

© IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.

This material is presented to ensure timely dissemination of scholarly and technical work. Copyright and all rights therein are retained by authors or by other copyright holders. All persons copying this information are expected to adhere to the terms and constraints invoked by each author's copyright. In most cases, these works may not be reposted without the explicit permission of the copyright holder.

# Cost of In-Network Adaption of MC-EZBC for Universal Multimedia Access

Heinz Hofbauer  
Salzburg University  
Department of Computer Sciences  
hhofbaue@cosy.sbg.ac.at

Andreas Uhl  
Salzburg University  
Department of Computer Sciences  
uhl@cosy.sbg.ac.at

## Abstract

A core concept of universal multimedia access is the use of scalable content. The scalable content should be as flexible as possible to achieve the credo of creating the content once and adapting it to fulfill given requirements. As far as flexibility goes wavelet based codecs are superior. A problem which arises is that the adaptation does not necessarily happen on a device which is aware of the codec which was used in content creation. To rectify this non-awareness, MPEG-21 introduced digital item adaptation in part 7 to abstract the bitstream of a given video. This allows in-network adaptation on nodes which are DIA aware to adapt any video stream as long as a DIA description is given. The drawback is that the DIA description must be sent parallel to the original video sequence. In this paper we will look at how a DIA description for a t+2D scalable wavelet codec looks like. We will evaluate the possibilities we have with various description options and we will also look at the overhead generated by the DIA description.

## 1 Introduction

The use of digital video in today's world is ubiquitous. Videos are viewed on a wide range of clients, ranging from hand held devices with QVGA resolution (320x240) over PAL (768x576) or NTSC (720x480) to HD 1080p (1920x1080) or higher. Furthermore, streaming servers should be able to broadcast over the internet with regard to a wide range of bandwidths, from fixed high bandwidth lines like ADSL2 to changing low bandwidths for mobile wireless devices. In such an environment it is simply not possible to encode a video for every application scenario. So content providers either have only a fixed number of options available or they use scaling video technology to adapt the video for bandwidth and resolution requirements of the client. The concept of creating the content once and adapting it to the current requirements is preferable and is better known as Universal Multimedia Access (UMA) [10].

One of the enabling technologies of UMA is the use of scalable video coding. This averts the need for transcoding on the server side and enables the server to scale the video. However, even scaling takes up computation time and reduces the number of connections the server can accept. Fur-

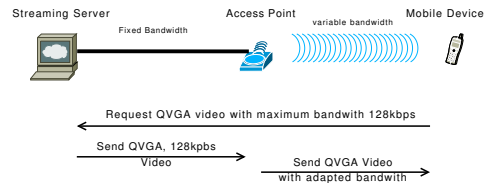


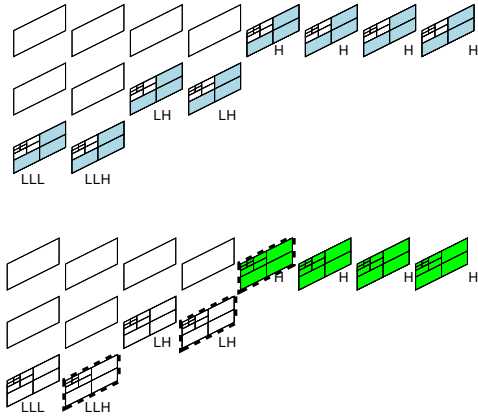
Figure 1. Example of video adaptation for a mobile device on the server and in the network.

thermore, variable bandwidth conditions, which happen frequently on mobile devices, further taxes the server with the need to adapt the video stream. The solution to this is usually in-network adaptation, shifting the need to scale to the node in the network where a change in bandwidth is occurring. Figure 1 shows an example of this scenario, where a mobile device requests a video stream from the server which fits its capabilities. The core adaptation with these restrictions takes place on the server and on the fly adaptation due to actual channel capability is done in-network. Wu et al. [11] give an overview of other aspects of streaming video ranging from server requirements to protocols, to QoS etc.

For video streaming in this environment, i.e. a high number of possible bandwidths and target resolutions, wavelet based codecs can be considered. Wavelet based codes are naturally highly scalable and rate adaptation as well as resolution or temporal scaling is easily achieved. Furthermore, wavelet based codecs achieve a coding performance similar to H.264/SVC, c.f. Lima et al. [7].

