Investigations Into Efficient Temporal Video Scaling

Michael Kropfberger, Klaus Leopold, Hermann Hellwagner*

April 10, 2002

Abstract

This paper discusses the different types of scaling an MPEG-4 video to saturate a given (or changing) network bandwidth. We differentiate between realtime and non-realtime adaptability. Mainly we will look into temporal scalability, which means simply dropping the least important frames. This can be done on the fly, in a realtime manner.

To do so, we have to look into the patterns generated out of subsequent frames. Those patterns can be modified in different ways and weighted according to multiple heuristics.

We will show an XML scheme to describe an initial state and periodically sent modification information, and we introduce a binary representation for efficient communication on the IP layer.

Finally, ideas on possible optimizations and our developed statistics software is introduced.

Contents

1 Adaptation Methods **2**

1.1 Non-Realtime Scalability 2

1.2 Realtime Scalability 2

2 Pattern Modification Generation **3**

3 Realtime Stream Adaptation **4**

4 Communication Protocol in an Adaptive Environment **6**

5 BSDL (XML) Example for Video Adaptation **7**

5.1 Pre-Sent Video Information 7

5.2 Per-Pattern Information Packets 8

5.3 Modification choice 9

6 Efficient Binary Representation of XML Contents **10**

7 Ideas on Optimization **10**

8 Available Software **11**

9 Conclusion and Future Work **14**

*{mike,klaus, hellwagn}@itec.uni-klu.ac.at

1 Adaptation Methods

MPEG-4[9] supports different profiles[8], how video might be coded and also scaled to different qualities. If we're talking about adaptability and scalability in the context of video transmission over networks we have to differentiate between realtime and non-realtime adaptability and scalability of video streams. Realtime methods can be handled on the router side while forwarding the packets from one network to the other. Non-realtime adaption methods have to be performed at the video server or a video proxy server.

1.1 Non-Realtime Scalability

The amount of non-realtime scalability methods is rather huge. In this paper we'll outline a couple of methods for *video* scalability which are influencing the average bandwidth of a video that later will be transferred over a network.

The most obvious modifications are

- reducing the resolution
- grayscale
- changing the colordepth

If the the end device is a PDA, it makes sense to *resize* the whole video. The maximum resolution of such a device is quite small and the network connection is supposed to be very low too.

Another impact on quality and bandwidth consumption is *grayscaling*. Sometimes the end device doesn't support colors (PDAs) so one can save bandwidth by grayscaling the video and transferring the transcoded video to the end device.

The *colordepth* is another influencing property of a video. Decreasing the amount of bits used per pixel for describing the color value in a certain color space (RGB or YUV), will massively decrease the overall bandwidth requirements.

When we look at the above mentioned methods, we might notice, that everything will diminish the visual quality. It would be also interesting to enhance the visual quality by increasing resolution, adding color depth and so on, but to allow this, we would need the original uncompressed video data in most of the cases. Anyway, this should be kept in mind as a viable solution in *some rare* cases.

1.2 Realtime Scalability

Realtime scalability describes all possibilities for intermediate devices on the way between the server and the client, to react and adapt to the actual network load or other environmental influences, always without delaying the video playback.

- Temporal Scalability
- Spatial Scalability[6]
- SNR Scaling (Signal to Noise Ratio)
- FGS (Fine Grained Scalability)[10][11]

All of the above mentioned methods draw heavy advantage of storing a low-quality video in a base layer and extending the quality by adding an enhancement layer. Those two layers are two separately stored MPEG-4 streams, where only the enhancement layer has dependencies to the base layer.

In the following, we will investigate into efficient temporal video scaling and will ignore all other realtime scalability methods. They will be addressed in further publications in the near future.

2 Pattern Modification Generation

If we are talking about MPEG-4[9] Video elementary streams[7], we mean multiple video frames encoded by different coding types known as I-Frames, P-Frames and B-Frames, where I-Frames are independent from any other frames, P-Frames are based on predictions from the last reference frame and B-Frames are based on predictions of the previous and following reference frame. A reference frame might be either an I-Frame or a P-Frame, so only B-Frames are totally unreferenced by any other frame type.

When we look at a bitstream, we can write out a pattern generated by the subsequent frame types which might look like `IBPBPB`. This Pattern might be repeated over the whole video, or there will be different patterns to immediately react to scene changes. So the first frame after a cut should be an I-Frame, with any combination of following referencing frames.

Temporal Scalability, the simplest way of adaptation, allows us to simply drop frames out of a stream, where the decoder only has to cope with the missing frames, which shouldn't be much of a problem, since every frame has a presentation timestamp stored in its structure. It is the decoders' duty to interpolate over missing frames or at least keep the last shown frame viewed until the next available frame has to be presented.

