Secure Transport and Adaptation of MC-EZBC Video Utilizing H.264-based Transport Protocols[☆]

Hermann Hellwagner[b], Heinz Hofbauer[a,*], Robert Kuschnig[b], Thomas Stütz[a], Andreas Uhl[a]

[a] *Department of Computer Sciences, University of Salzburg*
Jakob-Haringer-Straße 2, 5020 Salzburg, Austria
[b] *Institute of Information Technology, Klagenfurt University*
Universitätsstraße 65–67, 9020 Klagenfurt, Austria

Abstract

Universal Multimedia Access (UMA) calls for solutions where content is created once and subsequently adapted to given requirements. With regard to UMA and scalability, which is required often due to a wide variety of end clients, the best suited codecs are wavelet based (like the MC-EZBC) due to their inherent high number of scaling options. However, most transport technologies for delivering videos to end clients are targeted toward the H.264/AVC standard or, if scalability is required, the H.264/SVC. In this paper we will introduce a mapping of the MC-EZBC bitstream to existing H.264/SVC based streaming and scaling protocols. This enables the use of highly scalable wavelet based codecs on the one hand and the utilization of already existing network technologies without accruing high implementation costs on the other hand. Furthermore, we will evaluate different scaling options in order to choose the best option for given requirements. Additionally, we will evaluate different encryption options based on transport and bitstream encryption for use cases where digital rights management is required.

Keywords: Scalable Video Coding (MC-EZBC), In-network Adaptation, RTP/SRTP MANE, generic Bitstream Syntax Description (gBSD), Video Encryption, Selective Encryption, Format Compliance

1. Introduction

The use of digital video in today's world is ubiquitous. Content consumers desire to retrieve content through a multitude of networks, from 3G to broad-

[☆]Supported by Austrian Science Fund (FWF) project P19159-N13.
[*]Corresponding author
Email addresses: hermann.hellwagner@uni-klu.ac.at (Hermann Hellwagner), hhofbaue@cosy.sbg.ac.at (Heinz Hofbauer), robert.kuschnig@uni-klu.ac.at (Robert Kuschnig), tstuetz@cosy.sbg.ac.at (Thomas Stütz), uhl@cosy.sbg.ac.at (Andreas Uhl)

band Internet, on a broad range of consumer devices, from cell phones to high performance PCs. However, consumers do not care about the technicality necessary to provide the content over this wide range of networks but rather about their quality of experience (QoE), i.e., they want to consume the best possible quality in a timely manner. This creates a problem for content providers since it is costly, in both time and storage space consumption, to provide content for every conceivable end device and network link. Re-encoding on the other hand is expensive in the way that it requires significant time which reduces the QoE for end users.

The solution to this problem is called Universal Multimedia Access (UMA) [1]. The goal of UMA is to encode content once and adapt it in a timely manner to current end user requirements. 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 requires computation time and reduces the number of connections the server can accept. Furthermore, variable bandwidth conditions, which happen frequently on mobile devices, further tax 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. The core adaptation with these restrictions takes place on the server and adaptation due to actual channel capability is done in-network.

For video streaming in the UMA environment, i.e., a high number of possible bandwidths and target resolutions, wavelet based codecs should be considered. Wavelet based codecs are naturally highly scalable and rate adaptation as well as spatial and temporal scaling is easily achieved. Furthermore, wavelet based codecs achieve a coding performance similar to H.264/SVC, c.f. Lima et al. [2]. Under similar considerations Eeckhaut et al. [3] developed a complete server to client video delivery chain for scalable wavelet-based video. However, there are already standardized ways of transporting multimedia data, namely the Real-time Transport Protocol (RTP) [4]. Similarly, there is a protocol for handling a single or several time-synchronized stream of continuous media, e.g., audio and video, the Real Time Streaming Protocol (RTSP) [5] which can use RTP as its mode of transportation. Besides RTP and RTSP the MPEG-21 Part 7 "Digital Item Adaptation" (DIA) [6] can be used to provide content related metadata. A codec agnostic description, the generic Bitstream Syntax Description (gBSD) [7], can also be used as a basis for an informed adaptation process.

In order to use existing technology, i.e., RTP streaming and in network adaptation, modules for handling the motion compensated embedded zero bit codec (MC-EZBC) have to be created to facilitate packetization for RTP and media awareness for adaptation nodes. However, the existing technology can already deal with H.264/SVC, e.g., [8] describes the H.264/AVC payload for RTP and multimedia aware network elements (MANE) and [9] extends this to H.264/SVC. Since the H.264/* bitstream is build from network abstraction layer units (NALUs), the fastest route to utilize the existing infrastructure is to encapsulate the MC-EZBC into a NALU bitstream which presents itself as H.264/SVC to those components. Following this route it is, apart from the

MC-EZBC to NALU conversion, trivial to use the existing infrastructure. Also note that, while we only take a look at MC-EZBC to NALU conversion, such a conversion can be constructed for other scalable video codecs and the theoretical and experimental analysis will by and large also hold for those conversions.

In this paper we will provide a method of encapsulating the MC-EZBC into a NALU bitstream. Additionally, we will investigate how this encapsulated bitstream can be transported, encrypted, and scaled, and at what cost in terms of payload overhead and network delay. Furthermore, we will look at surrounding issues which have to be taken into account, e.g., initial vectors for encryption.

In section 1.1 we will describe the basics of the chosen wavelet based video codec, the MC-EZBC, in section 2.1 a description of the layout of the bitstream will be given and the adaptation to the RTP packetization scheme will be given in section 2.2. An overview of the MPEG-21 DIA generic Bitstream Syntax Description (gBSD) will be given in section 2.3. Section 2.4 describes additional requirements for the RTP streaming process for the MC-EZBC and presents the outline of the encapsulation process.

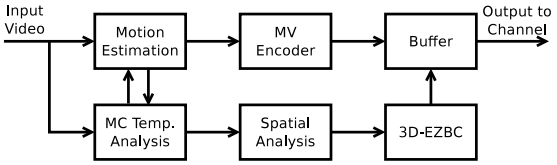
The main concern of research regarding UMA is usually performance with respect to scaling and in-network adaptation. However, digital rights management and security is also a prime concern for providers of commercial videos. Furthermore there are a range of other aspects of video streaming, ranging from server requirements to protocols, to QoS etc., Wu et al. [10] give a good overview of these aspects. General principles and possible goals of digital rights management (DRM) will be explained in section 1.2 and application of encryption to the MC-EZBC codec will be discussed in section 3.

In section 4 we will compare the different aspects and options of the adaptation and streaming process theoretically and experimentally.

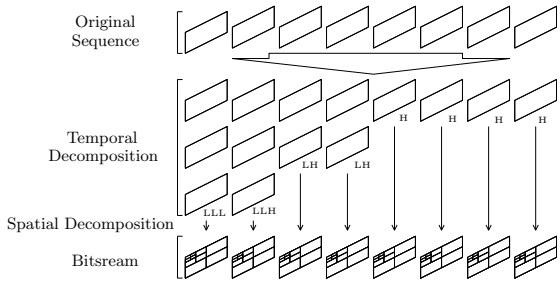
1.1. The Motion Compensated Embedded Zero Bit Codec (MC-EZBC)

For reasons of scalability which fit the UMA principle we use the enhanced MC-EZBC wavelet based video codec for in-network adaptation. This choice was made mainly because the source code is available [1], which enables our experiments. The MC-EZBC codec [11, 12, 13, 14] 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, an overview of the encoding pipeline is given in fig. 1a. This method, temporal first and spatial later, is referred to as t+2D coding scheme, see fig. 1b for an example of this decomposition for a group of picture (GOP) size of 8. For temporal filtering a full decomposition is used and thus the GOP size is discernible by the number of temporal 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.

[1] The source for the ENH-MC-EZBC is available from `http://www.cipr.rpi.edu/research/mcezbc/`.



(a) Overview of the coding pipeline.



(b) Decomposition of a GOP of size 8 showing the arrangement of temporally and spatially decomposed frames in the bitstream.

Figure 1: MC-EZBC encoding overview.