For this reason we will consider the ENH-MC-EZBC wavelet based video codec for in-network adaptation. This choice was made mainly because the source code is available <sup>1</sup>, which enables our experiments. The MC-EZBC codec [4, 12] is a scalable t-2D video codec which uses motion compensated temporal filtering, with 5/3 CDF wavelets, followed by regular spatial filtering, with 9/7 CDF filtering, see fig. 2 for a GOP size of 8. This method, temporal first and spatial later, is referred to as t+2D coding scheme. For temporal filtering a full decomposition is used and thus the GOP size is discernible by the number of tem-

<sup>1</sup>The source for the ENH-MC-EZBC is available from <http://www.cipr.rpi.edu/research/mcezbc/>.



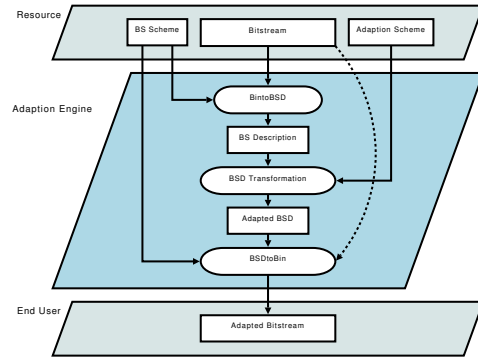
**Figure 2. Overview of the decomposition of a GOP with GOP size 8 with marked (gray parts) high temporal layer (bottom), high spatial layer (top) and possible I-frames as dashed outline on the lower half.**

poral decomposition levels. Both temporal and spatial filtering is done in a regular pyramidal fashion. Statistical dependencies are exploited by using a bit plane encoder, the name giving embedded zero bit coder. Motion vectors are encoded with DPCM followed by an arithmetic coding scheme.

For an overview about wavelet based video codecs and a performance analysis as well as techniques used in those codecs see the overview paper by Adami et al. [1].

The scalability of the video codec is important for UMA which means it is also necessary for servers and network nodes to be able to perform scaling. This can either be achieved by making them aware of the video codec, which would make upgrading to a different codec later quite troublesome, or by abstracting the actual bitstream. MPEG-21 gives a specification how such an abstraction has to look like. Part 7 of MPEG-21 [5] deals with Digital Item Adaption (DIA), more precisely it specifies a Bitstream Syntax Description Language (BSDL) which is based on the XML schema as specified by the W3C. The idea behind DIA with BSDL is that a syntax description of the bitstream is available and can be used to extract a XML description of the bitstream. On this abstraction of the bitstream the scaling is performed and mapped back to the original bitstream via the BSDL, see fig. 3 for an illustration of the process. As a result each node in the network only needs to be capable of understanding and handling DIA as per MPEG-21 part 7. For bitstreams which do not follow a marker based syntax, specifically if parsing the bitstream would be required to generate a description, the approach using BSDL does not work. For this cases a generic bitstream syntax description (gBSD) is available in MPEG-21 part 7 (see Panis et al. [8]) which can be used to directly describe the bitstream.

Usually when research is done on in-network adaptation the focus is on client and server layout as well as computational demand on the network node which performs scal-



**Figure 3. Overview of the adaptation process using the bitstream syntax description language.**

ing. That is, memory consumption on the network node and the scaling options as well as resulting video quality under those scaling options are evaluated, e.g. Eeckhaut et al. [3]. What is missing, especially when considering gBSD rather than BSDL, is that the transfer of the bitstream description also takes up bandwidth. Essentially for a fixed bandwidth the use of a bitstream description reduces the available bandwidth for the actual bitstream which in turn results in a reduction of video quality. For similar research regarding h.264/SVC see Kuschnig et. al. [6].

In section 2 we will describe the MC-EZBC bitstream in such detail as is necessary to map it to gBSD and provide possible gBSD descriptions of the bitstream. Section 3 will look at the overhead generated by gBSD and section 4 will give a conclusion and outlook.

## 2 Mapping an MC-EZBC Bitstream to gBSD

In the following the bitstream of the MC-EZBC will be described with regard to the gBSD mapping. Then we will give a brief description of the gBSD elements we use to map the bitstream to gBSD and give two possible mappings. The scaling option we want to maintain for in-network adaptation reflects which information we will need in the gBSD. We will focus on the two reasonable end points of the spectrum, i.e. full scalability in order to retain the advantage the wavelet based codec has vs. regular rate-distortion scaling with a limited number of scaling points reflecting the application scenario given in fig. 1.