- B-Frames can be dropped at will since there no other frames referencing them
- When dropping a P-Frame P_p, all previous B-Frames forward-referencing[1] P_p and all following $P_{p'>p}$ and B-Frames have to be dropped too
  eg. `IPBBPBBPBB(P)BBPBBI` $\rightarrow$ `IPBBPBBP(BBPBBPBB)I`
- Dropping I-Frames means losing all following P and B-Frames until the next I-Frame, and again all forward-referencing B-Frames. Since I-Frames are used very seldomly in a frame pattern, dropping I-Frames makes nearly no sense.

To add more flexibility on possible adaptions, MPEG-4 also supports a two-layer approach, where the *base layer* stores eg. `I-B-P-B-P-B-` and the *enhancement layer* stores `-P-B-B-P-B-B`. When those two layers are interleaved, the received pattern would be
`IPBBPBBPPBBB`. Since the *enhancement layer* is totally independent, we can also drop the whole *enhancement layer* (eventually including P-Frames) without interfering with any P-Frames stored in the *base layer*.

Figure 1 shows a possible (non-exhaustive) modification tree on a base layer pattern where some combinations are grayed out, so they won't go into the *pool of possible pattern modifications*. This Pool will be supported by the routers as a valid adaption to a detected bandwidth. Since we know the frame sizes for I, P and B-Frames occurring in this pattern, we can store accumulated pattern bandwidth requirements for each node. This bandwidth requirement is the main selection criteria for *Routers* to choose a specific modification.

Good heuristics for generating a usable pattern tree are based on

- Importance ($I > P > B$)
- Timely balanced distribution
  - Pattern `I-*-P-*-P-` better than `I-B-P-*-*-`

[1] B-Frames have a back- and forward-reference frame, where both might be either an I or P-Frame. So we have to drop all B-Frames after either P_{p-1} or the nearest I-Frame in backward direction.

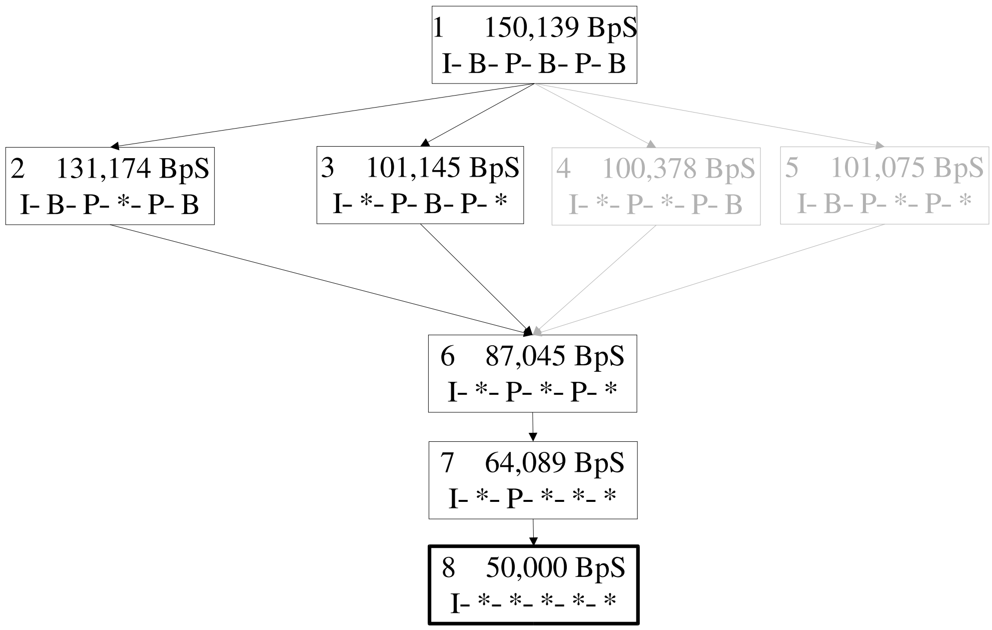

Figure 1: Tree of Possible Base Layer Pattern Modifications

  - skipping the whole enhancement layer might even be better than `P-*-*-*-*-*` (not worth the hassle)

- averaged Signal to Noise Ratio (SNR) for modified pattern

- Tree size vs. fine grained scalability

  - Tree chopping with respect to the above mentioned heuristics like timely balanced distribution

  - delete similarly sized nodes based on threshold values possible decision rules are:

    * preferring higher frame rates
    * preferring higher-quality patterns (`I-*-P-*-P-` better than `I-B-*-B-*-`)

Figure 2 shows possible enhancement layer pattern modifications following the same rules as a base layer tree, except the final node might be an empty one, which means skipping the whole enhancement layer.

Dealing with scalable streams will add further complexity and adaptation restrictions to the above mentioned approach, since the enhancement layer might reference to reference frames in the base layer. Chopping also has to be done with respect to the achievable “Quality” of combinations of base and enhancement layer patterns.

