ETSI TR 126 947 V15.0.0 (2018-07)

Universal Mobile Telecommunications System (UMTS);
LTE;
Multimedia Broadcast/Multicast Service (MBMS);
Selection and characterisation of application layer Forward Error Correction (FEC)
(3GPP TR 26.947 version 15.0.0 Release 15)

Reference
RTR/TSGS-0426947vf00

Keywords
LTE,UMTS

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - NAF 742 C
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° 7803/88

Intellectual Property Rights

Essential patents

IPRs essential or potentially essential to normative deliverables may have been declared to ETSI. The information pertaining to these essential IPRs, if any, is publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (https://ipr.etsi.org/).

Pursuant to the ETSI IPR Policy, no investigation, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Trademarks

The present document may include trademarks and/or tradenames which are asserted and/or registered by their owners. ETSI claims no ownership of these except for any which are indicated as being the property of ETSI, and conveys no right to use or reproduce any trademark and/or tradename. Mention of those trademarks in the present document does not constitute an endorsement by ETSI of products, services or organizations associated with those trademarks.

Foreword

This Technical Report (TR) has been produced by ETSI 3rd Generation Partnership Project (3GPP).

The present document may refer to technical specifications or reports using their 3GPP identities, UMTS identities or GSM identities. These should be interpreted as being references to the corresponding ETSI deliverables.

The cross reference between GSM, UMTS, 3GPP and ETSI identities can be found under http://webapp.etsi.org/key/queryform.asp.

Modal verbs terminology

In the present document "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the ETSI Drafting Rules (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in ETSI deliverables except when used in direct citation.

Contents

Foreword

This Technical Report has been produced by the 3rd Generation Partnership Project (3GPP).

The contents of the present document are subject to continuing work within the TSG and may change following formal TSG approval. Should the TSG modify the contents of the present document, it will be re-released by the TSG with an identifying change of release date and an increase in version number as follows:

Version x.y.z

where:

x the first digit:

1 presented to TSG for information;

2 presented to TSG for approval;

3 or greater indicates TSG approved document under change control.

y the second digit is incremented for all changes of substance, i.e. technical enhancements, corrections, updates, etc.

z the third digit is incremented when editorial only changes have been incorporated in the document.

Introduction

The on-going commercialization of LTE networks has precipitated increasing interest in the deployment of eMBMS. As the industry is considering the first deployments of eMBMS it is important to enhance the performance and usability of its core features.

After development of the initial MBMS specifications, SA4 has focused its subsequent work on adding new features to the service. While application layer FEC usage in support of download and streaming delivery methods have been specified since Rel-6, those mechanisms have not been updated to reflect performance improvement developments in more recent years. Examples of ongoing FEC enhancement efforts are the latest activities in IETF's RMT and FECFRAME working groups. Such FEC improvements can also provide more efficient support of MBMS use cases.

The objective of this TR is to document the progress of the work item to investigate and evaluate proposed FEC technologies and, if appropriate, adopt one which provides the most significant enhancement to the performance of the MBMS system over the Rel-6 application layer FEC in MBMS. Aspects of system performance, which would provide benefit to the system, include, but are not limited to

- Improving the bandwidth efficiency of streaming and download services delivery over MBMS
- Improving the reliability of streaming and download services delivery over MBMS, e.g. by increasing the amount of tolerable lost packets for a given FEC overhead
- Reducing the required computational and memory resources for decoding in UEs
- Addressing backward compatibility issues by considering deployments of pre-Rel-11 MBMS FEC

The evaluation and selection process for the proposed improvements is documented in this TR.

This Technical report contains the following attachments:

- `Attachment-1-Tools.zip`: This attachment contains relevant tools for the purpose of setting up the evaluation framework.
- `Attachment-2-Benchmark-Codes.xls:` This attachment contains all relevant results for the benchmark codes
- `Attachment-3-Submission.zip:` This attachment contains all submitted code overhead and device-evaluation results of the candidates

- `Attachment-4-Verification.zip:` This attachment contains all verification results for device-based evaluation.
- `Attachment-5-6330.zip:` This attachment contains the proposed application of 6330 as MBMS application layer FEC as provided by the proponent.
- `Attachment-6-RS+LDPC.zip:` This attachment contains the proposed application of RS+LDPC as MBMS application layer FEC as provided by the proponent.

