Study of packet dropping policies on layered video

Frederic Raspall[ab], Christoph Kuhmünch[c], Albert Banchs[ab], Federico Pelizza[b] and Sebastià Sallent[a]

[a] Universitat Politècnica de Catalunya, Barcelona, Spain
[b] C&C Research Laboratories, NEC Europe Ltd., Heidelberg, Germany
[c] Lehrstuhl Praktische Informatik IV, University Mannheim, Germany
raspall@ccrle.nec.de, cjk@pi4.informatik.uni-mannheim.de, banchs@ccrle.nec.de
peliz@bigfoot.com, sallent@mat.upc.es

Abstract

Packet-dropping policies are widely used in router queues in order to prioritize some packets with respect to others. Since in layered video packets are labeled with different layers according to their level of importance, the use of packet-dropping policies giving higher priority to packets belonging to the most important layers seems to be a good approach to maintain video quality when congestion occurs. In this paper, we investigate the impact of using different packet-dropping policies for layered video. For this purpose, we have implemented such packet-dropping policies in a Linux testbed, and we have used our own layering video codec. With respect to quality, the results obtained show a strong dependency on the specific packet-dropping policy used and the type of video source; in some cases, the quality obtained with the packet-dropping policies is even poorer than if plain FIFO queuing with random packet dropping is used. We conclude that, even though the concept of using packet-dropping policies for layered video can provide significant advantages, the choice of the packet-dropping policy and the video codec to use needs to be done carefully in order to achieve the desired results.

Keywords: *packet dropping policies, layered video, multicast.*

1 Introduction

Video transmission is an increasingly important application of the Internet. With the development of the MBone, multi-point transmission of media streams became possible. Typical scenarios are, for example, multi-point video conferencing, video on demand services or tele-teaching. Unfortunately, the capacity of the Internet is still very heterogeneous because it connects high bandwidth backbone networks as well as low bandwidth dial-in lines. Common video encoding and compression techniques fail in such heterogeneous multicast scenarios because they cannot adapt the data rate of the video stream individually for each participant. As a solution to this problem layered video encoding schemes like the layered DCT approach described in [AMV96] have been developed. The idea of these schemes is to encode video signals not only into one but into several output streams.The quality of the decoded video depends on the number of streams received.

From the transport layer's point of view the question arises how the number of streams can be individually adapted according to the network resources of each participant. A well-known approach is receiver-driven layered multicast (RLM) [MJ96]. This scheme updates the number of streams periodically according to the network behavior (loss rate) experienced by each receiver. Unfortunately this scheme has several drawbacks. The most important problem is the slow response to network congestion [MG01].

In this article we follow an approach that places the adaptation task into the network layer. Instead of an end–to–end adaptation we propose a media-aware filter mechanism that drops packets in case of a network congestion. We study the feasibility of making this adaptation at the queues of the network nodes, in such a way that in case of congestion the network nodes first drop the packets belonging to the highest layers. The advantage of this approach is that it responds much faster to network congestion than RLM. This is of special relevance in the case of quickly changing network congestion due to bursty traffic, since receiver based approaches would not be able to adapt to these changing conditions. Note also that, in contrast to RLM, our approach does not require any kind of signaling.

We present three different packet discarding policies that filter incoming packets at a network node. The three policies have been integrated into the multicast forwarding module of the Linux kernel in order to eval-

Submitted to Workshop on Packet Video 2001, Korea.

uate their impact on the quality of a transmitted video signal. Our goal was to test the effectiveness of the three policies within different situations, e.g. with constant bit rate background traffic as well as with highly bursty background traffic. More specifically we were interested to find out which policy leads to the best video quality at the given network conditions.

We used a video codec that can separate the video stream into up to 20 layers. The codec is based on ITU-recommendation H.261. It scales the spatial quality of each video picture.

The rest of the paper is structured as follows. In Section 2 related work is discussed. In Section 3 we describe our layered video encoding. Section 4 discusses the packet-dropping policies and Section 5 details our test-bed. The focus of the article is Section 6 which presents the evaluation results gained with the layered video codec. The paper closes with some conclusive remarks and hints to further research.

2 Related Work

One of the most prominent adaptation mechanisms for layered media streams is receiver-driven layered multicast (RLM) described in [MJ96]. McCanne and Jacobson propose an end–to–end mechanism that transports layered video on different multicast addresses. Receivers can add layers by joining multicast groups and drop layers by leaving multicast groups. RLM works fine if the available bandwidth for each layer stays more or less constant for each receiver. Unfortunately the approach suffers from a very slow reaction time which makes it not useful to react to bursty cross traffic.

In [DRR99] Duffield et. al. propose a rate adaptation algorithm for video called SAVE. The algorithm adapts the bandwidth of a video signal to the current network load. In contrast to our approach SAVE is an end–to-end protocol, i. e., the sender reduces/increases the bandwidth of the video signal. Mosri and Ghanbari [MG01] discuss prioritized delivery algorithms for layered video over IP networks. They compare an RSVP based approach with a prioritized delivery mechanism PD which gives higher priority to packets containing video base layer information. In contrast to our approach PD monitors RTCP packets in order to decide if packets must be discarded. Nakauchi et. al. [NMA00] present a rate control scheme called network supported layered multicast NLM. NLM implements a packet-dropping policy which is based on the average input queue length as well as on the average interval length between two packet arrivals. They provide extensive evaluation results gained with a network simulator but do not study the impact of the proposed packet-dropping policy in a real test-bed with video streams.