### 2.1 MC-EZBC Bitstream

The basic layout of the MC-EZBC bitstream is depicted in the upper part of fig. 4 and a more detailed overview of the 'image data' required for fine grain scalability is given in lower part. The bitstream is lead by a general header giving resolution, frame rate, prediction options etc., most of which stay the same during scaling. The header however has three fields we need to adjust when scaling is performed: a `bitrate` field giving the bit rate to which the bitstream is scaled, `t_level` giving the number of temporal layers dropped and `s_level` giving the number of

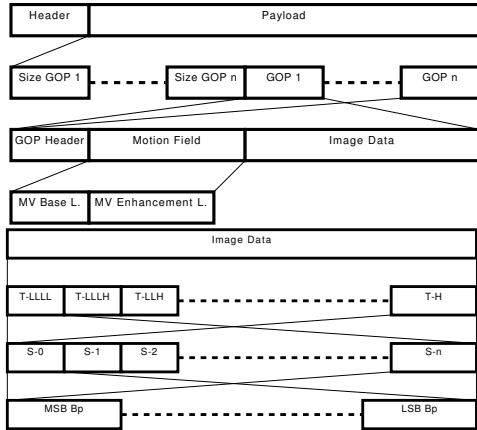


Figure 4. Layout of the MC-EZBC bitstream.

spatial layers dropped. The header is followed by a GOP size list giving the size of a GOP without GOP header size and motion field, i.e. only image data size. For any scaling done the GOP size list has to be adjusted to reflect the new size of image data.

Following this general information are the motion and image data ordered by GOPs in increasing order. Each GOP contains a GOP header, basically giving scene change information, i.e. which frames are encoded as I frames. Following the GOP header is the motion field for the current GOP. The GOP header and motion field are never changed during scaling, i.e. motion vectors are never scaled with the image data. Following the motion field is the image data in frame order of temporal decomposition, c.f. fig. 2 and fig. 4 lower part.

The layout of the image data consists of a number of data chunks consisting of size information and data. For each frame every spatial decomposition level is given as one chunk where color information and direction of decomposition are grouped together, fig. 5 illustrates this. The order of this chunks in the bitstream is from lowest subband to highest subband. For scaling, the size information of the chunks needs to be reset to the reduced data in the chunk, thus a description of the bitstream has to be at least down to the level of chunks. For a limited number of scaling options this would be enough since the chunk data can be subdivided into blocks which we can remove. However, if we want to retain the full scalability capability of the wavelet bitstream we have to go into more detail. In each chunk there is a three byte header which may never be removed for regular scaling, however when the whole resolution is dropped these three bytes can be dropped too. Then the data is ordered in terms of bitplanes, most significant to least significant. The reason we need the bitplane information is that the scaling algorithm performs quantization at a bitplane level, so for an implementation of the scaling algorithm the size information of the bitplanes is paramount.

## 2.2 gBSD Mapping

We use the gBSD from MPEG-21 DIA for describing the bitstream. While the gBSD allows more structural informa-

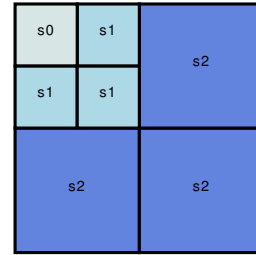


Figure 5. Grouping of decompositions for a frame with two spatial decomposition levels.

tion to go into the description we will keep the bitstream description simple so as not to generate too much overhead.

The gBSD is prefaced with a `dia:DIA` root tag specifying namespaces followed by a `dia:Description` tag specifying the description type (`gBSDType`) followed by address information. Since the MC-EZBC bitstream is byte based we set it to `addressUnit="byte"` and `addressMode="Absolute"`. The address mode gives the method of accessing parts of the bitstream, this is reflected by the use of start and length attributes in subsequent tags. For the bitstream description we need two different types of tags. First we need a copy and paste descriptor stating that a part of the original bitstream should be carried over to the scaled version. The `gBSDUnit` tag is used for this purpose, we give start and length information to mark a part of the bitstream to be kept. Additionally we need access to the size information, in a scaling case this is not simply a copy from the original bitstream but needs to be adapted. The `Parameter` tag is used for this, which gives the length of the data block to insert into the bitstream. The actual information contained in the parameter is given by the required child `Value`. The attribute `xsi:type` gives the type of data and the content of the tag gives the actual value. By using parameter and value we can access the actual value and change it according to the adaptation, while the `gBSDUnit` tags let us copy parts of the actual bitstream. Both parameter and `gBSDUnit` also have an attribute `marker` which allows to give a handle to the tag to access it directly. For more information on the tags and attributes used see MPEG-21 part 7 [5].