3 Realtime Stream Adaptation

To be efficient in the routers, we have to break down the chopped trees into lists or binary trees sorted by the needed bandwidth per pattern (or Bits per second). These *modification pools* will be transmitted to the subsequent routers, so they can chose the best fitting pattern modification. Video frames are sent via RTP [13] using a standardized packet format for MPEG-4 [12].

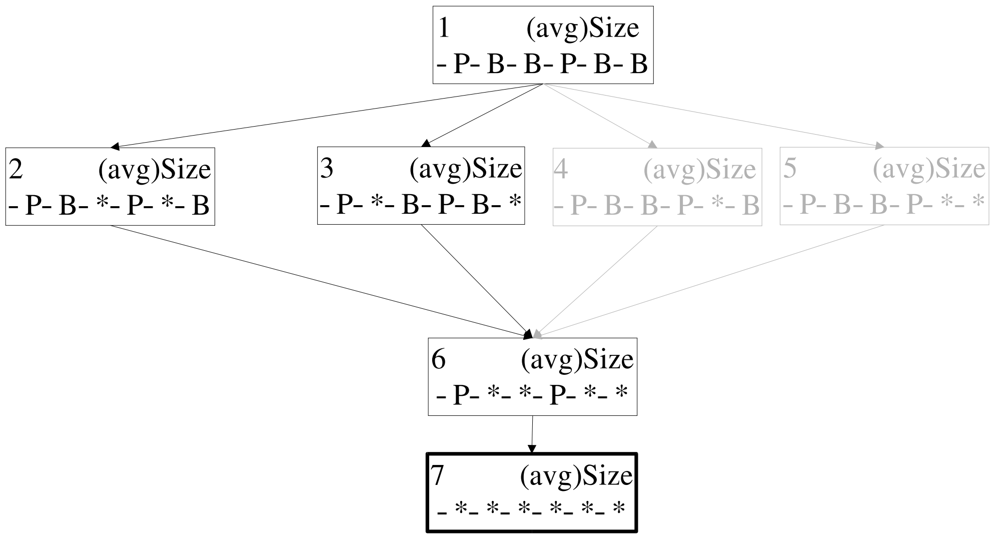

Figure 2: Tree of Possible Enhancement Layer Pattern Modifications

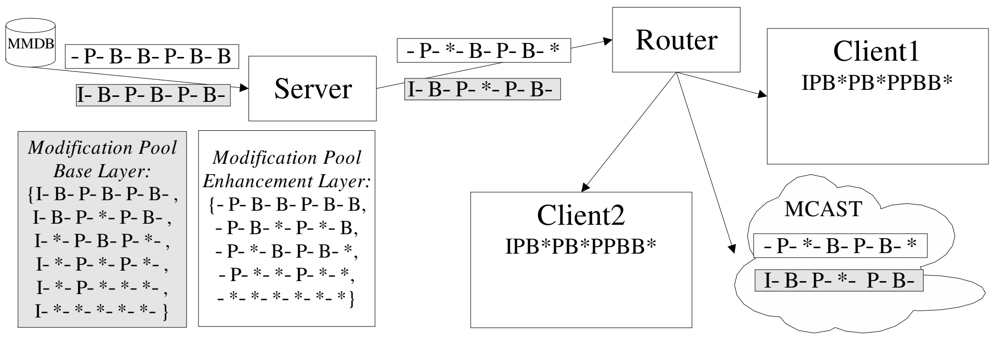

Figure 3: Two clients requesting same quality, base and enhancement layer arrive via multicast

The *RouterSrv* reads the base layer and enhancement layer from disk and only sends the requested frames to the next *Router* hop. This next *Router* requests the maximum frame pattern of all connected clients (or following routers). In Figure 3, both clients want the same quality, so the adapted base and enhancement layer is sent via multicast.

To keep network traffic low, we share as much information possible in the multicast layers. As shown in Figure 4, the *Router* generates a shared base and enhancement layer and extracts two new enhancement layers and one base layer adjusted for the two clients' individual needs. Since standardized players are only capable of one base layer and one enhancement layer, we have to add an intelligent client side "network funnel" that puts together multicasted and unicasted frames into a correct base and enhancement layer format, which is feeded to the real player. We will need this extra intelligence on client side anyways, since we want to talk with our *Router* using or sophisticated protocol.