1 Scope

The present document documents the progress of the work item to investigate and evaluate proposed FEC technologies and adopt one which provides the most significant enhancement to the performance of the MBMS system over the Release 6 application layer FEC in MBMS.

2 References

The following documents contain provisions which, through reference in this text, constitute provisions of the present document.

- References are either specific (identified by date of publication, edition number, version number, etc.) or non-specific.
- For a specific reference, subsequent revisions do not apply.
- For a non-specific reference, the latest version applies. In the case of a reference to a 3GPP document (including a GSM document), a non-specific reference implicitly refers to the latest version of that document *in the same Release as the present document*.

[1] 3GPP TR 21.905: "Vocabulary for 3GPP Specifications".

[2] 3GPP TS 22.146: "Multimedia Broadcast/Multicast Service (MBMS); Stage 1".

[3] 3GPP TS 26.346: "Multimedia Broadcast/Multicast Service (MBMS); Protocols and codecs".

[4] IETF RFC 3926 (October 2004): "FLUTE - File Delivery over Unidirectional Transport", T. Paila, M. Luby, R. Lehtonen, V. Roca and R. Walsh.

[5] IETF RFC 5053 (February 2004): "Raptor Forward Error Correction Scheme for Object Delivery", M. Luby, M.Watson, A. Shokrollahi, and T. Stockhammer.

[6] IETF RFC 6363, "Forward Error Correction (FEC) Framework," M.Watson, A. Begen and V. Roca, October 2011.

[7] 3GPP TR 36.942, "Evolved Universal Terrestrial Radio Access (E-UTRA); Radio Frequency (RF) system scenarios."

3 Definitions, symbols and abbreviations

3.1 Definitions

For the purposes of the present document, the terms and definitions given in TR 21.905 [1] and the following apply. A term defined in the present document takes precedence over the definition of the same term, if any, in TR 21.905 [1].

Multimedia Broadcast/Multicast Service (MBMS): See 3GPP TS 22.146 [2].

MBMS user services: See 3GPP TS 22.146 [2].

MBMS delivery method: mechanism used by a MBMS user service to deliver content
There are two MBMS delivery method instances: download and streaming.

MBMS download delivery method: delivery of discrete objects (e.g. files) by means of a MBMS download session

MBMS streaming delivery method: delivery of continuous media (e.g. real-time video) by means of a MBMS streaming session

MBMS download session: time, protocols and protocol state (i.e. parameters) which define sender and receiver configuration for the download of content files

MBMS streaming session: time, protocols and protocol state (i.e. parameters) which define sender and receiver configuration for the streaming of content

3.2 Abbreviations

For the purposes of the present document, the abbreviations given in TR 21.905 [1] and the following apply. An abbreviation defined in the present document takes precedence over the definition of the same abbreviation, if any, in TR 21.905 [1].

ALC	Asynchronous Layered Coding
BM-SC	Broadcast-Multicast - Service Centre
ESI	Encoding Symbol ID
FDT	File Delivery Table
FEC	Forward Error Correction
FLUTE	File deLivery over Unidirectional Transport
IP	Internet Protocol
LCT	Layered Coding Transport
MBMS	Multimedia Broadcast/Multicast Service
PSS	Packet Switch Streaming
RTP	Real-Time Transport Protocol
SBN	Source Block Number
TOI	Transport Object Identifier
UDP	User Datagram Protocol

4 Use of FEC in MBMS

4.1 Introduction

Application Layer FEC is used in MBMS to compensate remaining losses on or below the IP layer in unidirectional delivery environments.

4.2 Architecture

Figure 1 depicts the MBMS network architecture showing MBMS related entities involved in providing MBMS user services as specified in TS 26.346 [3] with special focus to the FEC component. The FEC is included in the MBMS User Services which are part of the BM-SC on the network side and MBMS receiver on the UE side. FEC is specifically included in the File Delivery over Unidirectional Transport (FLUTE) [4] protocol and the Forward Error Correction (FEC) Framework (FECFRAME) [6] protocol.