First we will look at the description of the bitstream for a two case scenario to get lower bound for the limited case scenario. The temporal and spatial resolutions stay fixed and we give the option of scaling to 1024kbps and 512kbps. We only need to describe the bitstream down to the level of chunks of image data. Also, since we do not want resolution dropping the header description is simplified since only the bitrate field need to be changed. The GOP size list needs to be described by the parameter tag as it will change when scaling is done. Motion vectors and GOP headers however can be put together as one `gBSDUnit` since they are consecutive and will remain unchanged. Following the GOP headers are the chunks of image data, the header is described by a parameter tag and the data is described by using two `gBSDUnits` reflecting the two scaling options we want. One

```

...
<dia:Description xsi:type="gBSDType"
addressUnit="byte" addressMode="Absolute">
  <gBSDUnit start="0" length="14" marker="hdr1"/>
  <Parameter length="2" marker="bitrate Q0">
    <Value xsi:type="xsd:unsignedShort">1024</Value>
  </Parameter>
</gBSDUnit start="16" length="80" marker="hdr2"/>
...
  <Parameter length="2" marker="hdr Q0">
    <Value xsi:type="xsd:unsignedShort">118</Value>
  </Parameter>
  <gBSDUnit start="545775" length="18" marker="data"/>
  <gBSDUnit start="545793" length="100" marker="data Q0"/>
  <Parameter length="2" marker="hdr Q0">
    <Value xsi:type="xsd:unsignedShort">185</Value>
  </Parameter>
  <gBSDUnit start="545895" length="21" marker="data"/>
  <gBSDUnit start="545916" length="164" marker="data Q0"/>
</dia:Description>
</dia:DIA>

```

**Figure 6. gBSD representation of the flower sequences scaled to 1024 kbps with marker for 512kbps.**

```

...
<dia:Description xsi:type="gBSDType"
addressUnit="byte" addressMode="Absolute">
  <gBSDUnit start="0" length="14" marker="hdr1"/>
  <Parameter length="2" marker="bitrate Q1">
    <Value xsi:type="xsd:unsignedShort">512</Value>
  </Parameter>
</gBSDUnit start="16" length="80" marker="hdr2"/>
...
  <Parameter length="2" marker="hdr Q1">
    <Value xsi:type="xsd:unsignedShort">18</Value>
  </Parameter>
  <gBSDUnit start="545775" length="18" marker="data"/>
  <Parameter length="2" marker="hdr Q1">
    <Value xsi:type="xsd:unsignedShort">21</Value>
  </Parameter>
  <gBSDUnit start="545895" length="21" marker="data"/>
</dia:Description>
</dia:DIA>

```

**Figure 7. gBSD representation of the flower sequence, downscaled to 512kbps.**

gBSDUnit locates the data we need for the 512kbps version of the bitstream, the next describes the additional data for the 1024kbps case. The rest of the data, in case we start with a bitstream with bitrate greater than 1024kbps, does not need to be described since it will be cut out in case of scaling anyway. Figure 6 gives a part of the description of the bitstream which can be used to scale to 1024kbps. It also shows the description of the header where it can be seen that only the bitrate has to be described as parameter and that it needs to be set to 1024 to properly reflect the bitrate of the stream. The resulting description of the stream still consists of two gBSDUnit descriptions discerning between 512 and 1024 kbps. Compare this to fig. 7 which describes the bitstream for the 512kbps scaling case. The shown part of the description refers to the same section of the bitstream as the 1024kbps case. It is clear that the scaling of the bitstream also entails a scaling of the gBSD description. But, it also is clear that adding another scaling option for this fixed case requires the insertion of another gBSDUnit partition for each chunk.