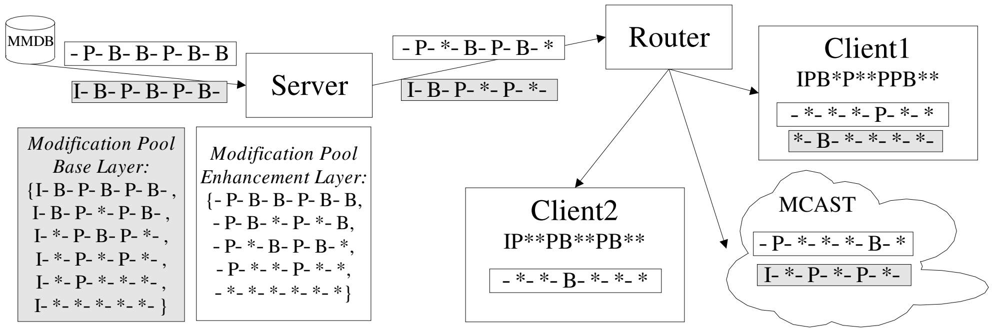

Figure 4: Newly generated shared layers arrive via multicast, unique parts are sent directly

4 Communication Protocol in an Adaptive Environment

1. A new *Client* sends initial specification of video name, resolution, environment and more to the *Router*
2. The *Router* requests the initial video information like resolution, colordepth, average bandwidth and average bandwidth reached by maximum adaptivity (see XML in Section 5.1)
3. *RouterSrv* sends first *modification pool* (in any efficient representation format) with bandwidth requirements respective to the different modifications (see XML in Section 5.2)
4. The *Router* checks available bandwidth to *Client(s)*
5. *Router* requests necessary maximum frame pattern to serve all connected *Clients*
6. *RouterSrv* sends out the requested frame pattern and the next *modification pool* (see Figure 5)

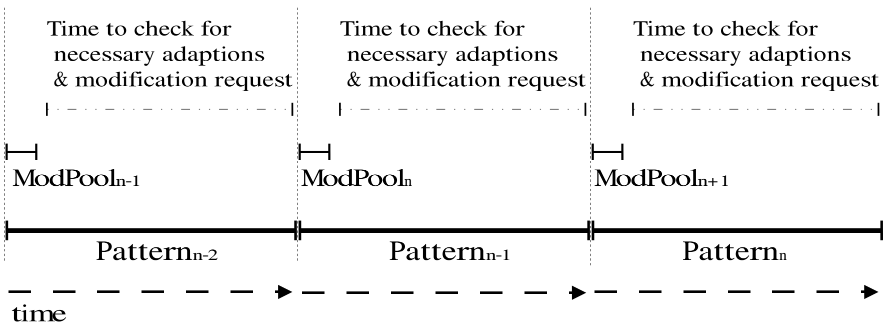

Figure 5: Modification Pool MP_n is sent with Pattern P_{n-1}, so there is enough time to react to changing bandwidth requirements

7. *Router* sends out the shared base and enhancement layer frames via multicast
8. *Router* generates the specific base and enhancement layers and sends them to the connected *Clients* via unicast

9. if no new clients arrive, go back to Step 4

5 BSDL (XML) Example for Video Adaptation

The following XML[4] examples should only show the basic idea of what has to be sent when to the client, ignoring any real BSDL[5] conform style guides. In the near future we will “transcode” our XMLish idea into BSDL, using the correct style sheets and capabilities[14]. For now, we kindly ask the interested reader to accept this as an incomplete substitute.

5.1 Pre-Sent Video Information

This information is sent when a connection is set up. It contains basic information about the quality of the video, the connection properties and possible non-realtime modifications to adapt the video quality to the network's (client's) bandwidth.

The network section of the XML file contains information about the base and enhancement (if any) layers. The RTP ID of the particular stream is sent as well as the bandwidth the stream will consume (in bits per seconds).

The Modifications section of the XML file tells the router the possible adaptions and the benefits. Every modification has a distinct number during the whole session which is represented in the ModID tag. The ScaleType tag reads the type of the scalability. In this paper we concentrate on the type of Temporal scalability. Other ScaleTypes could be Spatial, SNR, Object or Audio for example.

If the router chooses a modification with its *ScaleType* set to `Temporal`, it will receive modification patterns as described in the next subsection.

In the context of adaption on the router side the Feasibility tag indicates whether the router can make the adaption or not - routers can just do RealTime modifications of the stream. There are a lot of non real time modifications that can be handled on a proxy for example.

```
<Init>
  <Resolution>
    <X>352</X>
    <Y>288</Y>
  </Resolution>
  <FrameRate>15</FrameRate>
  <ColorDepth>12</ColorDepth>

  <BaseLayer>
    <RTPid>514</RTPid>
    <BpS>150000</BpS>

    <Modifications>                       <!-- Possible modifications -->
      <Choice>
        <Feasibility>RealTime</Feasibility>
        <ModID>0</ModID>                  <!-- Modification ID -->
        <ScaleType>Temporal</ScaleType>   <!-- Type of modification -->
      </Choice>
      <Choice>
        <Feasibility>NonRealTime</Feasibility>
        <ModID>1</ModID>                  <!-- Modification ID -->
        <ScaleType>GrayScale</ScaleType>  <!-- Type of modification -->
        <BpS>67000</BpS>
```

```
    </Choice>
  </Modifications>
</BaseLayer>

<EnhanceLayer>
  <RTPid>516</RTPid>
  <BpS>200000</BpS>

  <Modifications>                       <!-- Possible modifications -->
    <Choice>
      <Feasibility>RealTime</Feasibility>
      <ModID>0</ModID>                   <!-- Modification ID -->
      <ScaleType>Temporal</ScaleType>   <!-- Type of modification -->
    </Choice>
    <Choice>
      <Feasibility>NonRealTime</Feasibility>
      <ModID>1</ModID>                  <!-- Modification ID -->
      <ScaleType>GrayScale</ScaleType>  <!-- Type of modification -->
      <BpS>113000</BpS>
    </Choice>
  </Modifications>
</EnhanceLayer>
</Init>
```

