encoding. Both cases that are uncontrolled in the access network achieve the same throughput in this case, which is very close to the connection’s limit. We note, however, that the completely uncontrolled case allows for a more stable bandwidth in this case, i.e., less quality changes than the congestion-controlled approach. The most appalling observation in the non-hierarchical graphs is certainly that this smoothness is not achieved for the approach that uses congestion control in the backbone and no congestion control in the access network. Although the number of packets that are forwarded onto the access link in an uncontrolled manner far exceeds the available bandwidth, the rate fluctuations of the controlled approach are disseminated to the access network. We can conclude from this that even if simple shaping is performed in a cache server that separates the backbone from the access network, a queue should be introduced into the forwarding path to hide the bandwidth fluctuations.

6 CONCLUSION AND FUTURE WORK

In this paper, we presented the architecture 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 modifications in RTP we also presented an extension to LC-RTP to support the lossless distribution of layer encoded videos into cache. In addition, we showed how RTSP signaling can be used to negotiate the type of streaming (congestion controlled or not) and maximum bandwidth between the single entities of our streaming system. Based on our implementation we conducted experiments in both a testbed and the Internet.

Several conclusions can be drawn from the obtained results. First, TCP-friendly streaming is feasible and can provide the user with an acceptable quality of the video stream. Although uncontrolled UDP transmissions consume all available bandwidth on the access link, random losses impair the quality of the transmitted video so severely in the case of hierarchically layer-encoded video that most of this bandwidth is wasted. We can see the TFRC advantage that the transmission of lower layer packets is nearly ensured, while higher layer packets are discarded. In the UDP scenario, on the contrary, we see random loss of packets, and lower layers are lost as frequently as higher layers.

For hierarchically layer-encoded video, the use of TFRC and a short, filtering queue in the cache server can lead to relatively smooth transmissions (few layer changes) and therefore, achieve a high perceived quality at the client. The alternative of using uncontrolled UDP in the access network, which is compatible with standard clients that are not able to perform congestion control in combination with a congestion controlled transmission has been shown to perform considerably worse. This seems to be a useful approach for cases in which the cache is locate in DSL or cable modem based access networks. Nevertheless, in the case of hierarchical layered video a better transmission quality can be achieved if filtering in the cache and congestion control on the access link are also performed.

For non-hierarchically layered video, the completely uncontrolled streaming case seems to have an advantage over the congestion controlled cases that control only the backbone or both backbone and access link. We conclude that it is in all cases appropriate to introduce a short queue in the cache that separates backbone and access link in order to reduce rate fluctuations in the backbone. Given such a queue, it becomes an interesting question whether shaping with slow rate adaptation may be better suited for our scenario than the frequent feedback of approaches such as TFRC. These reports generate a considerable number of packets on the uplink, which may be problematic if the links are strongly asymmetrical.

From the results obtained in the experiment we conclude that all connections should be congestion controlled, but certainly if hierarchical layered video is used. With standard clients that can not perform congestion control on the access link the usage of non-hierarchical layered video would be an option. Another possibility would be to guess the transmission rate based on the RTCP feedback and use the filter in the cache as it is done with TFRC to drop higher layers of a hierarchical layered video. In future work, we plan to implement such an approach by using the *Loss-Delay Based Adjustment Algorithm* [26] and verify its applicability by further experiments. We also plan to make use of real layered video instead of the *layer dummy* solution presented here. With such an implementation we hope also to gain information about the perceived quality by users which make use of such a video distribution system. In addition, we want to conduct further experiments to get a better insight in problems like the ones that might occur if the setting of the client's requested bandwidth is much higher than the one that is available on the congested link.

7 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 Modeling*, 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, April 2002. Proceedings of the International Conference on Communications 2002, New York, New York, USA.

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

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

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

[10] S. Sen, J. Rexford, and D. Towsley. Proxy Prefix Caching 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.

[11] 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.

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

[13] R. Rejaie and J. Kangasharju. Mocha: A Quality Adaptive Multimedia Proxy Cache for Internet Streaming. In *Proceedings of the 11th International Workshop on Network and Operating System Support for Digital Audio and Video, Port Jefferson, New York, USA*, pages 3–10, June 2001.

[14] 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.

[15] 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.

[16] M. Zink, C. Griwodz, and R. Steinmetz. KOM Player - A Platform for Experimental VoD Research. In *Proceedings of the 6th IEEE Symposium on Computers and Communications, Hammamet, Tunesia*, pages 370–375. IEEE Computer Society Press, July 2001.

[17] M. Zink, C. Griwodz, J. Schmitt, and R. Steinmetz. Exploiting the Fair Share to Smoothly Transport Layered Encoded Video into Proxy Caches. In *Proceedings of SPIE/ACM Conference on Multimedia Computing and Networking (MMCN), San Jose, USA*, pages 61–72. SPIE, January 2002.

[18] 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.

[19] C. Krasic and J. Walpole. Priority-Progress Streaming for Quality-Adaptive Multimedia. In *ACM Multimedia Doctoral Symposium, Ottawa, Canada*, October 2001.

[20] C. Krasic and J. Walpole. QoS Scalability for Streamed Media Delivery. Technical Report OGI CSE Technical Report CSE-99-011, Oregon Graduate Institute of Science & Technology, September 1999.

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

[22] J.Ott, S.Wenger, S.Fukunaga, N.Sato, K.Yano, A.Miyazaki, K.Hata, R.Hakenberg, and C.Burmeister. Extended RTP Profile for RTCP-based Feedback. Internet Draft, March 2002. Work in Progress.

[23] 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.

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

[25] J. Schmitt, M. Zink, S. Theiss, and R. Steinmetz. Improving the Start-Up Behaviour of TCP-friendly Media Transmissions. In *To appear in Proceedings of the INC 2002, Plymouth, UK*, July 2002.

[26] D. Sisalem and H. Schulzrinne. The Loss-Delay Based Adjustment Algorithm: A TCP-Friendly Adaptation Scheme. In *Network and Operating System Support for Digital Audio and Video, 8th International Workshop, NOSSDAV'98, Cambridge, UK*, July 1998.