Iatrou and Stavrakakis [IS00] provide a performance analysis of a dynamic regulation scheme for real-time traffic management. The goal of their regulation scheme is to control delay-jitter due to changing work load of routing nodes. Thus the paper has a different goal than our approach.

3 Hierarchical encoding and layered video

Common video encoding and compression techniques already allow the adaptation of the compression rate and thereby of the quality of the video to the available bandwidth. These techniques fail if a video is transmitted simultaneously to several receivers with different network capacities. In order to solve this problem, *hierarchical* or *layered encoding* schemes have been developed. The idea of these schemes is to encode video signals not only into one but into several output streams. Each stream S_i depends on all lower streams $S_0, ..., S_{i-1}$, in other words it can only be decoded together with these lower streams. Each stream adds to the quality of the transmitted video.

3.1 Encoding scheme

In order to evaluate our filtering policies we developed a simple hierarchical video encoding scheme. Our scheme scales the quality of each picture. We integrated the scheme into a codec implementing ITU-T recommendation H.261 but it can be adapted to all mayor video coding standards that operate in the frequency domain. H.261 has the advantage that its syntax is quite simple, it consists of a few syntactical elements only. Similar to JPEG, H.261 transforms each picture to the frequency domain by applying the DCT to 8×8 pixel blocks. After quantization, the coefficients are encoded into *runs*. A *run* is a tuple (z, n) that stands for a sequence of z coefficients which have a value of zero followed by a single coefficient with a non-zero value n. The runs are produced by processing each 8×8 pixel block in zig–zag order. Each *run* is encoded by using a static Huffman table.

Our layered scheme called *lH.261* comes into action after run length and Huffman encoding. Instead of transmitting the runs in a single bit-stream the runs are distributed over several layers. Since a block contains 64 coefficients a maximum number of 64 runs can be produced (this happens if all coefficients are non-zero). Thus the maximum number of layers is 64. Due to quantization it is very likely that the maximum number of runs is lower than 20.

The runs can be distributed over the layers in several ways. A straight forward approach is to equally distribute the runs among the layers. In order to gain a very fine granularity we have applied a different policy: Except for the last layer only a single run is placed into each layer. The last layer then contains all remaining runs. This approach gives us a very fine granularity. All layers but the last will produce more or less the same data rate except for the last one.

3.2 Transport

For transport over the network we designed an RTP payload for our encoding. Figure 1 shows the RTP header extension. The first six bits (`layer`) contain the layer number. For base layer packets this field has to be set to zero. The next field (`no-runs`) contains the number of runs transported on this layer. The following four bits (`unused`) are reserved for later specification and should be set to zero. The next 16 bits (`sequence number`) contain a packet counter which is incremented separately for each layer. The remaining 32 bits are used according to the payload for H.261 as defined in RFC 2032.

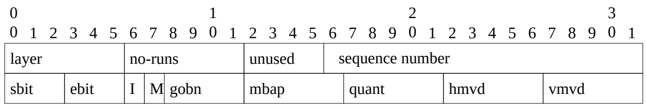

Figure 1: RTP header extension for Layered H.261 (lH.261).

4 Packet-dropping policies

Packet losses in networks occur when routers run out of space in their queues. In our approach, packet dropping policies are used to discriminate among layers when router queues are filled up. These packet-dropping policies aim at dropping the higher layers in case of congestion, in such a way that the information contained in the lower layers is always preserved. In this section we describe three different packet-dropping policies. Each of the proposed policies is an adaptation of an existing concept for packet discarding to the requirements of layered video.

4.1 Static levels policy

The idea behind the *static levels policy* is to avoid queues growing too much, i. e., it is a "preventive" policy. The queue size is controlled by means of packet drops in such a way that packets are dropped in order of importance as the size of the queue grows; the likelihood of queue overflow (random losses) is thus reduced.

In the *static levels policy*, the packet dropping as a function of the queue size is done in a static way: a packet that belongs to layer i is only kept in the queue if the size of the queue does not exceed a static value $threshold_i$. In the following, the pseudo-code for this policy is given:

```
$layer$ := read layer value of the packet;
if $queue.size < threshold_{layer}$
  then enqueue packet;
   else drop packet;
fi
```

Algorithm 1: Packet enqueue procedure.

Note that this policy is based on the same concept as W-RED [FJ93], which is the basis of the Differentiated Services architecture [Bea98]. In W-RED, there is a static threshold beyond which packets of the corresponding service class are no longer enqueued.

The *static levels policy* achieves an optimum performance in the case of constant traffic: in this case, the queue grows up to a size such that the higher layers that cannot be forwarded are dropped, and the lower layers are transmitted. In the case of bursty traffic however, this policy may drop packets unnecessarily, if the burst size is smaller than the queue length.

4.2 Adaptive threshold policy

The *adaptive threshold policy* follows the same philosophy as the *static levels policy*: queue growth is prevented by means of packet drops. With this policy, however, the packet dropping as a function of the queue size is done in a dynamic way.