For an overview of wavelet based video codecs and a performance analysis as well as techniques used in those codecs see the overview paper by Adami et al. [15]. Again, while we concentrate only on the MC-EZBC in this paper the encapsulation process described later can in a modified version still be applied to other scalable video codecs. Likewise the analysis performed will also be indicative for other scalable video codecs.

1.2. Overview of Encryption and Digital Rights Management

Shannon's work [16] on security and communication shows that the highest security is reached through a secure cipher operating on almost redundancy free plain text. Current video codecs exploit redundancy for compression and we can consider the bitstream to be a redundancy free plain text in the sense of Shannon. Thus for maximum security we just need to encrypt the whole bitstream with an state of the art cipher, i.e., the Advanced Encryption Standard (AES) [17]. However, the choice was made to keep information in plain text in order to facilitate scalability in the encrypted sequence. Regarding security, Lookabaugh et al. [18] showed that such a selective encryption is sound and demonstrated its relation to Shannon's work. However, Said [19] showed that side information can compromise security.

Thus, we can differentiate between:

Traditional Encryption or full encryption where the full range of the plaintext is encrypted and security in the sense of Shannon is achieved.

Selective Encryption or partial encryption where, carefully selected, parts of the plaintext are left unencrypted. Two common reasons for this approach are reduction in resources, usually time saved when only a part of a plaintext is encrypted, or maintaining properties of the plaintext in the encrypted domain.

The encryption approach used for the MC-EZBC is of the second kind where the objective is to retain the ability to scale the encrypted bitstream, which is not possible when using traditional encryption.

Furthermore selective encryption can be utilized to protect only parts of the bitstream for digital rights management (DRM) scenarios, e.g., a freely decodeable preview version with embedded but encrypted high quality version. The possible security goals we want to achieve with selective encryption in different DRM scenarios are as follows:

Confidentiality Encryption means MP security (message privacy). The formal notion is that if a system is MP-secure an attacker can not efficiently compute any property of the plaintext from the ciphertext [20].

Sufficient Encryption means we do not require full security, just enough security to prevent abuse of the data. Regarding video this could for example refer to destroying visual quality to a degree which prevents a pleasant viewing experience.

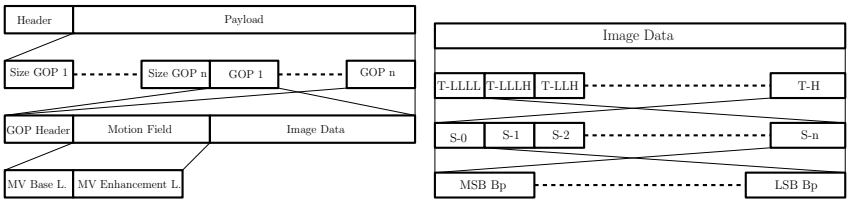
Transparent Encryption means we want consumers to be able to view a preview version of the video but in a lower quality while preventing them from seeing a full version. This is basically a pay per view scheme where a lower quality preview version is available from the outset to attract the viewer's interest. The distinction is that for sufficient encryption we do not have a minimum quality requirement, and often encryption schemes which can do sufficient encryption cannot ensure a certain quality and are thus unable to provide transparent encryption.

2. Particulars of the Protocols

In this section we will describe the details of the MC-EZBC bitstream which are required to perform scaling. Furthermore we will describe the NALU bitstream requirements related to the encapsulation of the MC-EZBC bitstream in order to provide scalability on the transport layer. Likewise, the subset of gBSD syntax elements related to describing the MC-EZBC bitstream are discussed. The requirements introduced by utilizing the RTP are explained and an overview of the process which encapsualtes the MC-EZBC bitstream into gBSD and NALU with respect to RTP are presented.

2.1. MC-EZBC Bitstream

The basic layout of the MC-EZBC bitstream is depicted in fig. 2a and a more detailed overview of the 'image data' required for fine grain scalability is shown



(a) General bitstream layout. (b) Layout of image data in the bitstream.

Figure 2: Layout of the MC-EZBC bitstream.

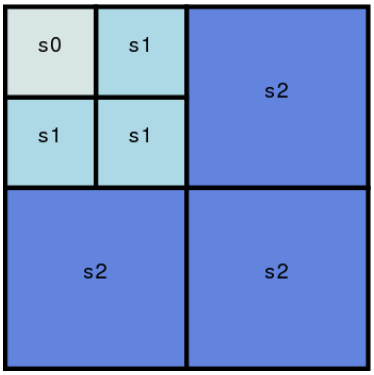


Figure 3: Grouping of decompositions for a frame with two spatial decomposition levels.

in fig. 2b. 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 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 specifying the 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 GOP, i.e.: Header, motion vectors of GOP 1, image data of GOP 1, motions vectors of GOP 2, and so on. Each GOP contains a GOP header, containing 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 not changed during scaling, i.e., motion vectors are not scaled with the image data. Following the motion field is the image data in frame order of temporal decomposition, c.f. fig. 1b and fig. 2 lower part.

The layout of the image data consists of a number of data chunks consisting

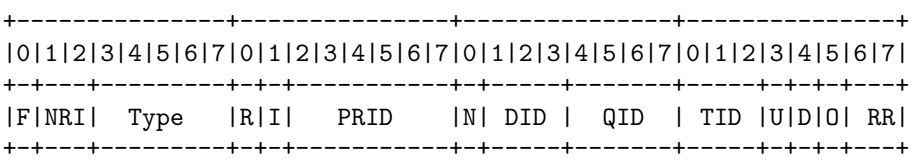


Figure 4: Schematic of the NALU header with SVC extension.

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. 3 illustrates this. The order of these 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, consequently a description of the bitstream which allows scaling has to include access to chunk size information. For a limited number of scaling options, this would be enough since the chunk data can be subdivided into blocks which we can remove. In each chunk there is a three byte header which must never be removed for regular scaling, however when the whole resolution is dropped these three bytes can be dropped too.

2.2. NALU Bitstream

The layout of the MC-EZBC bitstream lends itself naturally to the transformation into a NALU bitstream. In the following we will describe the layout of a valid NALU bitstream as well as an adaptation scheme of the MC-EZBC bitstream. The NALU bitstream is composed of NALU headers, marker segments and payload. In order to properly parse the NALU bitstream, the headers need to be valid, the payload must not contain marker sequences and a marker sequence has to properly indicate the end of a payload segment.

Figure 4 shows the NALU header with SVC header extension, which is used exclusively in our case.

The fields we use for adaptation are:

PRID The priority ID is a 6-bit field which provides application specific priority settings and is used to specify the encoded bitstream part.

TID Temporal ID is a 3-bit field specifying the temporal level and is mapped to the temporal decomposition level.

DID Dependency ID is a 3-bit field which provides inter-layer dependency, i.e., higher DID depends on lower DID, and is used to indicate spatial decomposition level.

QID The quality ID is a 4-bit field specifying quality level dependency and is used to further subdivide a spatial decomposition level into bit rate adaptation cutting points.

More specifically, since we always use the SVC extension header, the header type (denoted by `Type`) is always set to 20. The priority ID reflects the type of data from the original MC-EZBC bitstream: header information (PID 0), GOP header information (PID 1), motion field (PID 2) and image data (PID 3). The GOP length information of the original MC-EZBC bitstream is dropped.

In order to ensure that no marker sequences appear in the bitstream, an escape sequence can be used to escape such marker information. The following table shows the transforms:

`0x000000` → `0x00000300`
`0x000001` → `0x00000301`
`0x000002` → `0x00000302`
`0x000003` → `0x00000303`

Also note that the escaped sequences are not allowed to appear in the bitstream but since this is done by inserting `0x03` and the fact that `0x000003` is also in the marker sequence list this problem solves itself.

Another problem with transforming the bitstream is that the NALU header is prefixed with a marker sequence which is of the form `0x0000 (00)* 01`. Usually three byte sequences are used, except for the the first header which uses a four byte sequence as a synchronization marker. The problem is the arbitrary number of zero bytes in the marker sequence. The specification was done with H.264/SVC in mind where an encoded slice can not end in `0x00`. For the MC-EZBC however this is not the case and thus a trailing zero byte would be counted as belonging to a marker and be lost. To fix this, we append `0x03` to the end of every payload.