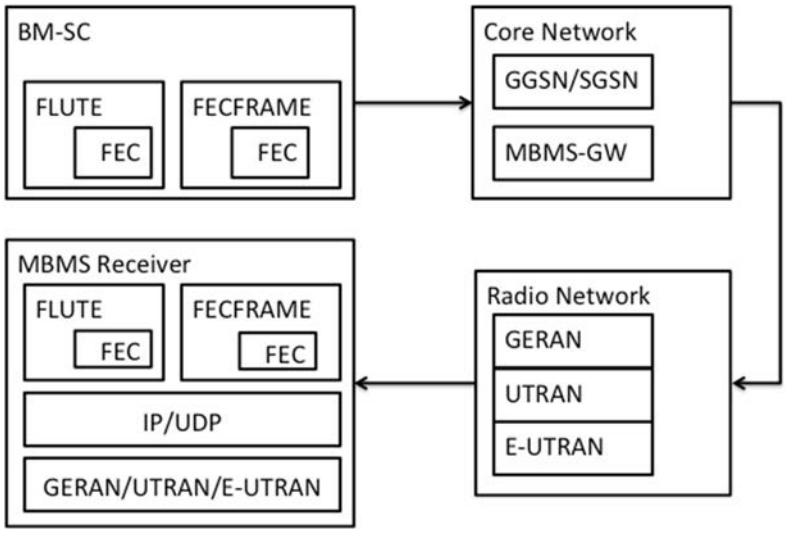


Figure 1: Overview on FEC operation in MBMS

4.3 MBMS Bearer in UTRAN

The MBMS UTRAN Bearer service reuses most of the legacy UMTS protocol stack in the packet-switched domain. Only minor modifications are introduced to support MBMS. The IP packets are processed in the Packet Data Convergence Protocol (PDCP) layer where for example header compression might be applied. In the Radio Link Control (RLC) the resulting PDCP-Protocol Data Units (PDUs), generally of arbitrary length, are mapped to fixed length RLC-PDUs. The RLC layer operates in unacknowledged mode as feedback links on the radio access network are not available for point-to-multipoint bearers. Functions provided at the RLC layer are for example segmentation and reassembly, concatenation, padding, sequence numbering, reordering and out-of-sequence and duplication detection. The Medium Access Control (MAC) layer maps and multiplexes the RLC-PDUs to the transport channel and selects the transport format depending on the instantaneous source rate. The MAC layer and physical layer appropriately adapt the RLC-PDU to the expected transmission conditions by applying, among others, channel coding, power and resource assignment, and modulation.

4.4 MBMS Bearer in E-UTRAN

The MBMS E-UTRAN Bearer service reuses most of the legacy LTE protocol stack in the packet-switched domain. Only minor modifications are introduced to support MBMS. The IP packets are processed in the Packet Data Convergence Protocol (PDCP) layer where for example header compression might be applied. In the Radio Link Control (RLC) the resulting PDCP-Protocol Data Units (PDUs), generally of arbitrary length, are mapped to fixed length RLC-PDUs. The RLC layer operates in unacknowledged mode as feedback links on the radio access network are not available for point-to-multipoint bearers. Functions provided at the RLC layer are for example segmentation and reassembly, concatenation, padding, sequence numbering, reordering and out-of-sequence and duplication detection. The Medium Access Control (MAC) layer maps and multiplexes the RLC-PDUs to the transport channel and selects the transport format depending on the instantaneous source rate. The MAC layer and physical layer appropriately adapt the RLC-PDU to the expected transmission conditions by applying, among others, channel coding, power and resource assignment, and modulation.

4.5 Streaming Delivery User Service

4.5.1 Introduction

The purpose of the MBMS streaming delivery method is to deliver continuous multimedia data (i.e. speech, audio, video and DIMS) over an MBMS bearer. The streaming delivery method is particularly useful for multicast and broadcast of scheduled streaming content. RTP is the transport protocol for MBMS streaming delivery. RTP provides means for sending real-time or streaming data over UDP.