5.2 Per-Pattern Information Packets

This XML file shows the regularly sent information needed for temporal scalability on a certain RTP connected bitstream. Starting with a RTP sequence number, a certain number of frames (a pattern) will be sent and they may be modified by choosing out of the attached modification pool sorted by bandwidth sizes.

The RTP SeqStart tag lets the router know the first RTP sequence number of the pattern, so the router can easily estimate the correct RTP packets to drop or keep.

In the Modifications section there's another BpS tag which indicates the bandwidth if this modification is chosen. After the BpS tag the scale/adaption commands are listed. The Drop tag indicates that the content of the next frames have to be dropped.

```
<Pattern>
  <BaseLayer>
    <RTPid>514</RTPid>
    <NumFrames>30</NumFrames> <!-- no of frames in the following pattern -->
    <RTPseqStart>3456</RTPseqStart>
    <Modifications>
      <Feasibility>RealTime</Feasibility>
      <Choice>     <!-- No Modification -->
        <ModID>0</ModID>
        <ScaleType>None</ScaleType>
        <BpS>150000</BpS>
        <TotalSize>300000</TotalSize>
      </Choice>
      <Choice>     <!-- drop every 8th frame -->
        <ModID>1</ModID>
        <ScaleType>Temporal</ScaleType>
```

```
        <BpS>100000</BpS>
        <TotalSize>200000</TotalSize>
        <Drop>
          <Frame>8</Frame>
          <Frame>18</Frame>
          <Frame>28</Frame>
        </Drop>
      </Choice>
      <Choice>     <!-- drop every 5th frame, temporally distributed -->
        <ModID>2</ModID>
        <ScaleType>Temporal</ScaleType>
        <BpS>80000</BpS>
        <TotalSize>160000</TotalSize>
        <Drop>
          <Frame>3</Frame>
          <Frame>8</Frame>
          <Frame>13</Frame>
          <Frame>18</Frame>
          <Frame>23</Frame>
          <Frame>28</Frame>
        </Drop>
      </Choice>
    </Modifications>
  </BaseLayer>

  <EnhanceLayer>
    <RTPid>516</RTPid>
    <Modifications>
     ...
     ...
     ...
    </Modifications>
  </EnhanceLayer>
</Pattern>
```

5.3 Modification choice

If the router decides to adapt a video stream, it has to inform the server about the modification. In order to consume as less bandwidth as possible the server also has to adapt the multicast stream to the router with the least common patterns for all clients (see Section 3).

The choice of a modification is rather simple. The router just sends the RTP ID and the modification ID to the server.

```
<Choose>
  <RTPid>514</RTPid>
  <ModID>5</ModID>
</Choose>
```

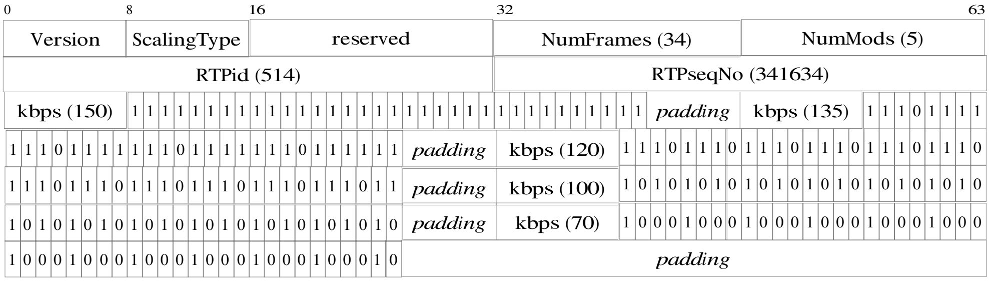

Figure 6: Packet format for per-pattern modification pool

6 Efficient Binary Representation of XML Contents

Since we cannot directly handle XML files on the router side because of performance issues, we have to send a highly efficient binary representation of at least the periodically sent per-pattern modification pool. This low-level packet layout (Figure 6) is preferred to an automatically generated MPEG-21 BiM representation, because with our approach we keep the packet size at a minimum. Other devices might still receive their XML (BSDL) information in a less efficient BiM representation, since the data stored in our packet format and the XML are identical.