The transformation from the NALU bitstream to a MC-EZBC bitstream is a bit more complicated. The data in the NALU bitstream follows the same order as the bitstream representation of the MC-EZBC, i.e., no reordering has to be performed. But since we need to reconstruct the header information for the MC-EZBC bitstream in case scaling occurred, we need to put the information in a treelike structure representing the temporal and spatial decompositions of the MC-EZBC. This is done by monitoring drops in NALU header fields, i.e., a drop in a field refers to parsing a lower *ID value than the previous parsed *ID value. For example, if a drop in the QID occurs we move to a different spatial decomposition or a different frame, depending if a drop in DID is also detected, or to a different GOP and so on. After this is done, we need to restructure the whole bitstream in order to find the maximum decomposition levels, e.g., if there is a resolution drop in one GOP, the other GOPs need to be adjusted to reflect this, in order to properly determine a resolution for the overall header. When this is done the overall header information is calculated and corrected and the GOP length information which was dropped in the transformation to the NALU bitstream is reconstructed.

2.3. gBSD

The gBSD is part of the MPEG-21 part 7 "Digital Item Adaptation" and is used to describe a bitstream in a format agnostic way. This enables devices to

understand a single high level interface (gBSD) and thus perform operations on a bitstream, e.g., scaling, without knowledge about the actual bitstream. While the gBSD allows more structural information to go into the description, we will keep the bitstream description simple so as not to generate too much overhead. For more information on the tags and attributes used see MPEG-21 part 7 [6].

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 descriptor specifying that a part of the original bitstream should be retained in the scaled version. The `gBSDUnit` tag is used for this purpose, it takes start and length information to mark a part of the bitstream to be kept.

Additionally we need access to the bitstream in positions where the header has to be adapted, e.g., size information in a scaling case. Such information can not be copied over from the original bitstream but has to be adapted depending on the target resolution or bitrate. The `Parameter` tag is used for this purpose and 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.

Figure 5 shows a part of the description of the bitstream for the flower sequence which can be used to scale to 1024kbps and 512kbps. 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 bitstream consists of two `gBSDUnit` descriptions discerning between 512 and 1024 kbps.

In order to perform repeated adaptations in the network, the gBSD has to encompass all adaptation possibilities and has to be kept accurate. In order to do this, the gBSD has to be adapted via extensible stylesheet language transformations (XSLT) which is done on the network adaptation node. However, the more fine grained the adaptation choices should be, the more fine grained the gBSD has to be which results in a bigger gBSD file and a more complicated XSLT script. The gBSD together with the XSLT script produce an overhead which limits the size of the actual bitstream, so it is best to keep them as simple as possible. Furthermore, if no more adaptation steps are necessary, the gBSD file can be dropped, i.e., from the last node in the network to the end device the full channel bandwidth can be used.

Figure 6 illustrates how gBSD is used for adaptation, fig. 6a shows the overall

```
...
<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>
```

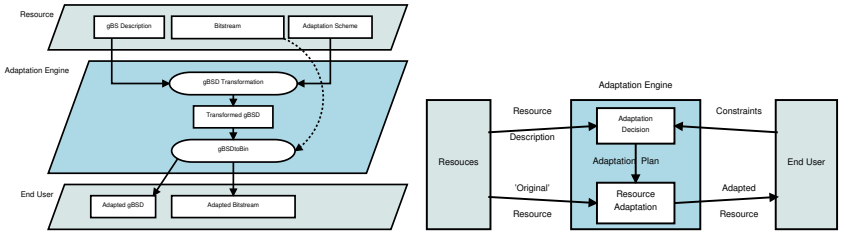
Figure 5: gBSD representation of the flower sequences quality scaling options for 1024 kbps and 512kbps.

layout of an adaptation process, a bitstream and a corresponding gBSD are sent together. According to an adaptation scheme the adaptation engine can scale the bitstream, and adapt the gBSD to fit the scaled bitstream. The adaptation scheme can be fixed, i.e., only certain fixed scaling options are included, or it can be generated based on user preference or requirement, this part of the adaptation engine process is illustrated in fig. 6b. The adaptation based on user preference, especially if more than one user is involved, however increases the size of the gBSD since more options have to be taken into account. Furthermore, either the overhead is increased by creating a more complex adaptation scheme, which anticipates possible user preferences, or the delay is increased by having the adaptation engine request a custom adaptation scheme from the server. A more detailed information about the gBSD adaptation of the MC-EZBC is available in [21]. The paper also shows that there are problems with the gBSD for different types of sequence, like the increase in relative gBSD description size in low motion sequences.

2.4. RTP

Apart from the NALU encapsulation the RTP streaming requires timing information for the packetization, cf. [8]. Furthermore, in order to stream the gBSD with along the same channel utilizing RTP it has to be embedded in the NALU bitstream. This is done by adding supplemental enhancement information (SEI) messages, cf. [22], to the NALU bitstream.

In order to produce timing information for the RTP server, the conversion from MC-EZBC to NALU will also produce an XML output which describes



(a) Overview of a gBSD based adaptation. (b) Overview of the gBSD adaptation process.

Figure 6: Overview of the gBSD adaptation planning process.

the resulting NALU bitstream, including timing information which can be calculated from the framerate given in the original MC-EZBC header and the frame number. This XML description can be used not only as a source of timing information but also as a basis to generate interleaved gBSD descriptions. Should a gBSD be required the produced XML description can be annotated to create the basis of an SEI embedded gBSD description. The annotated XML file can then be broken up to conform to the desired access units (AU) of the bitstream, i.e., the interleaving granularity. This AU gBSD fragments are then compressed and wrapped in an SEI message and inserted into the NALU bitstream in such a way that they precede the AU which they describe. Figure. 7 gives a schematic overview of the transformation process.

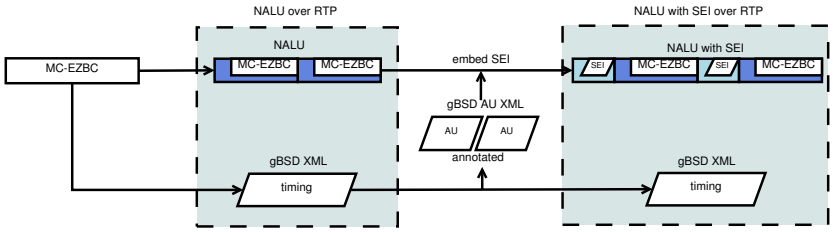


Figure 7: Scheme for MC-EZBC to NALU encapsulation with SEI embedding.

3. Encryption

In order to encrypt the content and still retain the ability to scale there are two options, content encryption, i.e., encrypt the bitstream either on a MC-EZBC or NALU level, or transport encryption. Both methods have advantages and disadvantages regarding computational requirements and security provided.

3.1. Transport Encryption

Transport encryption can be done by using the Secure Real-time Transport Protocol (SRTP), defined in RFC3711 [23]. SRTP is a profile to the Real-

Time Transport Protocol (RTP), defined in RFC3550 [4], providing encryption, message authentication and protection against replay attacks for both uni- and multicast.

The drawback of using the SRTP is the need to decrypt the whole communication on any MANE, where potential scaling takes place. The decryption on each MANE is required, whether scaling is performed or not, in order to inspect the bitstream to determine if scaling has to be performed. This puts a high computational strain on the MANE, which has to decrypt as well as encrypt, compared to encryption only on the server and decryption only on the client. Furthermore, since the key for decryption has to be known on any MANE where scaling can take place each MANE introduces a potential attack point to the system.
On the other hand, we gain security against replay attacks since the whole of the communication is encrypted. Furthermore, the delay for delivery to the consumer is reduced in comparison to prior encryption. Since no prior encryption is employed the streaming can start sooner and the overhead of encryption is distributed in time over the whole streaming process.
However, this option still does not provide confidential security akin to traditional encryption. Due to the headers of the encapsulating SRTP packages remaining in plain text, the length information can be used combined with side channel attacks to compromise security, see Hellwagner et al. [24].