In the *adaptive threshold policy* there is a layer threshold, *lthreshold*, such that a packet that belongs to layer i is only enqueued if i is below the threshold. *lthreshold* is computed dynamically every N_e incoming packets. AfterN_e new packets have arrived, the evolution of the queue length is analyzed. If the queue size has increased, this means that too many packets are being accepted, and therefore *lthreshold* is to be decreased to reduce the aggregate incoming rate. Otherwise *lthreshold* is increased. In the following, the pseudo-code for the *adaptive threshold policy* is given:

```
                                   /* called every $N_e$ packets */
$A_q$ := compute queue growth after $N_e$ packets;
if $A_q > 0$
  then $lthreshold = max(0, lthreshold - 1)$;
   else $lthreshold = min(lthreshold + 1, max\_layer)$
fi
```

Algorithm 2: Threshold update procedure.

```
$layer$ := read the layer-label of the packet;
if $layer < lthreshold$
  then enqueue packet;
   else drop packet;
fi
```

Algorithm 3: Packet enqueue procedure.

This policy is based on the same concept as CSFQ [SSZ98]: in CSFQ there is also a dynamic threshold below which packets are dropped.

The fact that this policy is dynamic leads to frequent changes of the number of layers received. Note that even in the case of constant traffic the value of *lthreshold* oscillates between two consecutive integers. Also, if the traffic conditions change drastically, this policy can be slow in adaptation because increase and decrease are in steps of 1.

4.3 Layer swapping policy

The *layer swapping policy* is based on a different concept than the two previous policies. In this policy, all incoming packets are enqueued as long as the queue is not full. However, if a packet arrives and the queue is full, the least important packet in the queue is searched and compared to the arriving one: if the arriving packet is more important, the one in the queue is removed to make room for the arriving packet; otherwise, the arriving packet is dropped. In the following, the pseudo-code for the *layer swapping policy* is given:

```
if queue not full
  then enqueue incoming packet;
   else
        layer := read the layer-label of the packet;
        oldpacket := find the packet with highest layer;
        if layer < highestlayer
           then drop oldpacket;
                enqueue incoming packet;
            else drop incoming packet;
        fi
fi
```

Algorithm 4: Packet enqueue procedure.

This policy is based on the packet swapping concept, which reacts to the situation of finding the queue full rather that preventing it, in contrast to the two previous policies . In contrast to the two policies mentioned previously, the layer swapping policy reacts to the situation of having the queue full, instead of preventing it by means of early dropping.The *layer swapping policy*, therefore, never drops packets unnecessarily, and, as a consequence, can potentially provide a higher throughput than the two other policies. However, the *layer swapping policy* also implies a higher processing overhead for every incoming packet, which may harm the throughput. The total throughput achieved by this policy is a combination of these two factors, and is platform dependent.

5 Test-bed

In order to analyze the impact of the packet-dropping policies explained in the previous section, we implemented them into the multicast forwarding of the MoPVC architecture [BGD+98], which was used to forward layered video encoded with our lH.261 scheme.

Note that even though the tests described in this paper were performed with the MoPVC architecture and lH.261, the concepts can also be applied to any other multicast architecture and any other video encoder that provides a multi-layer output stream.

5.1 MoPVC architecture

The Multicast over PVC (MoPVC) architecture has been developed by NEC in order to provide IP Multicast to ADSL networks, and is currently being used at the ADSL field-trial in Aachen, Germany, in the framework of the ANETTE project[1]. The packet-dropping functionality for layered video studied in this paper will also be included in the field-trial.

The MoPVC architecture is based on MARS/MCS [Arm96]: for address resolution a MARS server is used, while the delivery of multicast packets is done by an MCS server: when a client has a multicast packet to transmit, it sends it to the MCS which is responsible for delivering it to the members of the corresponding group.

This architecture has been implemented on a Linux platform. The most relevant part of the implementation for this paper is the MCS forwarding since in this part the packet-dropping policies were included. In the following, an explanation of the MCS forwarding is given (see Figure 2).

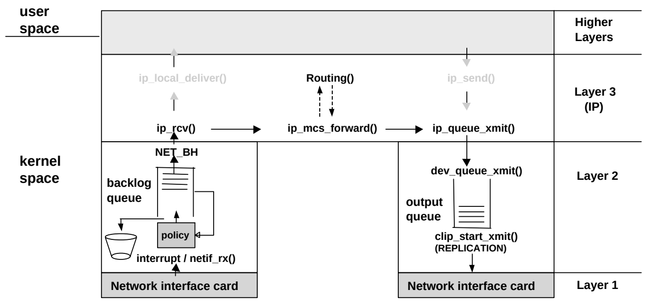

Figure 2: The course of multicast packets in the MCS forwarding.

When a packet is received by a network device, a hardware interrupt is triggered and the procedure `netif_rx()` is called. This procedure queues the packet into the backlog queue. The first time critical part of the interrupt routine, called "top-half", is finished at this time.

The second part of the routine, called "bottom-half" (`NET-BH`), is invoked by the kernel scheduler. `NET-BH` proceeds with the next packet of the backlog queue and determines the appropriate protocol to handle it (in our case IP). The routine `ip_rcv()` checks the correctness of the packet, and `ip_mcs_forward()` performs the MCS forwarding at the same level as the standard IP forwarding. From this point on, packets are handled in exactly the same way as if they were generated by the local host, invoking `ip_queue_xmit` and

[1]Url:http://www.informatik/uni-mannheim.de/informatik/pi4/projects/anette/

`dev_queue_xmit`, which queues the packet into the output queue corresponding to the network device. From this queue, the transmission of a packet is performed by `clip_start_xmit()`, which is also responsible for replicating the multicast packets to the different receivers.