TS 26.346 defines a generic mechanism for applying Forward Error Correction to streaming media. The mechanism consists of three components:

(i) construction of an FEC source block from the source media packets belonging to one or several UDP packet flows related to a particular segment of the stream(s) (in time). The UDP flows include RTP, RTCP, SRTP and MIKEY packets.

(ii) modification of source packets to indicate the position of the source data from the source packet within the source block

(iii) definition of repair packets, sent over UDP, which can be used by the FEC decoder to reconstruct missing portions of the source block.

The details on transport for the streaming delivery service are provided below.

An alternative way to deliver streaming services over MBMS is the use of DASH and FLUTE. This is use case is discussed in section 4.7.

4.5.2 Transport in streaming delivery service

The MBMS streaming framework operates on RTP packets or more precisely UDP payloads, incoming at same or different UDP ports. According to TS 26.346, clause 8.2.2, the FEC layer for streaming delivery is based on top of the UDP layer. The legacy RTP packets and the UDP port information are used in order to generate FEC repair symbols.

Original UDP payloads become FEC source packets by appending a 3 byte FEC source payload ID field at the end of each UDP payload. These packets are then UDP encapsulated and transported on the IP multicast bearer.

According to Figure 2 a copy of these packets is forwarded to the FEC encoder and is arranged in a source block with row width *T* bytes at the first empty row. The encoding symbol starts at the beginning of a new row, but it is preceded by a 3 byte field containing the UDP flow ID (1 byte) and the length field (2 bytes). In case the length of the packet is not an integer of the symbol the remaining bytes in the last row are filled up with zero bytes. The source block is filled up to *k* rows whereby *k* is flexible and can be changed dynamically for each source block. The selection of *k* depends on the desired delay, the available terminal memory and also might depend on aspects such as desired zapping time in mobile TV applications. Typically for a streaming service a *protection period* is defined and the value of the protection period dynamically determines the source block size.

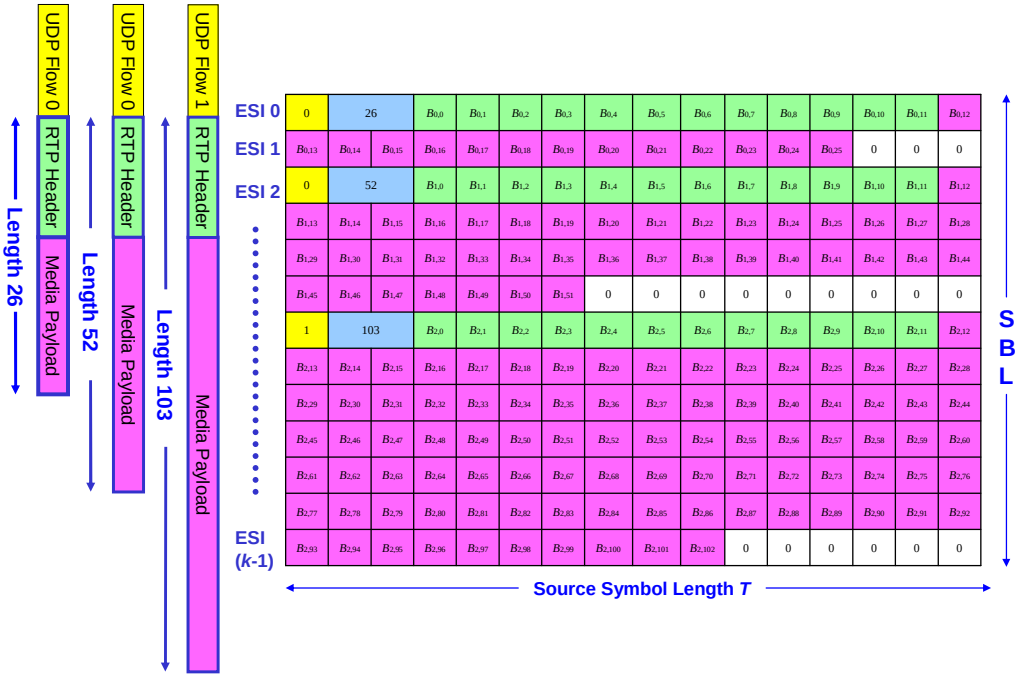


Figure 2: MBMS Streaming Framework

After processing all packets to be protected within one source block, the FEC encoder generates *n-k* FEC repair symbols of size *T* by applying FEC. The generated FEC repair symbols can be transmitted individually or as blocks of symbols as payload of a single UDP packet. Each FEC source and repair packet contains sufficient information such that the receiver can correctly insert them in the receiver source and repair block.