The second scenario we want to look at is full grained scalability. For this case we have to render a finer description of the bitstream down to bitplane level. The overhead in the header is rather small, we just need to add the resolution drop fields as parameters. For the GOP size list, GOP headers and motion vectors nothing changes compared to the two case scenario. For the description of the image data chunks however we need a lot more detail and con-

```

...
  <gBSDUnit start="272992" length="1" marker="data sp 59"/>
  <gBSDUnit start="272993" length="2" marker="data sp 58"/>
  <gBSDUnit start="272995" length="1" marker="data sp 57"/>
  <Parameter length="2" marker="hdr">
    <Value xsi:type="xsd:unsignedShort">19</Value>
  </Parameter>
  <gBSDUnit start="272998" length="6" marker="data sp 79"/>
  <gBSDUnit start="273004" length="3" marker="data sp 71"/>
  <gBSDUnit start="273007" length="1" marker="data sp 67"/>
  <gBSDUnit start="273008" length="3" marker="data sp 63"/>
  <gBSDUnit start="273011" length="5" marker="data sp 62"/>
  <gBSDUnit start="273016" length="1" marker="data sp 59"/>
...

```

**Figure 8. Detailed gBSD representation of the flower sequence, downscaled to 512kbps.**

sequently a lot more gBSDUnit tags. First we need the sub-band header, which will not be changed in any case. This can either be described as an extra tag or can be contained in the first bitplane following the size information. We choose the latter version since the scaling algorithm must be aware of the header anyway when calculating the overhead. For the rest of the image data we have to model each bitplane as a separate gBSDUnit since the size of the bitplanes is required by the scaling algorithm. Despite the possibility that a bitplane is reduced in size we still can describe them with a gBSDUnit because the actual content does not change and a reduction in size can be achieved by resetting the length attribute. Figure 8 shows a part of the gBSD description for a downscaled version to 512kbps, the bitplane quantization can be clearly seen, i.e. the lowest bitplane in the figure is the 57th.

In our examples the description is kept as simple as possible so as not to use up too much bandwidth. However, for an actual application it would be beneficial to retain some structure by nesting gBSDUnits. While this increases the size of the gBSD description it makes XSLT writing much easier and helps to avoid errors.

### 3 Evaluation of gBSD Overhead

In the two case scenario we can give a good approximation of the overhead. The framerate  $f$  in the video sequence and the encoded sequence stay the same but the number of GOPs is dependant on the temporal decompositions  $t$ . The number of spatial decompositions  $s$  depends on the resolution of the original and can change from sequence to sequence. We also have an approximate number of bytes each descriptive element of the gBSD requires. The number of bytes a Parameter  $p$  and gBSDUnit  $g$  require are 105 and 55 bytes respectively. This numbers are calculated with average variable length information (i.e. length value, start value), additionally we have a overhead for the DIA declaration which is 393 bytes. This means that the start and length fields as well as the value of parameters are only estimated since this information can vary widely. However, the use of a typical marker element is included since the marker will be a near constant in length. We can now calculate an approximate size of the gBSD. The main header consists of one changeable field with size  $p$  and two gBSDUnits of size  $g$  which stay constant. With a temporal resolution of  $t$  we have a GOP size of  $2^t$  and the number of

Scenario	kbps	size	compressed
Bitstream	full	5.5M	
Bitstream	1024	540k	
Bitstream	512	272k	
Detailed gBSD	full	1.2M	112k
Detailed gBSD	1024	400k	32k
Detailed gBSD	512	268k	20k
Two case gBSD	full	84k	8k
Two case gBSD	1024	84k	8k
Two case gBSD	512	64k	4k

**Table 1. Comparison of bitstream and gBSD file sizes for the flower sequence with 128 frames, GOP size of 128 and two spatial decompositions.**

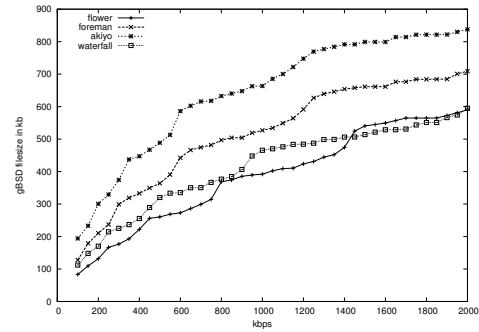
GOPs is  $G = f/2^t$ , for each GOP the header is followed by a GOP size entry as a parameter  $p$ . For each GOP we have a single gBSDUnit for the GOP header and motion vectors. Then for each frame we have a single chunk for each spatial decomposition level  $(s + 1)g$ , i.e. if we do two spatial decompositions we have three subbands, see fig. 5. The chunks here have to be separated into the number of cases  $C$  we want to deal with. The resulting approximation in byte is thus size  $S$ :

$$S = 393 + \underbrace{p + 2g}_{\text{header}} + \underbrace{G * p}_{\text{GOP size list}} + \underbrace{G(g + 2^t(s + 1)(p + Cg))}_{\text{single GOP}}$$