3.2. Bitstream Encryption

For bitstream encryption the choices are either to encrypt the MC-EZBC prior to encapsulating it into a NALU bitstream, or to directly encrypt the NALU bitstream. However, the use of the NALU bitstream for encryption is somewhat problematic. A cipher should optimally produce output resembling a uniform distribution, and thus can create marker sequences which have a special meaning, cf. section 2.2. However, the creation of marker sequences can be prevented or remedied, for a more detailed discussion of NALU encryption see Hellwagner et al. [24], in this paper we will not look into encryption on a NALU level. Encrypting the MC-EZBC on the other hand is easier in technical terms. Through the length information in data chunks, no marker sequences are needed and the content of a chunk can be directly encrypted. Furthermore, the transformation to NALU automatically takes care of possible NALU marker sequences as described in section 2.2. When utilizing UMA, a highest quality source video is used; thus in order to reduce computational cost, encryption should be performed after defining quality levels, i.e., only a part of the source video is used. Furthermore to reduce parsing cost the best option is to include encryption into the NALU encapsulation process. When the encryption is applied just prior to NALU encapsulation, the occurrence of possible marker sequences is automatically taken care of by the encapsulation process.

There are a number of options on how to encrypt the MC-EZBC bitstream depending on the desired results in terms of DRM, i.e., transparent or sufficient encryption, discussed in more detail in [25]. However, in order to allow scaling in the encrypted domain, information about the bitstream has to be kept in

plaintext, i.e., headers. This information can be used in side channel attacks as shown in [26]. In these side channel attacks, the fact that the lengths of encoded video sequence parts correlate to the contained video material is exploited. This can be combined with the information of possible streaming content (the side channel) to identify which video is streamed. While this does not allow an attacker to reconstruct the visual material, the confidentiality is broken. In this section we will mainly look initial vectors for encryption, how the encryption schemes presented in [25] can be used in the NALU encapsulation scenario and how they compare to transport encryption (SRTP).

3.2.1. Considerations for the Initial Vectors

The high scalability of the MC-EZBC bitstream introduces some requirements for a potential encryption method. First and foremost is the ability to perform quality scalability which enables the bitstream to be cut at byte aligned positions. This enforces the use of stream ciphers or block ciphers in streaming mode, e.g., AES in CBC, CFB, OFB or counter mode, [27]. Additionally, due to the scalability in temporal and spatial resolutions as well as scalability in quality, a cipher needs to be restarted for each new chunk. Ciphertext feedback (CBC, CFB) is obviously not able to bridge the resulting gap of data, since ciphertext feedback uses prior ciphertext information to generate a key for following ciphertext. In case of missing data, the keystream for following ciphertext can no longer be constructed. But since information about the original length of a chunk is not kept, pre-ciphertext feedback (OFB, CTR) are also unable to continue over this gap of data. In this case the ciphertext itself is not needed but an iteration is performed in order to construct the keystream and the number of iterations is tied to the length of the missing data. This requires some form of providing an initial vector (IV) for each chunk of data. The data of a chunk has no fixed minimal length and can be scaled down to arbitrary small size. This prohibits the use of plaintext or ciphertext for crafting new IVs for the next chunk in the bitstream.

The solution is to send IVs separately or generate them from a separate source. A separate source could be a single IV which is encrypted to generate a different IV and thus iteratively generate the IVs of the chunks as they appear in the bitstream. This can however lead to synchronization errors, i.e., when a whole GOP is dropped, the next chunk in the bitstream and all subsequent chunks would receive faulty IVs. This happens because the GOPs are not numbered and synchronization can not be restored. Something similar can happen when a whole frame is dropped, then from this frame forward the rest of the GOP would receive a faulty IV. However the next GOP can be properly synchronized because the number of frames in a GOP is known. Similarly, a dropped spatial resolution level would result in the faulty IVs for the rest of the frame and be synchronized at the beginning of the next frame.

This leads to the following options:

- Send a limited number of IVs and generate subsequent IVs by iterated encryption:

  - A single IV is sent at the beginning, resulting in the lowest overhead but can result in synchronization loss for the whole bitstream when a whole GOP is dropped during transport.
  - An IV is sent for each GOP, leading to synchronization at the GOP borders but frame drops can destroy the rest of the current GOP.
  - An IV is sent for each frame, synchronization is now per frame but a resolution drop can destroy the rest of the current frame.
- Send a single IV for each chunk of data in the bitstream. This has the highest overhead but desynchronization can not occur.

Regarding overhead we can give a simple upper bound by looking at the number of spatial resolutions. The number of frames per GOP remains the same since full temporal decomposition is used. Assuming a framerate of f with s spatial decomposition levels we can simply give the overhead as:

$$o_{IV} = f * (s + 1) * b,$$

for a block size of b. For AES of a PAL video, a resolution of 768x576, 6 decomposition steps and 25 frames per second, this would result in an overhead of $o_{IV} = 21.875$kbps. To put this into relation, consider streaming over an old, low bandwidth IEEE 802.11 WLAN with a channel capacity of 2Mbps this would be $\approx 0.01\%$ of the channel capacity. In essence, the overhead of sending frequent IVs is negligible and does hardly impact channel bandwidth. For newer WLAN standards featuring higher bandwidth the overhead of sending frequent IVs becomes even less of a problem.

4. Comparison and Evaluation

In this section we will compare the overhead introduced by encapsulation in NALU and gBSD respectively. Since RTP is used as transport protocol for both NALU and gBSD, we will not take into account the RTP overhead since it is the same for both formats.

4.1. Protocol Overhead

In [28] it is shown that seven quality levels are usually enough to support almost all required target applications. While this is a reasonable goal for comparison we will look at the overhead in a more general fashion. This is done mainly because if a request for a certain bandwidth, framerate or bitrate is issued the MC-EZBC source can be transformed on the fly to support the requested target scalability which can lead to an actual lower number of scaling options. As a result of a lower number of scaling options the encapsulating protocols generate less overhead. The encoding overhead is important since the actual bitrate of the bitstream can only be the channel bitstream minus the required overhead.

In the following we will denote the number of frames as f, the number of temporal decompositions as t and the number of spatial decompositions as s. Consequently we have a GOP size of 2^t and the number of GOPs is $G = f/2^t$, for simplicity we assume that the framerate is a multiple of the GOP size, and in total we have $s+1$ spatial decomposition bands. Furthermore we will denote the number of quality levels by q.

4.1.1. Evaluation of gBSD Overhead

For the number of bytes each descriptive element of the gBSD requires, we use an approximation obtained from empirical analysis of the used bitstreams and resulting gBSD descriptions. While most of the markers have a fixed structure, like element and attribute names, the value of the attributes change depending on the encoded sequence, see fig. 5 as an example containing the `gBSDUnit` element. The average size in bytes a `Parameter` and `gBSDUnit` require are $p = 105$ and $g = 55$ bytes respectively. These numbers are calculated with average variable length information (i.e., length value, start value) but excluding the marker attribute since the value is essentially user defined. Additionally, we have an overhead for the DIA declaration which is 393 bytes, which is the length of the fixed header, see. fig. 5. 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 three changeable fields, bitrate, spatial and temporal scaling level, with size p and five `gBSDUnit`s of size g which stay constant. The main header is followed by a list of GOP sizes, with one entry per GOP, each entry in the list is given as a `Parameter` with size 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. The chunks here have to be separated into the number of quality levels we want to deal with. The resulting approximation in byte is thus size S:

$$S = 393 + \underbrace{3p + 5g}_{\text{header}} + \underbrace{G\,p}_{\text{GOP size list}} + G\underbrace{(g + 2^t(s+1)(p+qg))}_{\text{single GOP}}$$

For a sequence with 128 frames, $t = 7$ and $s = 2$ this would estimate a gBSD file size of 81kB for two quality levels and 60kB for the downscaled version. However, this assumes that the gBSD is transferred in plaintext which is unusual. A gBSD description is text based and can be compressed quite well, see Augeri et al. [29] for an overview. Furthermore, there are XML aware compression schemes which are designed for ease of access on network nodes and alleviate the need to decompress the description of the whole bitstream, see Timmerer et al. [30] for an overview. For the rest of this paper we will use bzip2 as compressor for gBSD which will compress by an order of magnitude. For a more detailed overview of gBSD regarding MC-EZBC and compression see Hofbauer et al. [21].