4.5.3 Examples

Examples are audio streaming applications or video streaming applications with bitrates ranging from 32 kbit/s to one or several MBit/s. The protection period is typically in the range of several seconds.

4.6 Download Delivery User Service

4.6.1 Introduction

According to TR 26.946, the MBMS Download Delivery Method allows the error-free transmission of files via the unidirectional MBMS Bearer Services. The files are "downloaded" and stored in the local files-system of the user equipment. Files may contain multimedia content or any other binary data. The MBMS Download Delivery Method allows the transmission of an arbitrary number of files within a single data transfer phase.

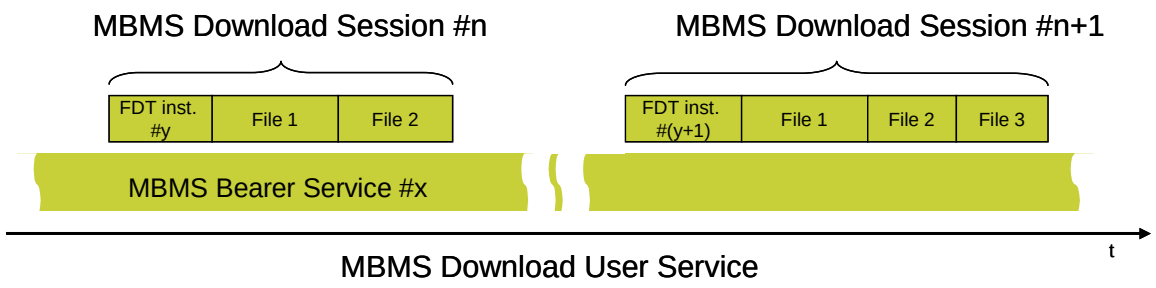


Figure 3: Definition of MBMS Download Sessions

Figure 3 is an example of an MBMS User Service based on the Download Delivery Method. The file transmission events are organized in MBMS Download Sessions. Each session is started with a File Delivery Table (FDT) instance, which describes in this example each file within the MBMS Download Session in terms of file name and file type (MIME Content Type). The service operator and the actual service determine the timing of MBMS Download Sessions. Depending on the service type, the MBMS Download session may require strict or more relaxed time-constraint delivery of content.

4.6.2 Transport in download delivery service

This clause explains briefly how files are constructed for and transported during a FLUTE session. The BM-SC takes a file, e.g. a video clip or a still image, which is used as the transport object for FLUTE (see figure 4). The BM-SC constructs source blocks by breaking the file into contiguous portions of approximately equal size. Each source block is broken into source symbols. One or more encoding symbols are carried as the payload of a FLUTE packet, thus the FLUTE packet size be divisible by the encoding symbol size. The target FLUTE packet size is configured by the BM-SC and, together with the file size, is used to determine the encoding symbol length. When FEC is used it may be beneficial to include several symbols in each FLUTE packet. Based on the transport object size, the encoding symbol size and the maximum source block length, FLUTE calculates the source block structure (i.e., the number of source blocks and their length).