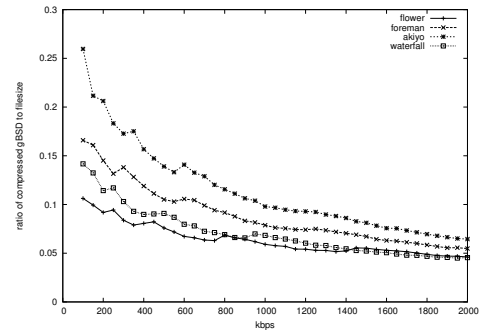
For a sequence with 128 frames,  $t = 7$  and  $s = 2$  this would estimate a gBSD file size of 81kb for the two case gBSD and 60kb for the downscaled version. This is compared experimentally to the actual file sizes of the flower sequence in table 1. The table gives the file size of the bitstream under the bitstream 'scenario', it also gives the two case gBSD file sizes scaled and unscaled. Note that the description for full is simply the complete gBSD containing both scenarios. When scaling to 1024kbps we still use the full description and for 512kbps the gBSD is reduced by the gBSDUnit sizes describing the 1024kbps parts of the bitstream. As can be seen the approximation given reflects the actual gBSD size quite accurately. Note that other sequences have a similar size for the limited case scenario, differing by less than 2%.

The transmission of the gBSD description will usually not be in plain text. XML which is the basis of gBSD can be compressed quite well, see Augeri et al. [2]. We used bzip2 to generate the compressed file sizes as given in table 1. While this may not be the best way to compress the data with regard to network nodes, where a XML aware compression scheme would be beneficial to save memory and time (see Timmerer et al. [9]), we will still use it as a baseline as it offers better compression.

For the detailed description case it is not possible to give a formula since the description, especially when scaling is performed, is heavily dependant on the layout of the bit-



**Figure 9. File size of the detailed gBSD description depending on the bitrate for a number of sequences, each with 128 frames and one GOP.**



**Figure 10. Ratio of compressed gBSD descriptions to bitstream size plotted over kbps for various sequences.**

planes. The number and size of the bitplanes however vary widely depending on the content of the sequence. During scaling it is possible that a number of bitplanes with a low amount of data are removed which reduces the gBSD file size drastically. For a different sequence the same scaling operation could only reduce the number of bytes in the affected bitplanes which would leave the gBSD size nearly unchanged. Figure 9 gives a plot of gBSD file size over kbps for a number of sequences. What is interesting here is that the gBSD file size is higher for low motion videos like Akiyo. This is due to the fact that sequences like Akiyo can be predicted very well which results in a low file size. Consequently, when scaling to a certain bitrate is performed, fewer bitplanes have to be dropped and likewise fewer bitplane descriptions are deleted. This results in a larger gBSD file when compared to high motion sequences.

Figure 10 shows an overview over the ratio of compressed gBSD descriptions to bitstream size. What can be seen is that the ratio gets worse as the bitstream size is reduced. While the description of the bitplanes scales along with the bitstream there is also a fixed amount of data consisting of DIA overhead and bitstream headers. The effect of low reduction in bitplane description and fixed overhead is especially detrimental. This effect is especially observed for low motion sequences like Akiyo where the *compressed* gBSD size can reach up to 25% of the bitstream size.

## 4 Conclusion

We have seen that the overhead of the gBSD description can be quite high depending on the actual bitrate of the video stream. This is true for both a limited case scenario as well as for a detailed description. The difference is that for a limited case scenario we can precalculate the estimated size of the description and it is the same whether we use two cases for a 2048kbps or a 128kbps bitstream. For the detailed description we retain the full flexibility of the wavelet codec even for in-network adaptation. At the same time however the description can become quite large, this is especially true for video sequences which attain a high compression ratio with the codec. Thus the use of either limited cases or detailed description depends on the application scenario. The main problem with the detailed description is that the network node can not judge how much of the gBSD will remain after scaling. As such, it is hard to allocate an overhead bandwidth to calculate the target rate to which to scale the video sequence. It would be possible to do a number of iterations but doing so would result in delay and higher computational load on the node. As such, the detailed description is somewhat problematic to use when the gBSD actually has to be transferred over the network link. However, there are scenarios when this is not necessary, e.g. the example in fig 1. Here we have a high bandwidth link to the access point where we can transfer the video sequence without problem. The ability to do a fine grained scaling is beneficial here since we can optimally use the available bandwidth to the client. Furthermore, the end client does not need the gBSD anymore so the possible overhead is of no concern here. The limited case scenario is for applications where the gBSD needs to be sent too. Since we can approximate how much overhead the description will take the bitstream can be scaled with that in mind. This prevents bandwidth problems and still enables us to do scaling, the only drawback is that we lose the full flexibility of the wavelet codec once the gBSD is generated. However, for generation of the gBSD we still enjoy that same flexibility. Since the bitstream does not need to be altered we can tailor the gBSD specifically to any application scenario given.