Size increase when including gBSD

	Filesize in byte for		
sequence	NALU	NALU+SEI	increase
bbbunny	12868259	12946658	0.61%
sintel	13424680	13546838	0.91%
football	1139309	1167002	2.43%
harbour	901722	928638	2.98%
crew	676359	702960	3.93%
foreman	449419	474891	5.66%

Table 1: Overhead of bzip2 compressed gBSD SEI inclusion into bitstreams of different quality levels.

While this is an overhead calculation for the whole bitstream it can be used as approximation for the AU based description as well. In order to create a well formed gBSD document the header has to be replicated which increases the overall size, but simultaneously the relative length information from the start of the description is shorter, resulting in lower p and g values. As such the given equation can be either used directly for overhead calculation, or in parts if a better fitting calculation is desired. As an example we can consider the overhead calculation for the whole sequence with GOP based gBSD descriptions. This can be easily done by extracting the single GOP part of the given equation and adding the cost of the header; the resulting overhead has to be taken into account for each GOP. The resulting overhead is

$$S_{AU_{GOP}} = G * (393 + (g + 2^t(s+1)(p+qg)).$$

What is problematic about this overhead is the fact that the overhead size is only dependent on the scaling options but not the quality of the contained bitstream. This means that for a given gBSD description the overhead relative to the size of the bitstream increases with decreasing quality. Table 1 shows an example for this increase in size when including bzip2 compressed SEI messages, as described in fig. 7, for various bitrates. The sequences used in the table are of CIF resolution with GOP size 16, 6 spatial levels and with bitrates 1045kbps(football), 822kbps (harbour), 611kbps (crew) and 398kbps (foreman), and 720p resolution with GOP size 16, 4 spatial levels and bitrates 3072kbps (bbbunny) and 2048kbps (sintel). The CIF sequences have a runtime of 10.24 sec while bbbunny and sintel have a runtime of 33sec and 52sec respectively.

4.1.2. Evaluation of NALU Overhead

For every piece of payload we have to take into account the marker sequence leading up to it (3 bytes), the NALU SVC header (4 bytes) as well as the payload end marker (1 byte). We denote the fixed overhead value as $o_f = 8$. Furthermore we have an overhead of 1 byte since the first NALU marker is a

4 byte synchronization marker, and we have a reduction in size resulting from the drop of the GOP size table of the original MC-EZBC bitstream which gives the overall overhead adjustment $o_o = 1 - G\,4$, since every size entry in the GOP table is a long integer 4 bytes in size. Thus, we can give the overhead as

$$O = o_o + o_f + o_f G(2 + q(s+1)2^t)$$

A NALU is created for the global header and for every GOP q NALUs are created per temporal and spatial resolution. This only reflects the fixed overhead, a further overhead occurs when marker sequences appear in the original bitstream and have to be escaped. However this can not be given in a deterministic fashion. Assuming uniform distribution of byte values we can calculate the chance P of a marker appearing at any given byte position as:

$$P = \underbrace{\frac{1}{2^8}}_{\text{0x00}} * \underbrace{\frac{1}{2^8}}_{\text{0x00}} * \underbrace{\frac{2^2}{2^8}}_{\text{0x}\{00,01,02,03\}} = \frac{1}{2^{22}}.$$

In this unlikely case a single byte is inserted into the three byte sequence, extending it by 4/3, thus on average the size of the bitstream will increase by a factor $F = 1 + \frac{1}{3*2^{20}} \approx 1.00000032$. The increase in size due to this factor is practically negligible. Furthermore, unlike the gBSD overhead this size increase is multiplicative instead of additive, i.e., dependent on the size of the original bitstream. Thus, while the overhead of the gBSD description stays the same for reduced quality versions the overhead due to this factor is reduced together with the bitstream size.

4.2. Encryption Performance

A direct comparison of bitstream encryption and transport encryption is not really possible. Bitstream based encryption is done only on the server and client and introduces a constant delay until streaming can start. Transport encryption (SRTP) on the other hand encrypts while streaming and thus the load on the server and client are distributed over the time it takes to stream the video sequence, but additional load is produced on the MANE where decryption and encryption also has to take place. With SRTP the delay to start streaming is basically shifted to frame delays during transport. As such we will, and can, not provide a direct comparison, rather both methods are looked at differently. Transport encryption will be looked at during the evaluation of adaptation performance since both are tied together.

For bitstream based encryption it is most important to get a notion of how long the delay to start streaming is since this has a direct influence on consumer satisfaction (QoE). In order to evaluate the time requirement for encryption for different DRM scenarios, a number of selective encryption types are used. As a baseline we will use the same cipher used for selective encryption and encrypt the whole bitstream. In order to better gauge the influence of the parsing overhead generated when using selective encryption, the same video sequence is

used but with different quality levels. Table 2a gives the encryption performance for a full quality version of the foreman sequence, the full quality version has a bitrate of about 9.5Mbps. For comparison we use a reduced quality version with a bitrate of 398kbps, which is later also used in the analysis of streaming performance, given in table 2b. For each bitrate version we performed different types of encryption which correlate to possible DRM applications. For more information about the encryption process and resulting quality see [25].

Full selective encryption refers to the encryption of all image data, i.e., excluding headers. Due to the plaintext headers, scaling is still possible with full selective encryption. This method is put in direct comparison with full traditional encryption, i.e., encryption of the whole bitstream including headers and motion fields. This option generates no parsing overhead but does not allow scalability in the encrypted domain. The parsing overhead for both bitrate versions is the same, since the layout of the bitstream is unchanged. This leads to an actual reduction in encryption time for very high quality bitstreams even for full selective encryption. For low bitrates however the overhead is quite significant, in the 398kbps test case the parsing overhead nearly doubles the time required for encryption.

Sufficient encryption refers to a significant reduction in visual quality. This is typically done by encrypting the base layer and leaving the enhancement layers intact. This leads to a significant reduction in encryption time in relation to full selective encryption. The time reduction is more pronounced for higher quality versions of the bitstream because more refinement information is contained in the bitstream and thus the reduction in the amount of data to be encrypted is more pronounced. Table 2 shows the two extremes, on one hand we have a high reduction in quality, and consequently the amount of data which needs encryption. For this case the parsing overhead renders any selective encryption slower than full traditional encryption. On the other hand, the high quality case shows that the parsing overhead becomes negligible in comparison to the amount of data which need encryption. Thus, the higher the quality the more time reduction can be gained from selective encryption.

Transparent encryption usually targets enhancement layer information in order to allow a decoding of a decent base layer quality as preview version. This version normally, except for low quality versions of a bitstream, encrypts an amount of data between sufficient and full encryption. Likewise the amount of time required for encryption is between full selective and sufficient.

Regarding which kind of encryption to use we can distinguish between application scenarios. Since we want to keep scalability intact, full traditional encryption can not be used. When the goal is to produce sufficient encryption, the best option usually is to encrypt I-frames only. I-frames have to be included even when encrypting only lowest spatial bands, in order to prevent the introduction of higher quality content in case of a scene change. Thus, the I-frames only option is in any case faster than the encryption of lowest spatial bands, since this would necessarily include I-frames. When transparent encryption is desired, the options are highest spatial or highest temporal bands. Which option to choose depends strongly on the video sequence, i.e., when encrypting

What was encrypted	Time	% of Bitstream
Sufficient Encryption		
I-frames only	49ms	21.34%
lowest spatial band	80ms	35.54%
lowest temporal band	84ms	39.85%
Transparent Encryption		
highest spatial band	179ms	88.96%
two highest temporal bands	156ms	75.74%
Full Encryption		
full selective encryption	201ms	99.50%
full traditional encryption	207ms	100%

(a) Full quality ($\approx$ 9.5Mbps)

What was encrypted	Time	% of Bitstream
Sufficient Encryption		
I-frames only	16ms	63.42%
lowest spatial band	15ms	65.98%
lowest temporal band	14ms	77.70%
Transparent Encryption		
highest spatial band	13ms	50.02%
two highest temporal bands	12ms	22.00%
Full Encryption		
full selective encryption	18ms	87.60%
full traditional encryption	10ms	100%

(b) Reduced quality (398kbps)

Table 2: The time required for selective encryption and the amount of the bitstream actually encrypted for the foreman sequence with CIF resolution, 256 frames and GOP size of 16.

a scene which contains little motion the drop in framerate from encryption of high temporal bands will hardly be noticeable. Otherwise, encryption of highest temporal bands usually contains less information and consequently is faster. For a more in depth discussion of encryption types and application scenarios see [25].