5.2 Packet-dropping Policies implementation

In the above description of multicast forwarding in Linux, there are two queues where the packet-dropping queues can be inserted: the backlog queue and the output queue. Our experiments determined that the bottleneck queue – i. e., the one where packets were lost – was the backlog queue. Therefore, the packet-dropping policies were implemented in the backlog queue. This queue has a capacity of 300 packets.

The *static levels policy* was configured with 10 levels, in such a way that the threshold for level i was $threshold_i = 300 - 30i$ for level $i = 0, \ldots, 9$. Note that with this configuration, if more than 10 layers are used, packets belonging to a layer beyond 9 are lost in the first level, i. e. $threshold_i = threshold_9$ for $i > 9$.

The *adaptive threshold policy* was configured with a constant packet window of $N_e = 50$ packets, i. e., the layer threshold was adapted after every 50 packets.

5.3 Test-bed Setup

For testing the packet-dropping policies in the MoPVC architecture, we built a test-bed consisting of four Linux machines: a sender, a receiver, the multicast forwarder (MCS) and an address resolution server (MARS). The sender sent multicast video packets mixed with different types of cross traffics (CBR, bursty and video traffic) to the MCS (where the policies were installed) through an ATM network, and the MCS forwarded the multicast packets to one real client and 29 "virtual" clients through ADSL. The client recorded the resulting video stream for further analysis, while the 29 "virtual" clients were nonexistent clients that were configured at the MCS in order to experience a significant number of losses. Thus, every incoming packet at the MCS was replicated and forwarded 30 times. This configuration lead to a high load at the MCS: even for a relatively low incoming rate (4 Mbps), the number of experienced losses was large. The test-bed configuration is depicted in Fig 3.

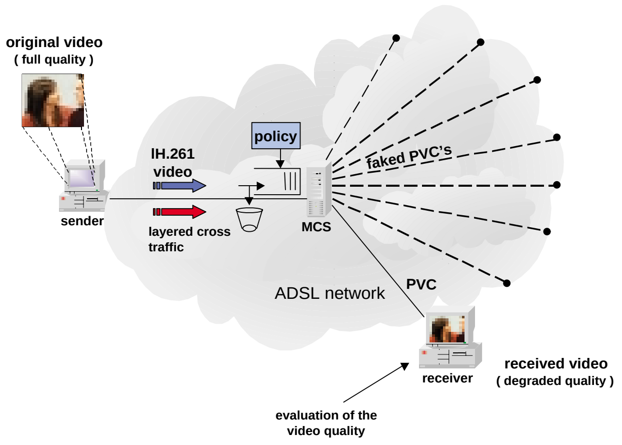

Figure 3: MoPVC test-bed for layered transmissions.

6 Performance analysis and evaluation

In order to analyze the performance of the proposed packet-dropping policies, we ran a number of experiments in the test-bed. The goals of the performed experiments were: a) to evaluate the benefits of packet dropping at network queues for layered video in terms of loss statistics, and b) to analyze the impact of the different concepts for packet dropping to the resulting video quality with our encoding technique. In order to fulfill these two goals we provide two series of evaluation results. The first evaluation, presented in Section 6.1, provides statistical measures obtained with artificial traffic that show the performance of the policies in terms of layer discrimination with respect to loses. The second evaluation, in Section 6.2 studies the perceived video quality, with objective measurements and subjective qualitative considerations, when a real video stream is transmitted.

6.1 Tests with artificial traffic

The three policies were designed to drop only the highest layers in case of congestion. In order to evaluate how well this criterion was met we ran tests with artificially generated traffic for each policy and studied its behavior. Tests were run for both CBR and bursty traffic, and the behavior of the policies was analyzed according to the measured *layer response*.

6.1.1 Layer response

Since the objective of our packet-dropping policies is to discriminate among layers in case of congestion, a natural way of measuring the performance is to analyze the percentage of successfully delivered packets in each layer. We call this measure the *layer response*.

Note that if the conventional queuing policy is applied (i. e., FIFO queuing with packet dropping at the tail), packet drops are independent of layers, and, as a consequence, we have a uniform *layer response*. On the other hand, the measured *layer response* with an ideal policy would be a step function, such that 100% of the packets belonging to the lower layers are delivered and the remaining are all dropped. The more similar is the *layer response* of a given policy to the step function, the better this policy meets its design criterion.

The performance of a policy also depends on the total throughput,i. e., the total amount of packets delivered in a time unit. The computational cost of a policy impacts the total throughput of a forwarding entity; in general, the more complex the policy, the lower the throughput.

Note that the resulting video quality is also influenced by other aspects besides the *layer response* and the total throughput, such as the dynamics of the policy. For example, the effect of losing packets belonging to the same frame is different than losing packets belonging to different frames.

6.1.2 CBR traffic

In order to evaluate how well each policy met its design criterion, we measured the *layer response* of the three policies with Constant Bit Rate (CBR) traffic and the setup. The packets had a size of 500 bytes, and four different incoming rates (2, 4, 6 and 8 Mb/s) were tested. The tests were also performed for a different number of layers (5, 10 and 15).