7 Ideas on Optimization

Even with the very low network footprint of the regularly sent pattern modification packets, we will add an average network load of approx. 60-100 bytes. In normal cases we can easily ignore this overhead, but on very low bandwidth networks, maybe with large minimal packet sizes or other per-connection overhead, but also to save computational power on the router side, we would like to further reduce the amount of data to send or process.

Given the case that

- a video uses the same pattern repeatedly over the whole lifetime (or at least for a couple of iterations),
- the average needed bandwidth per pattern is very similar,
- and the average framesizes between and within the patterns are similar,

we could merge multiple patterns into a *pattern group*. Each *pattern group* will hereby represent eg. the next fifty patterns and we calculate the pattern modification tree based on this averaged *pattern group*.

After building a list of very few *pattern groups* reflecting the whole video, we can send these few pattern modification lists within the initially transmitted video information XML file. No further updates are necessary, and the router knows exactly when to install the next modification list.

Unfortunately our measurements on different videos with multiple qualities, patterns and MPEG-4 encoders[3][2][1] revealed, that we cannot rely on the above mentioned assumptions at all.

There are too high variations already on a frame-to-frame basis, which make a *pattern group*, given a reasonable threshold, too small and patterns which seemed to belong to one *pattern group* by having the same average bandwidth, would differ from each other after applying modifications, which means dropping frames on certain positions in the pattern (see Figure 7).

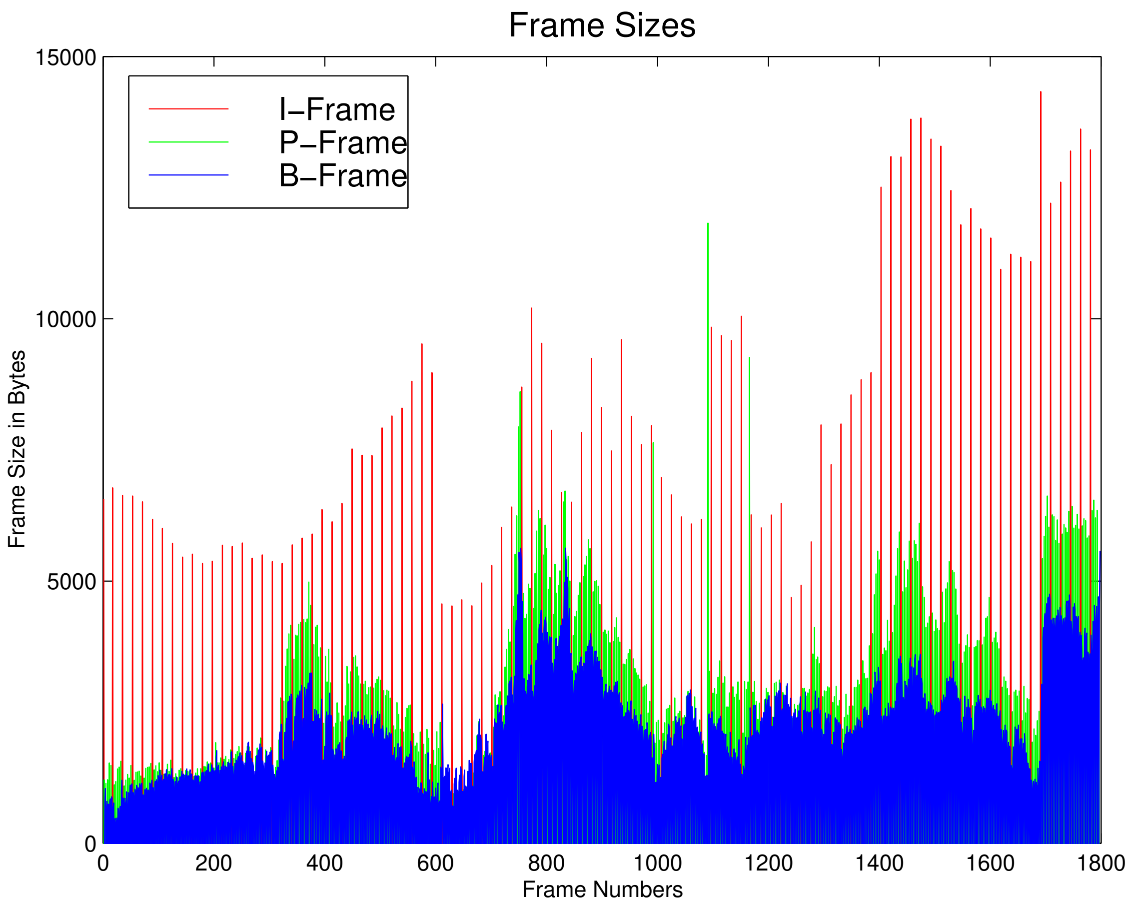

Figure 7: Frame Size Variations

The only possibility to reach the requirements would be a fixed bitrate encoding, which is not sufficiently supported by any of the MPEG-4 ISO encoders yet and would imply massive quality loss or bandwidth overhead.