4.3. Adaptation Performance

As discussed in previous sections there are certain options for scaling and encryption. However, depending on the method chosen, the computational load for adaptation is increased. If SRTP is utilized for encryption, the stream has to be de- and encrypted on the MANE. Likewise, gBSD description allows a more fine grained scalability but introduces an overhead in data sent as well as computational load on the MANE. While effects other than computational requirements have already been discussed, the question of computational load is still open. In this section we will compare gBSD and direct NALU scaling over RTP as well as SRTP to gauge the effects on server, client and MANE.

4.3.1. Evaluation Setup

As test setup we use a loop to measure timing information accurately, i.e., both server and client are on the same machine. The server is connected to the client via a MANE running on a second machine. Both machines have the same hardware, a DELL Optiplex 960 with Intel® Core™2 Quad Q9650 (3GHz, 1333MHz, 2x6MB L2 Cache) CPU with 4GB of DDR2 RAM. The machines are connected via Gigabit LAN using an Intel PRO/1000 GT Network Adapter and run the same software with Ubuntu Linux 10.04 as OS. A schematic drawing of the setup is given in fig. 8.

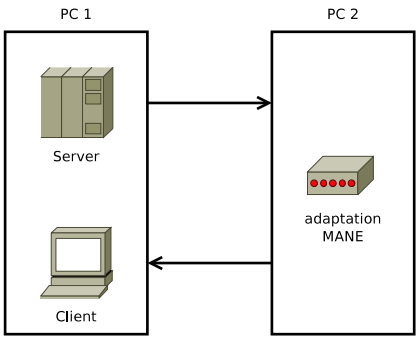


Figure 8: Schematic of test setup

As test sequences the well known crew, football, foreman and harbour sequences are used in CIF resolution with a running length of 10.24sec. The CIF sequences use a GOP size of 16 with a total of 256 frames and an fps of 25. Furthermore, two test sequences are chosen from an application point of view,

the trailers for the Sintel[2] and Big Buck Bunny[3] (abbreviated to bbbunny in tables and figures) movies in 720p resolution with a length of 52sec and 33sec respectively. The two trailers are encoded with a GOP size of 16. For the test each sequence was set to two quality levels. The quality levels and number of possible scaling points for temporal and spatial resolution for all sequences are given in table 3.

Sequence	resolution	T	S	Q1	Q0
bbbunny	720p	4	4	3072	2048
sintel	720p	4	4	2048	1045
football	CIF	4	6	1045	822
harbour	CIF	4	6	822	611
crew	CIF	4	6	611	398
foreman	CIF	4	6	398	256

Table 3: Overview of the video sequences in the testset. The quality levels Q0 and Q1 are given in kbps, the scaling options for temporal (T) and spatial (S) resolution are equal to the number of wavelet decompositions in the respective domain.

Scaling Test	Passed levels					
	CIF Resolution			720p Resolution		
	Temporal	Spatial	Quality	Temporal	Spatial	Quality
None	4	6	2	4	4	2
Temporal	**3**	6	2	**3**	4	2
Spatial	4	**4**	2	4	**3**	2
Quality	4	6	**1**	4	4	**1**

Table 4: Overview of scaling tests, showing which temporal, spatial and quality levels are passed through, bold numbers indicate scaling.

For evaluation, four tests were performed per video sequence, unscaled transport, quality scaling, temporal (framerate) scaling and spatial (resolution) scaling. For each test, 20 streams were simultaneously sent from server to client with adaptation on the MANE. Table 4 gives an overview of which levels are passed during which test. A temporal level of 4 represents the original 16 frames per GOP while a temporal level of 3 indicates a GOP size of 8, and consequently half the original framerate. For each sequence there are two quality levels, the quality levels differ for each sequence and are given in table 3 as Q0 and Q1 respectively. For spatial scaling of CIF sequences, 6 refinement levels reproduce

[2] `http://www.sintel.org`
[3] `http://www.bigbuckbunny.org`

the original CIF resolution while passing only the first 4 levels results in a reduction of resolution to SQCIF 88×72. For 720p sequences, 4 refinement levels reproduce the original sequence at 720p (1280×720) while passing only 3 levels results in a reduction of resolution to 640×360.

4.3.2. In-Network Performance Evaluation

For the performance evaluation we use the testset as described above with both RTP and SRTP. The difference in memory, CPU and frame delay when using NALU and gBSD for adaptation will be investigated. The gBSD is used to describe an underlying NALU bitstream. The NALU bitstream can easily be used to scale spatial and temporal resolution as provided by the MC-EZBC bitstream. Furthermore, during encapsulation of MC-EZBC into a NALU bitstream the number and range of quality scaling points can be freely chosen. However, it is not possible to scale according to higher semantics, e.g., marking certain frames or GOPs as less important. To enable such scaling options, gBSD can be used but this incurs an overhead in the bitstream and, through XML parsing and processing, in computational load. To facilitate a fair comparison, the scaling options for the NALU bitstream as given in table 4 are also used for gBSD testing.

What we expect to see is that the use of gBSD results in a distinct impact on memory and CPU usage on the MANE due to decompression and processing of the XML description. Likewise the use of SRTP is assumed to incur a higher CPU usage on client, server and MANE due to encryption and decryption. Regarding delay in delivery time, both gBSD and SRTP are expected to negatively impact frame delay due to processing cost.

Figure 9 shows the average memory and CPU consumption for the 20 parallel streams on the server, MANE and client for transport via RTP. For each stream 30000 frames were sent. In the figure, NALU refers to scaling based on NALU and SEI refers to scaling with a gBSD description, which is compressed and embedded in the bitstream as SEI messages.

Likewise figure 10 shows the CPU and memory consumption for the same test when using SRTP. This produces an overhead on server, MANE and client due to the encryption and decryption of the bitstream in order to process it. The ordinate for the MANE is different from server and client in order to see the difference for client and server memory and CPU consumption. However, to facilitate comparison between RTP and SRTP the ordinate scales are the same for each case. What is evident from these figures is that encryption for SRTP incurs a significant overhead, especially on the MANE which needs to decrypt as well as encrypt, leading to an almost fourfold increase in CPU consumption. Furthermore, the use of gBSD for scaling results in increased memory consumption on the MANE. This increased memory consumption is more pronounced when the relative size of the gBSD compared the NALU is higher, compare table 1. Additionally the decompression and processing of the SEI gBSD messages increases CPU consumption on the MANE.

In addition to the CPU and memory consumption for scaling, the processing on the MANE incurs a frame delay. Figure 11 plots the cumulative distribution

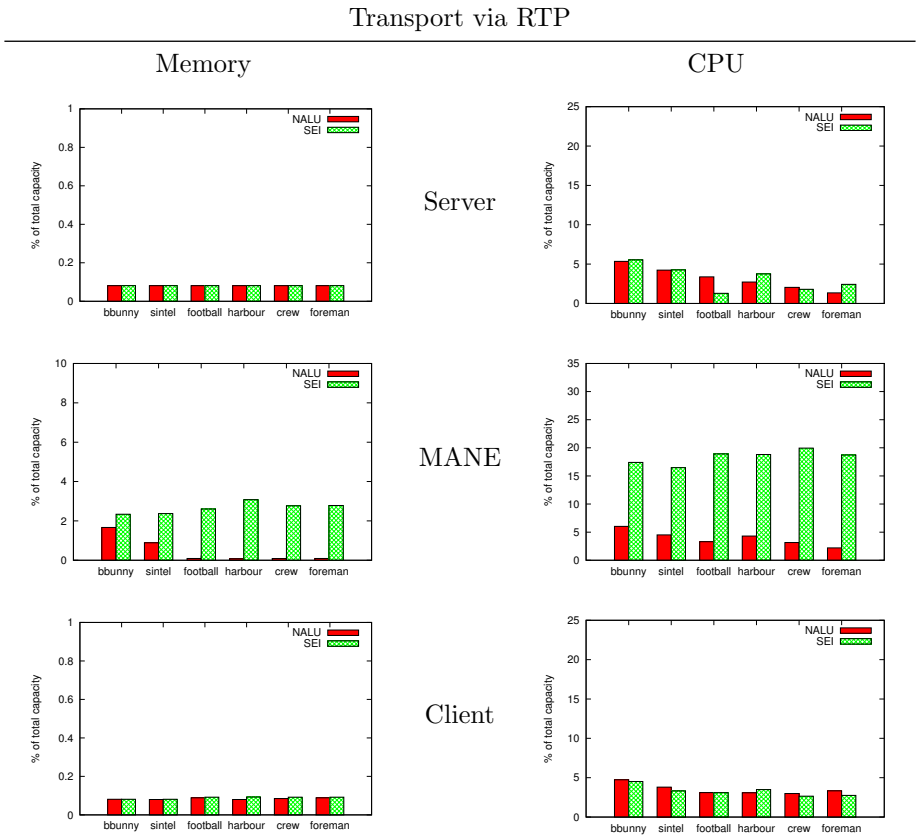


Figure 9: Average of CPU and memory consumption in percent over 20 simultaneous RTP streams and four scaling tests for Server, Client and MANE.