In Figure 4 the *layer response* of the three policies under study is depicted and compared to the obtained *layer response* when the conventional policy is used, for the case of an incoming rate of 4 Mb/s and 10 layers. In this Figure it can be observed that under CBR traffic, both the *static levels* and the *layer swapping* policies have a very sharp *layer response*, like an ideal policy. In contrast, the *layer response* of the *adaptive threshold policy* is not so sharp, due to oscillations of the $layer_{threshold}$. Note that the observed behavior when the conventional policy is used is the expected, i. e., flat *layer response*.

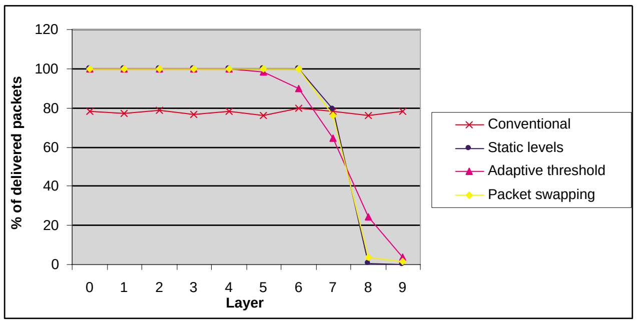

Figure 4: Layer response achieved with different policies for a 4 Mb/s 10-layer CBR traffic. The graphs show the percentage of successfully transmitted packets of each layer (e.g. about 80% of the packets belonging to layer 7 have been successfully transmitted by using the static levels policy)

The results of the tests performed with different incoming rates and different numbers of layers are not shown in this paper for space reasons. The conclusion of these additional tests is the same as the commented above, except for the case when more than 10 are used with the *static levels policy*. In this case, losses are uniformly distributed among the layers higher than 9, since all these layers share the same *threshold*.

The total throughput for each policy was also measured, obtaining similar values for all policies. An interesting conclusion from this result is the observation that the higher computational cost of the *layer swapping policy* has no impact on the achieved forwarding throughput by this policy on a Linux platform. Note that this result is platform specific.

6.1.3 Bursty traffic

Under static conditions (CBR traffic), the policies show a stable behavior, and their performance is almost optimal. In a dynamic situation (bursty traffic), it is more difficult to achieve the desired performance. In order to evaluate the capability of the three proposed policies to adapt to changing conditions, we ran the same experiments as the ones described for CBR traffic with a bursty kind of traffic. This traffic consisted of an ON/OFF process sending bursts of layered packets with a uniform distribution of layers.

In Figure 5 the *layer response* of the three policies is depicted for an incoming bursty traffic of rate 4 Mb/s with 10 layers. It can be observed that, as expected, the *layer response* under bursty traffic is less sharp than under CBR traffic, i. e., policies perform worse than in the former case. This is actually unavoidable unless the queue has an infinite length. As can be seen in the figure, the *layer response* gained with the three policies still performs much better than the *layer response* induced with plain FIFO queuing with random packet dropping. Therefore, even in the case of bursty traffic the policies are beneficial in terms of layer discrimination.

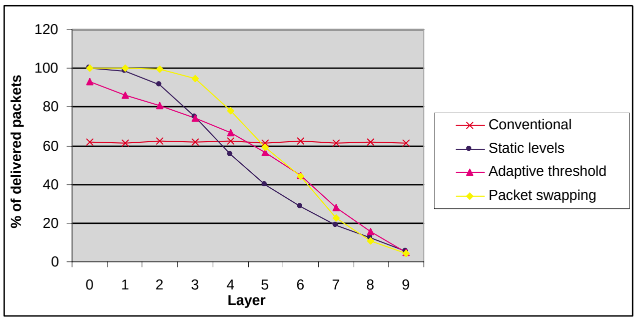

Figure 5: Layer response achieved with different policies for a 4 Mb/s 10-layer bursty traffic.

From the results shown in Figure 5, it can be seen that the policy that performs worst is the *adaptive threshold policy.* This policy proved to react too slowly to changing conditions, since its $layer_{threshold}$ changes at most one unit every N_e packets. The losses observed in the lower layers occurred when the queue was filled up due to this slow adaptation.

The *layer response* of the *static levels policy* shows a better shape than the *adaptive threshold*, but still achieves a lower total throughput than the *layer swapping policy.* This loss of throughput is inherent to any preventive policy in the presence of bursty traffic: a burst of packets smaller than the queue size with a preventive policy causes some unnecessary packet drops of the highest layers. In contrast, this situation does not cause any packet drop in a reactive policy such as the *layer swapping.*

Our conclusion concerning the tests performed with CBR and bursty traffic is that all policies behave according to their design criterion, since for all three, the probability of losing a high layer packet is much bigger than the probability of losing a low layer one. From the three policies, the one that performed best is the *packet swapping policy*; it is the one that achieved the highest total throughput and also the sharpest *layer response.* The *static levels policy* performed similarly to the *packet swapping policy* but with a lower throughput and the *adaptive threshold policy* had the worst *layer response.*

6.2 Tests with video traffic

In order to evaluate the benefits of using packet discarding policies in the queues for layered video, it is not sufficient to analyze the *layer response*, since the quality of the video traffic may depend on many other aspects. A proper evaluation requires measuring the quality of the received stream. We measured the quality of the received video using the *QM* parameter described below.