8 Available Software

We wrote a statistic suite including frame types and sizes, average deviations with threshold triggers. We can detect patterns over a video, again with average bandwidths and exact frame sizes, threshold triggers to indicate deviation to the prior patterns. We cumulate patterns into pattern groups and we generate averaged frame sizes and bandwidth requirements.

```
Total input bytes = 5196951
Number of VOPs = 1798


Frame    I-Frame(avg|deviation)  P-Frame(avg|deviation)  B-Frame(avg|deviation)
  0       6567
```

```
  1                              1211
  2                                                         780
  3                                                      *** HIGH DEVIATION ***
                                                         1052( 1052|  136|15%)
  4                              1142( 1176|  -34| 3%)
  5                                                      *** HIGH DEVIATION ***
                                                          678(  678| -187|22%)
  6                                                       801(  739|   62| 8%)
  7                              1215( 1195|   20| 2%)
  8                                                       751(  745|    6| 1%)
  9                                                       737(  741|   -4| 1%)
 10                             *** HIGH DEVIATION ***
                                 1535( 1535|  170|12%)
 11                                                       807(  774|   33| 4%)
 12                                                       849(  811|   38| 5%)
 13                              1485( 1510|  -25| 2%)
 14                                                       887(  849|   38| 4%)
 15                                                       863(  856|    7| 1%)
 16      6779( 6673|  106| 2%)
 17                                                      *** HIGH DEVIATION ***
                                                          687(  687|  -84|11%)
 18                                                       769(  728|   41| 6%)
 19                             *** HIGH DEVIATION ***
                                 1120( 1120| -195|15%)
 20                                                      *** HIGH DEVIATION ***
                                                          430(  430| -149|26%)
 21                                                       475(  452|   23| 5%)
 22                              1180( 1150|   30| 3%)
 23                                                       404(  428|  -24| 6%)
 24                                                       483(  455|   28| 6%)
 25                             *** HIGH DEVIATION ***
                                 1419( 1419|  135|11%)
 26                                                      *** HIGH DEVIATION ***
                                                          668(  668|  107|19%)
 27                                                       645(  656|  -11| 2%)
 28                              1455( 1437|   18| 1%)
 29                                                       574(  615|  -41| 7%)
 30                                                       704(  659|   45| 7%)
 31                              1566( 1501|   65| 4%)
 32                                                      *** HIGH DEVIATION ***
                                                          994(  994|  168|20%)
 33                                                      1205( 1099|  106|10%)
.
.
.
.
.
1780    13221(13282|  -61| 0%)
1781                                                     3969( 3781|  188| 5%)
1782                                                     3663( 3722|  -59| 2%)
1783                             6346( 6134|  212| 3%)
1784                                                     3836( 3779|   57| 2%)
1785                                                     4234( 4006|  228| 6%)
1786                             6543( 6338|  205| 3%)
```

```
1787                                                       4534( 4270|  264| 6%)
1788                                                       4280( 4275|    5| 0%)
1789                         6207( 6272|  -65| 1%)
1790                                                       4214( 4244|  -30| 1%)
1791                                                       4517( 4380|  137| 3%)
1792                         6348( 6310|   38| 1%)
1793                                                       4259( 4319|  -60| 1%)
1794                                                       4701( 4510|  191| 4%)
1795                        *** HIGH DEVIATION ***
                             4621( 4621| -844|15%)
1796                                                       4520( 4515|    5| 0%)
1797                                                      *** HIGH DEVIATION ***
                                                           5575( 5575|  530|11%)
                        TNo     TSize   TAvg
I-Frames                100     815961  8159
P-Frames                500     1611609 3223
B-Frames                1198    2769348 2311

  1(16frm): tsize:   21360 ( 0%) avg 1335: IPBBPBBPBBPBBPBB
###### ***END OF PATTERN GROUP (1 pats)*** avgTsize 21360 avgFrmSize  1335

  2(18frm): tsize:   21557 ( 0%) avg 1197: IBBPBBPBBPBBPBBPBB
  3(18frm): tsize:   24178 (-12%) avg 1343: IBBPBBPBBPBBPBBPBB
  4(18frm): tsize:   25941 (-20%) avg 1441: IBBPBBPBBPBBPBBPBB
  5(18frm): tsize:   26183 (-21%) avg 1454: IBBPBBPBBPBBPBBPBB
  6(18frm): tsize:   27913 (-29%) avg 1550: IBBPBBPBBPBBPBBPBB
###### ***END OF PATTERN GROUP (5 pats)*** avgTsize 25154 avgFrmSize  1397

  7(18frm): tsize:   28759 ( 0%) avg 1597: IBBPBBPBBPBBPBBPBB
  8(18frm): tsize:   26972 ( 6%) avg 1498: IBBPBBPBBPBBPBBPBB
  9(18frm): tsize:   27984 ( 3%) avg 1554: IBBPBBPBBPBBPBBPBB
 10(18frm): tsize:   27179 ( 5%) avg 1509: IBBPBBPBBPBBPBBPBB
 11(18frm): tsize:   29253 (-2%) avg 1625: IBBPBBPBBPBBPBBPBB
 12(18frm): tsize:   30943 (-8%) avg 1719: IBBPBBPBBPBBPBBPBB
 13(18frm): tsize:   32198 (-12%) avg 1788: IBBPBBPBBPBBPBBPBB
 14(18frm): tsize:   33079 (-15%) avg 1837: IBBPBBPBBPBBPBBPBB
 15(18frm): tsize:   34221 (-19%) avg 1901: IBBPBBPBBPBBPBBPBB
 16(18frm): tsize:   33450 (-16%) avg 1858: IBBPBBPBBPBBPBBPBB
 17(18frm): tsize:   33496 (-16%) avg 1860: IBBPBBPBBPBBPBBPBB
 18(18frm): tsize:   34054 (-18%) avg 1891: IBBPBBPBBPBBPBBPBB
###### ***END OF PATTERN GROUP (12 pats)*** avgTsize 30965 avgFrmSize  1720
.
.
.
.
.
```