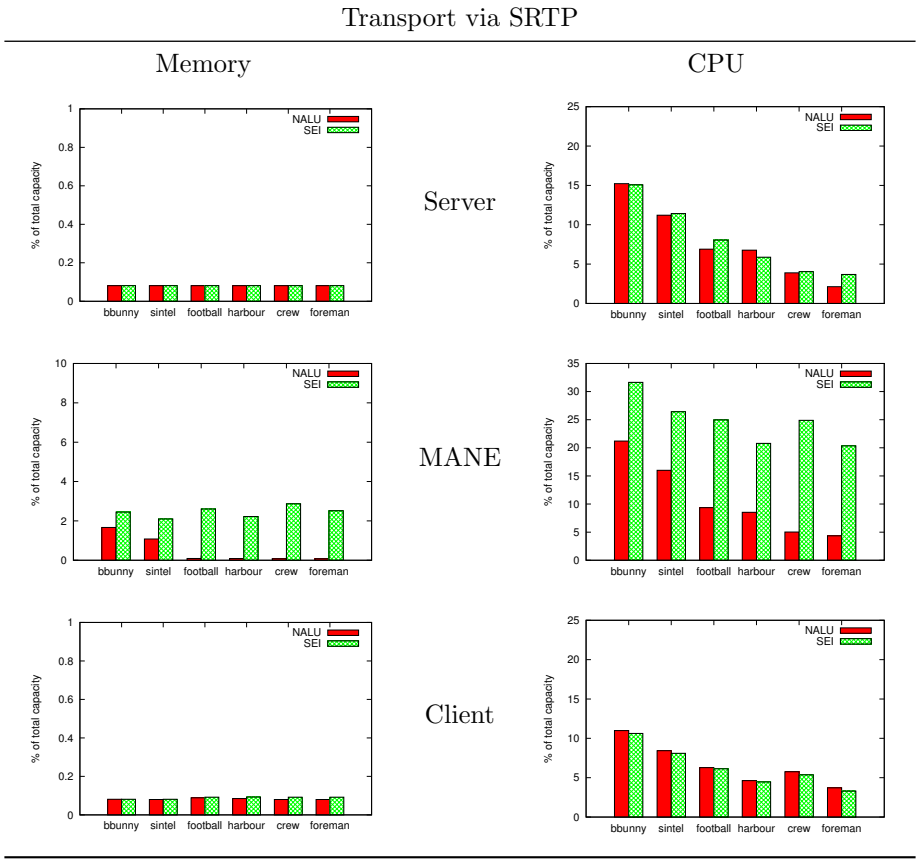


Figure 10: Average of CPU and memory consumption in percent over 20 simultaneous SRTP streams and four scaling tests for Server, Client and MANE.

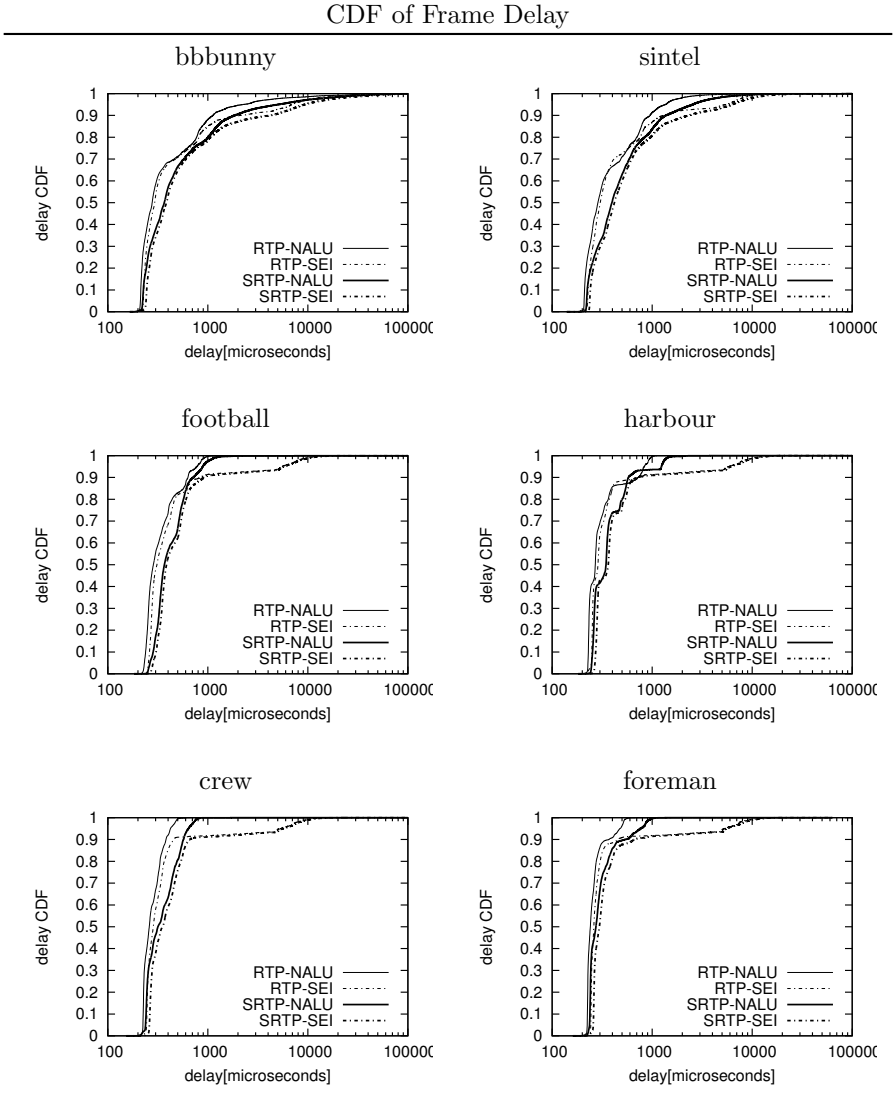


Figure 11: Comparison of cumulative frame delay (CDF) for NALU and SEI adaptation using (S)RTP as transport.

function (CDF) for the frame delay over the actual delay, given in microseconds on a logarithmic scale. This test is done for unscaled contents only since this is the worst case. Any scaling of contents results in less information the MANE has to send and thus smaller outgoing buffers and consequently lower delay. The delay is given based on transport protocol, SRTP and RTP, as well as encapsulation type, NALU and gBSD with SEI. What can be seen is that SRTP causes more delay than RTP since the required decryption and encryption steps have to be performed prior to adaptation checking and sending. Likewise SEI messages incur a higher delay than pure NALU based adaptation. This is due to the fact that the gBSD has to be decompressed and inspected before passing the adapted bitstream along to the client. Furthermore, the higher the bitrate the higher the frame delay, this stems from additional computational demand and fuller outgoing buffers. However, even for higher bitrate sequences the overall relation of SRTP, RTP, NALU and SEI holds.

sequence	RTP		SRTP	
	NALU	SEI	NALU	SEI
		new		
bbunny	13490	18521	28478	40524
sintel	2685	9934	6704	13688
football	925	10072	1165	10471
harbour	962	10428	1410	11136
crew	490	9682	756	10102
foreman	529	9501	938	10327

Table 5: Frame delay in μs for CDF= 0.99.