6.2.1 Objective Video Quality Measurement

In order to find out if layered video combined with filtering policies leads to an improved quality of the received video signal, we need a metric to measure the video distortion as perceived by the human observer as precisely as possible.

One measure widely used in the context of video quality is the signal-to-noise ratio (SNR). It describes the energy of an undistorted signal in relation to the noise introduced by processing the signal (e.g. compression and decompression of the signal). High SNR values do not always correspond to signals with high perceptual quality [BK97].

A number of measures exist, which try to estimate the distortion perceived by the human visual system. In order to measure the success of our packet filtering policies we chose the QMeasure algorithm (QM). QM has a temporal component which tries to measure temporal distortions such as the ones introduced by dropped frames, and a spatial component which tries to estimate the distortion of each picture. For the dynamic measure the motion energy between two succeeding images is used while the spatial component distinguishes between distortions of edges and texture. Because edges provide more important information to the human visual system the QMeasure puts a higher weight on edge distortions than on texture distortions. QM takes two sequences of frames as input parameter and returns the estimated difference perceived by the human visual system. A high difference between the two sequences results in a high QM value. If the two sequences are equal the QM value is zero. A more detailed description can be found in [KKSH01].

6.2.2 Tests

The goal of the test with video traffic was to analyze the impact of the proposed policies on the resulting video quality. For this purpose we used the lH.261 video coding described in Section 3. As the performance of a policy highly varied with test conditions, we ran tests under the following conditions:

Background traffic: One of the conclusions drawn from the tests described in Section 5 is that the performance of a packet-dropping policy highly depends on the statistics of the background traffic. Therefore, the quality of a video stream after a queue with a packet-dropping policy is very likely to be highly dependent on the statistics of the rest of the traffic going through this queue. We ran tests with three different types of background traffic: CBR, bursty and video traffic. The two former ones have been described in the previous section, while the latter consists of a number of lH.261 video streams. **Video source:** We used two different types of video sources: an action movie (star wars) and a self recorded video-conference. Since in the former there are many drastic changes of background and brusque movements, while in the latter the background is static, packet loses may have a very different impact in each case. **Number of layers:** The number of layers used in the coding determines the granularity of the encoded video. Also, the *static levels policy* is configured with a static number of levels, and, as a consequence, its performance depends on the number of incoming layers. We ran tests with 5, 10 and 15 layers.

In all tests, the total incoming rate at the MCS was 4 Mb/s. In the cases of CBR and bursty cross-traffic, the packet sizes of the cross-traffic had the same statistics as the video traffic.

6.2.3 Results

The Tables 1 and 2 show the obtained evaluation results. Each column shows the average QM value gained with the different policies. Again policy 0 stands for the conventional policy, policy 1 for the static levels policy, policy 2 for the adaptive threshold policy and policy 3 for the layer swapping policy. In general, these results confirm that the use of packet-dropping policies leads to an improvement in the video quality obtained. The fact of having a better quality when a policy is used is not surprising in itself, since it can be expected that a policy that drops high layers performs better than random dropping.

Comparing the different policies, it can be seen that in general similar results are obtained for all of them. Unfortunately, in the case of the *adaptive threshold policy* the quality measure *QM* is misleading, since it suffers from *flickering*, which is an aspect that is not captured in the *QM* parameter. The fact that the number of layers delivered by the *adaptive threshold policy* oscillates causes the resolution of the delivered video to change continuously and uniformly in the whole screen. This effect of *flickering* is annoying for the viewer and harms considerably the resulting quality. Therefore, the quality delivered by the *adaptive threshold policy* is considerably lower than the delivered by the other two policies[2].

cross-traffic	Policy 0 QM	Policy 1 QM	Policy 2 QM	Policy 3 QM
10 Layers				
cbr	1542.4	874.8	888.6	899.7
bursty	969.0	876.2	942.8	877.2
video	1281.4	941.0	950.3	952.0
15 Layers				
video	1666.8	989.9	994.9	1004.9
5 Layers				
video	1200	977.6	968	996.1

Table 1: Results of the star wars test sequence using 10, 15 and 5 layers.

The screen shots of the star wars test sequence displayed in Figure 6 depict the significant quality improvement gained by using filtering policies. Also, Figure 7, which plots the evolution of the video quality in the test with Star-wars with CBR cross traffic and 10 layers, shows how the policies inflict higher quality during the complete test video. However, a more thorough

[2]Note that this conclusion has been derived through subjective qualitative considerations, due to the limitation of the objective measurements to emulate the human perception of a video signal. In our test-bed this was the only situation in which the objective measures did not match the quality perceived subjectively

cross-traffic	Policy 0 QM	Policy 1 QM	Policy 2 QM	Policy 3 QM
cbr	1400.7	1447.8	1444.8	1445.3
bursty	1444.1	1439.9	1495.2	1442.0
video	1399.9	1454.1	1453.8	1434.6

Table 2: Results of the video conferencing test sequence with 10 layers.

analysis comparing the results leads to less obvious conclusions.