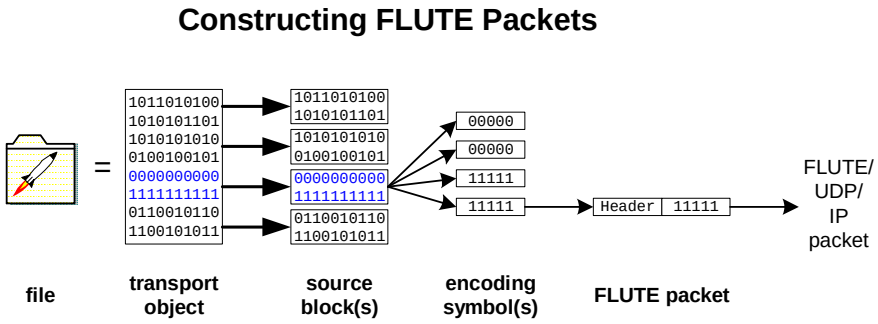


Figure 4: Constructing FLUTE packets

The BM-SC communicates the transport object size, the encoding symbol size and the file size to the receivers within the FLUTE session transmission such that the receiver can also calculate the source block structure in advance of receiving a file.

The FLUTE packet is constructed from FLUTE header and payload containing one or more encoding symbols.

The distinction between file and transport object is that the file is the object provided to the BM-SC and played-out or stored at the MBMS UE. Within the scope of FLUTE sessions, content encoding may be used, for instance to compress the file with gzip for delivery. In the presence of FLUTE session content encoding, the file and the transport object will be different binary objects, and in the absence of content encoding the transport object will be identical to the file. Any symbol calculations (including FEC) are performed on transport objects.

4.6.3 Download Examples

In a typical use case, multimedia files typically in 3GP or MP4 format are distributed through download delivery method. In this case the delivery rate and the media rate may be completely different as no real-time consumption is considered.

Table 1 shows some typical examples of file sizes for different types of multimedia content.

Table 1: Examples for Download delivery use cases

Number	File Size	Example
1	50 kByte (51 200 bytes)	JPEG coded logo
2	1 MByte (1 048 576 bytes)	AAC encoded audio clip
3	3 MByte (3 145 728 bytes)	MP3 audio clip
4	128 MByte (134 217 728) bytes	30 min SD movie coded at 500 kbit/s
5	1.8 GByte (1 887 436 800) bytes	2 hours HD movie coded at 2 MBit/s

4.7 Streaming using DASH and Download Delivery User Service

In another use case as indicated in TS 26.346, section 5.6, the download delivery method may be used to distribute DASH formatted content over MBMS. MBMS is designed to serve large receive groups with same content. The MBMS Download Delivery Method is designed to deliver an arbitrary number of (binary) files via MBMS to a large receiver population. MBMS Download defines several methods to increase reliability such as file repair. The download delivery method supports the delivery of media segments and even media presentation descriptions. Media segment URIs are described using the FDT in FLUTE.

In this case the media bit-rate and the delivery bitrate are typically the same to maintain real-time delivery capabilities and therefore the delivery delay of a segment is typically lower bounded by the segment duration.

Table 2 shows some typical examples of DASH media segment files for live services. In these examples, only one representation with constant media rate is being delivered over download delivery service.

Table 2: Examples for DASH segments

Number	Segment duration and media rate	FLUTE object (one segment) Size
1	1 sec DASH segment 250 kbit/s stream	32 kByte (32 768 bytes)
2	1 sec DASH segment for 1 Mbit/s stream	128 kByte (131 072 bytes)
3	2 sec DASH segment 250 kbit/s stream	64 kByte (65 536 bytes)
4	2 sec DASH segment for 1 Mbit/s stream	256 kByte (262 144 bytes)
5	4 sec DASH segment 250 kbit/s stream	128 kByte (131 072 bytes)
6	4 sec DASH segment for 1 Mbit/s stream	512 kByte (524 288 bytes)

5 MBMS Bearer Service Channel Modelling

5.1 Introduction

In order to investigate the performance of application layer FEC in the context of UTRAN and E-UTRAN, appropriate modelling of radio bearers is necessary.

5.2 Modelling of UTRAN MBMS Bearer

During the initial MBMS specification phase for Release-6, appropriate settings for UTRAN bearers for the simulation of FEC parameters had been defined and are summarized in Table 3.