Finally we support tools to generate an index for a MPEG-4 bitstream with frame type, size and state (drop vs. keep). We can feed the bitstream with this description index to generate bitstreams with the adapted video and the dropped frames. For control issues, we have a tool to merge those two files back into the original video.

9 Conclusion and Future Work

In this paper we outlined different types of scaling an MPEG-4 video to adapt to network bandwith changes. We decribed the internal representation of an MPEG-4 video bitstream and introduced the idea of pattern generation and modification of frame groups to do realtime adaptation for a video stream on a router. We outlined a communication protocol in an adaptive environment based on XML and described an efficient representation of the XML content so it can be handled by a router. Furthermore we presented ideas on optimization for the pattern generation and modification which can't be realized without restrictions.

Presently we are working on intelligent algorithms to generate patterns to massively improve the quality of the video on clients' side despite dropping frames on a router. Further more we are trying to deploy other methods for video scaling on a router separate from temporal scaling.

References

[1] Mpeg4ip. ISO/IEC 14496 Microsoft MPEG-4 Video Reference Software. http://mpeg4ip.sourceforge.net/.

[2] Opendivx. http://www.projectmayo.com/.

[3] MoMuSys: Mobile Multimedia Systems. ISO/IEC 14496 MPEG-4 Video Reference Software, ACTS-AC098, 1995-1999. Partners: Bosch, Siemens, University of Hamburg and Madrid, Deutsche Telekom, Heinrich Hertz Institut, and more.

[4] Extensible Markup Language (XML) 1.0 (Second Edition). *W3C Recommendation*, October 6th 2000. http://www.w3.org/TR/2000/REC-xml-20001006.

[5] Sylvain Devillers. Bitstream Syntax Definition Language (BSDL): An Input to MPEG-21 Content Representation. *ISO/IEC JTC1/SC29/WG11 M7053*, March 2001.

[6] Rakesh Dugad and Narendra Ahuja. A Scheme for Spatial Scalability Using Nonscalable Encoders. *submitted for publication in IEEE Trans. Circuits and Systems for Video Technology*, September 2001. http://vision.ai.uiuc.edu/dugad/.

[7] Carsten Herpel and Alexandros Eleftheriadis. MPEG-4 Systems: Elementary Stream Management. *Image Communication Journal. Tutorial Issue on the MPEG-4 Standard*, 15(1-2), January 2000. http://leonardo.telecomitalialab.com/icjfiles/mpeg-4_si/.

[8] Rob Koenen. Profiles and Levels in MPEG-4: Approach and Overview. *Image Communication Journal. Tutorial Issue on the MPEG-4 Standard*, 15(1-2), January 2000. http://leonardo.telecomitalialab.com/icjfiles/mpeg-4_si/.

[9] Rob Koenen. Overview of the MPEG-4 Standard. *ISO/IEC JTC1/SC29/WG11 N4030*, March 2001. http://mpeg.telecomitalialab.com/.

[10] Weiping Li. Overview of Fine Granularity Scalability in MPEG-4 Video Standard. *IEEE Trans. Circuits and Systems for Video Technology*, 11(3), March 2001.

[11] Matthias Ohlenroth and Hermann Hellwagner. Quality Adaptation Options of MPEG-4 Video Streams. Technical Report TR/ITEC/01/1.03, University Klagenfurt, December 2001.

[12] Matthias Ohlenroth and Hermann Hellwagner. RTP Packetization of MPEG-4 Elementary Streams. Technical Report TR/ITEC/02/1.01, University Klagenfurt, February 2002.

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

[14] Anthony Vetro. MPEG-21 Requirements on Digital Item Adaptation. *ISO/IEC JTC1/SC29/WG11 N4515*, December 2001.