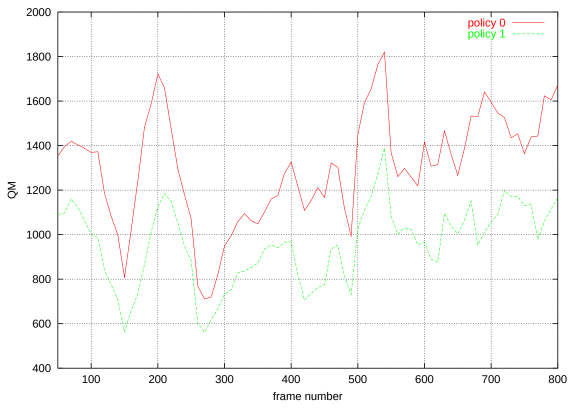

Figure 7: Development of QM with and without policies. Policy 0 stands for the "conventional policy". It can be seen that with policy 1 (*static levels policy*) a better QM is experienced in all instants as compared to using the conventional policy.

Comparing the results of Tables 1 and 2, it can be seen that the benefits of packet-dropping policies change drastically with the type of video source: while the quality with policies improves very much for an action movie video, this does not occur with a video-conference. This result is very relevant for our goal, since it implies that the use of packet-dropping policies does not always lead to an improvement in quality. The reasons for this result are explained in the following.

In an action movie, there are fast movements and frequent changes in background. Using the conventional policy leads to a degradation in the integrity of movement and errors in the background refresh as shown in Figure 8. These effects are both very harmful and cause the video quality to be extremely bad. In contrast, if more sophisticated policies are used, these effects are minimized. Even though with these policies the resolution is lower, since high layers are dropped, the resulting quality is much better.

In contrast to an action movie, in a video conference movements are slow and the background is static. In this scenario, using the conventional policy leads to a

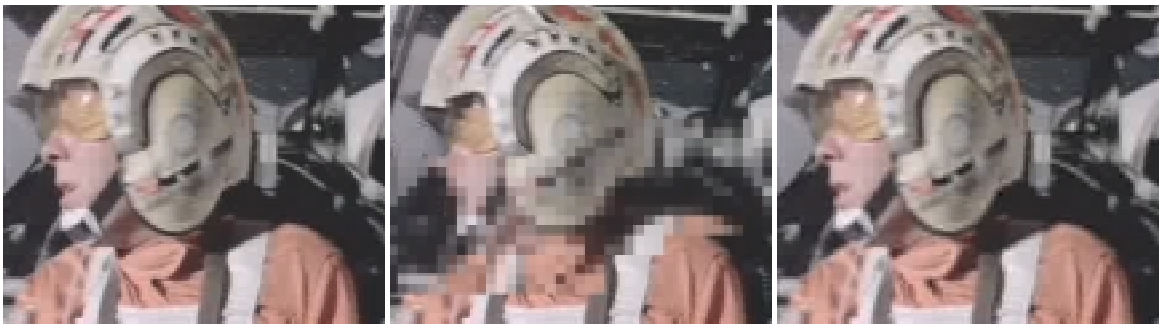
Figure 6: Decoded pictures of the star wars test sequence. Left: without packet loss, center: with the conventional policy, right: with static levels policy.

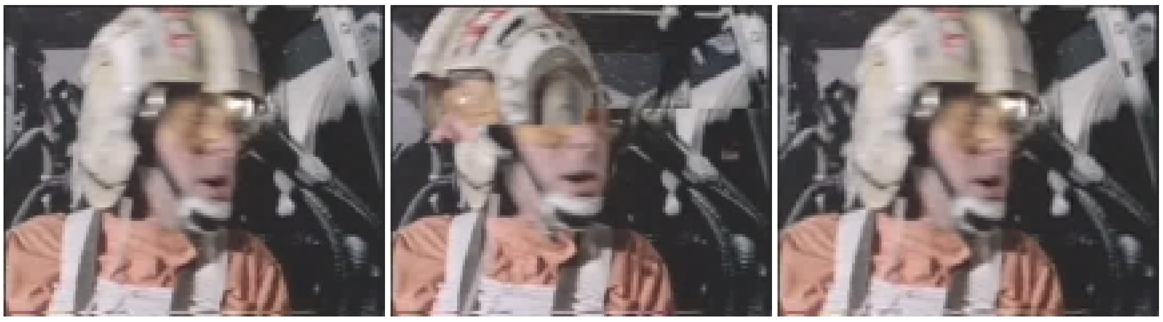
Figure 8: Missing background refresh. Left: without packet loss, center: with the conventional policy, right: with static levels policy.

high resolution image with random spots of low resolution. Since these random spots are quite tolerable for the human observer and can often be concealed by the decoder by using information from the previous frame the perceived quality is relatively good. In contrast, using policies leads to a lower resolution, since losses are concentrated on the higher layers. Also, when policies are used, the number of delivered layers oscillates due to changing traffic conditions. This effect, which was unnoticeable with an action movie, is quite annoying with a static background.

If we study the impact of background traffic in Tables 1 and 2, it can be seen that policies improve the experienced quality in all cases. It is also an interesting observation that when policies are used, the obtained quality is almost independent of the type of background traffic, in comparison with the case where the conventional policy is used. Thus, the use of policies contribute to make the video quality more uniform.