The CDF plot shows that the overall behavior is as expected, both SEI and SRTP incur a delay in delivery time. To better assess the actual impact rather than the general notion, we will take a closer look at the time delay for CDF= 0.99. Table 5 gives the average time, over 30000 frames, to deliver 99% of the image sequence to the end user, i.e., only 1% of the image sequence will take longer to deliver to the client. What can be seen is that the impact of SEI over NALU is tremendous: for RTP SEI is slower than NALU, but the slowdown becomes less sever the higher the overall processing cost. For SRTP the behavior is similar but overall less pronounces since the encryption and decryption overhead slows down both scaling methods. For NALU the switch from RTP to SRTP incurs a significant slowdown while for SEI the impact of SRTP over RTP is less pronounced. This is due to the decryption and encryption being faster by an order of magnitude than the decoding and parsing of SEI. Table 6 gives the factors of slowdown for all cases.

Overall, the expected impact in delivery time due to SRTP and SEI messages over RTP and NALU can clearly be seen.

NALU → SEI		
sequence	RTP	SRTP
bbbunny	1.37	1.42
sintel	3.70	2.04
football	10.89	8.99
harbour	10.84	7.90
crew	19.76	13.36
foreman	17.96	11.01

(a) Slowdown for RTP and SRTP when switching scaling method from NALU to SEI

RTP → SRTP		
sequence	NALU	SEI
bbbunny	2.11	2.19
sintel	2.50	1.38
football	1.26	1.04
harbour	1.47	1.07
crew	1.54	1.04
foreman	1.77	1.09

(b) Slowdown for NALU and SEI when switching from bitstream encryption to SRTP

Table 6: Frame delay slowdown factor for the different scaling and encryption options.

5. Conclusion

We have introduced a mapping of a wavelet based video coding format, the MC-EZBC format, to an H.264/SVC compatible bitstream in order to utilize existing transport and scaling protocols and technologies, i.e., RTP. Furthermore, we compared the bitstream based encryption to transport encryption, i.e SRTP, and evaluated different scaling technologies, i.e., NALU based adaptation versus MPEG-21 Part 7 'Digital Item Adaptation' with gBSD. In addition we have also provided an overhead estimation which is introduced by the mapping of MC-EZBC to a NALU based bitstream as well as the overhead introduced by the inclusion of gBSD in the bitstream.

When it comes to scaling, it is clear that a NALU based approach is better since it generates less overhead in terms of bitstream size. Furthermore, when compared to gBSD, the memory and CPU consumption on network scaling nodes is lower by a significant amount and consequently NALU based adaptation has a lower frame delay. Consequently, even though the NALU based approach is less flexible than gBSD based adaptation, NALU based adaptation should be the baseline and only in those cases where scalability beyond NALU capabilities is desired a gBSD based description should be used.

Regarding encryption, the available options are transport encryption via SRTP and bitstream based encryption on either NALU or MC-EZBC level. It is clear that encryption of the NALU bitstream provides no benefit over encryption of MC-EZBC bitstream prior to the mapping process. When comparing MC-EZBC based encryption to transport encryption, it was shown that the computational load on scaling network elements is much higher for transport encryption and an additional frame delay is introduced. Furthermore, the encryption and decryption of the streamed video content required on every MANE poses a security risk. However, transport encryption has less overall delay to start streaming than bitstream encryption. Any form of DRM, e.g., transparent encryption multicast with sufficient encryption, required a bitstream based en-

cryption since SRTP can not handle those cases. Confidential encryption is not possible since header attacks to leak information about the streamed content are always possible, whether they operate on the plain text information used to scale the bitstream in-network or on the packetization headers of the streaming protocol.

References

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

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

[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. *IEEE Transactions on Multimedia*, 9(7):1508–1519, November 2007.

[4] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. RFC 3550, July 2003.

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

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

[7] Gabriel Panis, Andreas Hutter, Jörg Heuer, Herman Hellwagner, Harald Kosch, Christian Timmerer, Sylvain Devillers, and Myriam 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.

[8] S. Wenger, M.M. Hannuksela, T. Stockhammer, M. Westerlund, and D. Singer. RTP Payload Format for H.264 Video. RFC 3984, February 2005.

[9] 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*, 19(8):529–542, September 2008.

[10] Dapeng Wu, Yiwei Thomas Hou, Wenwu Zhu, Ya-Qin Zhang, and Jon M. Peha. Streaming video over the Internet: approaches and directions. *IEEE Transactions on Circuits and Systems for Video Technology*, 11(3):282–300, Mar 2001.

[11] Shih-Ta Hsiang and J. W. Woods. Embedded video coding using invertible motion compensated 3-D subband/wavelet filter bank. *Signal Processing: Image Communication*, 16(8):705–724, May 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*, March 2004.

[13] P. Chen, K. Hanke, T. Rusert, and J. W. Woods. Improvements to the MC-EZBC scalable video coder. In *Proceedings of the IEEE Int. Conf. Image Processing ICIP*, volume 2, pages 81–84, Barcelona, Spain, 2003.

[14] Peisong Chen and John W. Woods. Bidirectional MC-EZBC with lifting implementation. *IEEE Transactions on Circ. and Systems for Video Technology*, 14(10):1183–1194, 2004.

[15] 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, September 2007.

[16] Claude E. Shannon. Communication theory of secrecy systems. *Bell System Technical Journal*, 28:656–715, October 1949.

[17] National Institute of Standards and Technology. FIPS-197 - advanced encryption standard (AES), November 2001.

[18] T. D. Lookabaugh and D. C. Sicker. Selective encryption for consumer applications. *IEEE Communications Magazine*, 42(5):124–129, 2004.

[19] A. Said. Measuring the strength of partial encryption schemes. In *Proceedings of the IEEE International Conference on Image Processing (ICIP'05)*, volume 2, pages 1126–1129, September 2005.

[20] M. Bellare, T. Ristenpart, P. Rogaway, and T. Stegers. Format-preserving encryption. In *Proceedings of Selected Areas in Cryptography, SAC '09*, volume 5867, pages 295–312, Calgary, Canada, August 2009. Springer-Verlag.

[21] Heinz Hofbauer and Andreas Uhl. The cost of in-network adaption of the MC-EZBC for universal multimedia access. In *Proceedings of the 6th International Symposium on Image and Signal Processing and Analysis (ISPA '09)*, Salzburg, Austria, September 2009.

[22] ITU-T H.264. Advanced video coding for generic audiovisual services, November 2007.

[23] M. Baugher, D. McGrew, M. Naslund, E. Carrara, and K. Norrman. The Secure Real-time Transport Protocol (SRTP). RFC 3711 (Proposed Standard), March 2004.

[24] Hermann Hellwagner, Robert Kuschnig, Thomas Stütz, and Andreas Uhl. Efficient in-network adaptation of encrypted H.264/SVC content. *Elsevier Journal on Signal Processing: Image Communication*, 24(9):740 – 758, July 2009.

[25] Heinz Hofbauer and Andreas Uhl. Selective encryption of the MC EZBC bitstream for DRM scenarios. In *Proceedings of the 11th ACM Workshop on Multimedia and Security*, pages 161–170, Princeton, New Jersey, USA, September 2009. ACM.

[26] Heinz Hofbauer and Andreas Uhl. Selective encryption of the MC-EZBC bitstream and residual information. In *18th European Signal Processing Conference, 2010 (EUSIPCO-2010)*, pages 2101–2105, Aalborg, Denmark, August 2010.

[27] National Institute of Standards and Technology. FIPS-81 - DES modes of operation, November 1981.

[28] T. Rusert, K. Hanke, P. Chen, and J. W. Woods. Recent improvements to MC-EZBC. *ISO/IEC JTC1/SC29/WG11 MPEG, M9232*, page 14pp., December 2002.

[29] Christopher J. Augeri, Dursun A. Bulutoglu, Barry E. Mullins, Rusty O. Baldwin, and III Leemon 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.

[30] 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.