Overall we have seen that there is a cost involved in using gBSD to enable in-network adaptation. On the other hand we can bring the flexibility of wavelet based codecs to the network. This brings us closer to the UMA idea of serving every possible end device in a flexible way without having to re-encode the video sequence when new application scenarios arise.

## References

[1] N. Adami, A. Signoroni, and R. Leonardi. State-of-the-art and trends in scalable video compression with wavelet-based approaches. *Circuits and Systems for Video Technology, IEEE Transactions on*, 17(9):1238–1255, Sept. 2007.

[2] C. J. Augeri, D. A. Bulutoglu, B. E. Mullins, R. O. Baldwin, and I. L. C. Baird. An analysis of xml

compression efficiency. In *ExpCS '07: Proceedings of the 2007 workshop on Experimental computer science*, page 7, New York, NY, USA, 2007. ACM.

[3] H. Eeckhaut, H. Devos, P. Lambert, D. De Schrijver, W. Van Lancker, V. Nollet, P. Avasare, T. Clerckx, F. Verdicchio, M. Christiaens, P. Schelkens, R. Van de Walle, and D. Stroobandt. Scalable, wavelet-based video: From server to hardware-accelerated client. *Multimedia, IEEE Transactions on*, 9(7):1508–1519, Nov. 2007.

[4] S.-T. Hsiang and J. W. Woods. Embedded video coding using invertible motion compensated 3-D sub-band/wavelet filter bank. *Signal Processing: Image Communication*, 16(8):705–724, May 2001.

[5] ISO/IEC 21000-7:2007. Information technology – Multimedia framework (MPEG-21) – Part 7: Digital Item Adaptation, Nov. 2007.

[6] R. Kuschnig, I. Kofler, M. Ransburg, and H. Hellwagner. Design options and comparison of in-network H.264/SVC adaptation. *Journal of Visual Communication and Image Representation*, Dec. 2008.

[7] L. Lima, F. Manerba, N. Adami, A. Signoroni, and R. Leonardi. Wavelet-based encoding for HD applications. In *Multimedia and Expo, 2007 IEEE International Conference on*, pages 1351–1354, July 2007.

[8] G. Panis, A. Hutter, J. Heuer, H. Hellwagner, H. Kosch, C. Timmerer, S. Devillers, and M. Amielh. Bitstream syntax description: a tool for multimedia resource adaptation within MPEG-21. In *Special Issue on Multimedia Adaptation*, volume 18 of *Signal Processing: Image Communication*, pages 721–747, Sept. 2003.

[9] Timmerer, C., Kofler, I., Liegl, J., and Hellwagner, H. An Evaluation of Existing Metadata Compression and Encoding Technologies for MPEG-21 Applications. In *Proceedings of the 1st IEEE International Workshop on Multimedia Information Processing and Retrieval (IEEE-MIPR 2005)*, pages 534–539, Irvine, California, USA, December 2005.

[10] A. Vetro, C. Christopoulos, and T. Ebrahimi. From the guest editors - Universal multimedia access. *IEEE Signal Processing Magazine*, 20(2):16 – 16, 2003.

[11] D. Wu, Y. T. Hou, W. Zhu, Y.-Q. Zhang, and J. M. Peha. Streaming video over the internet: approaches and directions. In *Circuits and Systems for Video Technology, IEEE Transactions on*, volume 11, pages 282–300, Mar 2001.

[12] Y. Wu, A. Golwelkar, and J. W. Woods. MC-EZBC video proposal from Rensselaer Polytechnic Institute. *ISO/IEC JTC1/SC29/WG11, MPEG2004/M10569/S15*, Mar. 2004.