Finally, Table 1 shows the impact of using different number of layers. Results show different qualities for different number of layers, but do not show a significant impact in the comparison of the different policies. It is important to note that the *static levels policy* is not negatively affected by using different number of layers, even though this policy was statically configured assuming a number layers equal to 10.

7 Conclusions and outlook

In this paper we have investigated the combination of layered video transmission with packet-dropping policies implemented in the network queues. Layered video has traditionally been used in the context of receiver-driven decisions. The advantage of making the decisions at the network queues is that it enables quicker reaction to changing traffic conditions since no signaling mechanisms are needed.

The performed experiments have shown a significant improvement in the quality of action movie-like videos when our packet-dropping policies are used. In contrast, when the transmitted type of video from a video-conference-like, packet-dropping policies do not improve the quality substantially.

The explanation for the observed behavior relates to the used error concealment scheme: If base layer information of a block is lost then the complete block is discarded and replaced by the block from the previous frame. In case of our packet discarding policies this happens almost never. Instead information from higher layers is discarded which results in a block with lower quality. In low motion videos many blocks do not change over long periods of time. Thus it is better to conceal the lost information by simply repeating blocks from previous pictures. In contrast, in high motion videos it is better to use low quality versions. It can be assumed that better error concealment strategies will overcome this problem.

As a result of our experiments, we have detected some problems in the use of packet-dropping policies for layered video. However, in our opinion, these results do not invalidate the concept for layered video. We believe that the potential advantages of filtering layers at the network queues are worth further investigation. For example, a new layered video coding adapted to a fluctuating number of layers may solve the problems we have detected. The scope of our work, however, has been limited to the packet-dropping policies.

In our investigations, we used three different packet-dropping policies: the *static levels policy*, the *adaptive threshold policy* and the *packet swapping policy*.

The *adaptive threshold policy* provides in all cases a worse performance than the other two. With this policy, the number of layers delivered inherently fluctuates. The effect produced by this fluctuation proved to be very harmful for the resulting quality. As a conclusion, any policy that adaptive tries to find the right number of layers to forward is probably not a good option.

The *packet swapping* and the *static levels* policies provided both good performance. The packet swapping policy has the advantage that it does not modify the queuing behavior as long as there is no congestion, but this comes at the cost of a higher complexity. However, this higher complexity proved not to harm the performance of the policy on a Linux platform. For the ANETTE project, which is based ion Linux machines, we have chosen to use the *packet swapping* policy.

One possible use of the *static levels policy* could be the Assured Forwarding service (AF) of Diffserv (Differentiated Services). The packet-dropping policy used by this service, W-RED, is very similar to the static levels policy proposed in this paper. Therefore isf the different video layers were mapped into different AF code points, the resulting quality would be similar to the quality obtained in this paper for the *static levels policy*. Note also that in this scenario, the amount of traffic that each user is allowed to send in each layer is determined by the Service Level Agreement (SLA) between the user and the ISP.

References

[AMV96] Elan Amir, Steven McCanne, and Martin Vetterli. A layered dct coder for internet video. In *Proc. of IEEE International Conference on Image Processing, Lousanne Switzerland*, pages 13–16. IEEE, 1996.

[Arm96] G. Armitage. Support for multicast over uni 3.0/3.1 based atm networks. IETF, RFC-2022, 1996.

[Bea98] S. Blake and D. Black et al. An architecture for differentiated services. IETF, RFC-2475, 1998.

[BGD$^+$98] A. Banchs, M. Gabrysch, T. Dietz, B. Lange, and H. J. Stuettgen. Supporting multicast in adsl networks. In *IEEE ATM Workshop, Kochi, Japan*, 1998.

[BK97] Vasudev Bhaskaran and Konstantinos Konstantinides. *Image and Video Compression Standards*. Kluwer Academic Publishers, Norwell, MA, 1997.

[DRR99] N. Duffield, Ramakrishnan, and Amy Reibman. Issues of quality and multiplexing when smoothing rate adaptive video. In *IEEE Transactions on Multimedia*, 1999.

[FJ93] Sally Floyd and Van Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1993.

[IS00] Steve Iatrou and Ioannis Stavrakakis. A dynamic regulation and scheduling scheme for real-time traffic management. *IEEE/ACM Transactions on Networking*, 8(1), 2000.

[KKSH01] C. Kuhmünch, G. Kühne, C. Schremmer, and T. Haenselmann. A video-scaling algorithm based on human perception for spatio-temporal stimuli. In *Proc. SPIE, MMCN*, 2001.

[MG01] Alejandra Flores Mosri and M. Ghanbari. Prioritised delivery of layered coded video over ip networks. *ACM Transactions on Multimedia*, to be published, 2001.

[MJ96] Steven McCanne and Van Jacobson. Receiver-driven layered multicast. In *Proceedings of the ACM SIGCOMM'96*, Stanford, CA, August 1996. ACM. Multicast.

[NMA00] Kiyohide Nakauchi, Hiroyuki Morikawa, and Tomonori Aoyama. Network support for multicasting streaming media in heterogenous networks. In *Proc. Network Group Communication NGC*, 2000.

[SSZ98] Ion Stoica, Scott Shenker, and Hui Zhang. Core-stateless fair queueing: Achieveing approximately fair bandwidth allocations in high speed networks. In *ACM SIGCOMM. Vancouver, Canada*, 1998.