Table 3: Typical UTRAN bearer parameters

UTRAN Bearer parameters		
	Bearer rates	64 kbit/s, 128 kbit/s, 256 kbit/s
	RLC PDU size	640 bytes, 1 280 bytes, 1 280 bytes respectively
	RLC BLER	1%, 5%, 10%, 15%, 20%, 30%
	RLC block loss pattern	Independent random loss

5.3 Modelling of E-UTRAN MBMS Bearer

To obtain some representative numbers for the performance of an FEC code in an LTE MBMS environment, some simple models are necessary for AL-FEC evaluation.

Figure 5 shows the mapping of RLC-SDUs to RLC-PDUs. RLC-SDUs in the context of MBMS are IP packets. The RLC header is 1 byte if the RLC SDU consists of 1 IP packet. The header is longer, if multiple IP packets are multiplexed in an RLC-SDU. A reasonable assumption is to use 3 byte header of the RLC-PDU assuming a 5 bit sequence number. The loss of one RLC-PDU results in the loss of all IP packets included in the RLC-PDU.

The MAC PDU consists of a number of MAC SDUs, where a MAC-SDUs is an RLC-PDU. The MAC multiplexer notifies the RLC layer of the available bits. The RLC layer would then create an RLC PDU that fits exactly into the available space in the MAC PDU. There is no need for fragmentation of MAC SDUs across subframes. Based on this, it can be assumed that the loss of one MAC-PDU results in the loss of one RLC-PDU.

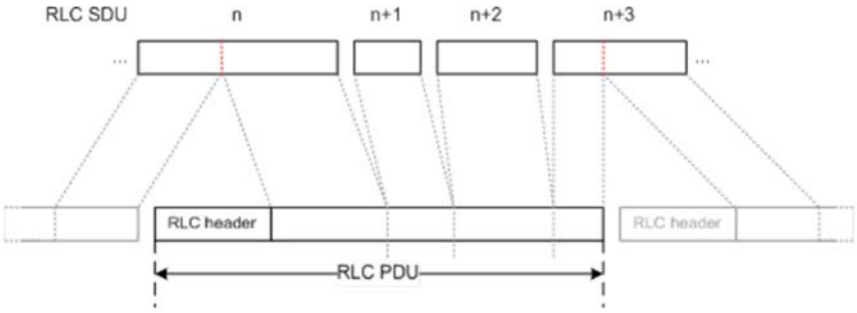


Figure 5: Mapping of IP packets (RLC-SDUs) to RLC-PDUs (see TS 36.300, section 6.2.2)

LTE MBMS defines modulations and coding schemes with a MAC-PDU size ranging from 680 bit to 18336 bit for a 5 MHz bandwidth.

Each MAC-PDU is mapped to a subframe. At allocation level 1, LTE MBMS can use up to 6 out of the 10 subframes of a 10ms frame. Each subframe is 1ms.

The interleaving for MBMS in LTE is the same as for regular unicast LTE delivery of 1 ms.

In communication with RAN1 and RAN2, it was agreed to use a two-state Markov model for the simulation of LTE RLC-PDU losses as shown in Figure 6:.

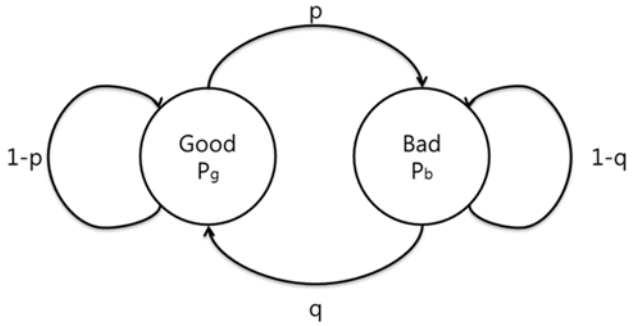


Figure 6: Markov model for LTE RLC-PDU losses

The model was parametrized based on the D1 simulation settings of 3GPP TR 36.942 [7] as reported in Table 4.

Table 4: Parameter Settings for MBMS LTE simulations

Parameter	*Setting*
Center Frequency (MHz)	2000
Cell radius (m)	288
Bandwidth (MHz)	5
Penetration Loss (dB)	20
Speed (km/h)	3
Antenna Down tilt (degree)	15
Antenna Height (m)	30
Antenna Clutter Height (m)	15
Dhb (m)	15
Slope	37.6
I	128.1
Average EIRP (dBW, 5MHz)	33
eNB Tx Power (dBW)	13
UE Antenna Loss (dB)	6
Implementation Loss (dB)	3
Noise Figure (dB)	6
Penetration Loss (dB)	20
Receiver Height (m)	1.5
Vertical Beamwidth (degree)	10
Horizontal Beamwidth (degree)	70

The simulation is carried out with a 19 sites configuration as shown in Figure 7:. Each site has 3 cells. All sites have 100% SFN operation. 30 UEs are uniformly dropped into the center site (dark green one) in each simulation run of 50 sec. In total 900 UEs are dropped and the SNR is sampled accordingly. The overall SNR distribution is also shown in Figure 7.

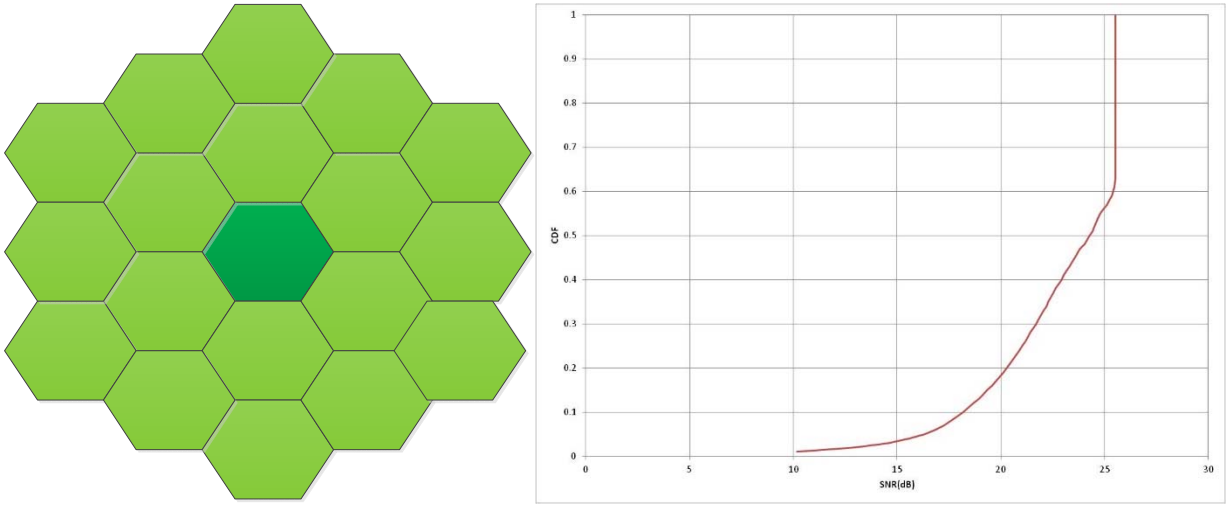


Figure 7: Simulation Grid and SNR distribution

Based on those SNR traces, two representative traces were selected that in combination with MCS24 result in a 1%, 5%, 10% and 20% target BLER.

The parametrization of the Markov model is as follows:

- each state persists for 10ms, and
- a state is good if it has:
  - less than 10% packet loss probability for the 1% and 5% BLER simulations,
  - less than 40% packet loss probability for the 10% and 20% BLER simulations.

- MCS=24 was used for all cases and then users at different 'MBMS geometry' were picked to get the different average error rate.

The parameters for Markov channel modelling are provided in Table 5.

Table 5: Markov channel parameters

Parameter	Meaning
p	transition probability from Good state to Bad state
q	transition probability from Bad state to Good state
p_g	BLER in Good state
p_b	BLER in Bad state
$\frac{1}{\mathrm{p}}$	Average Length of Bad state segment
$\frac{1}{\mathrm{q}}$	Average length of Good state segment

The time in a good state Tg or time in a bad state Tb may be computed by multiplying the average length of a good (bad) segment by the sampling period. The probability of the good state and probability of a bad state may be computed as q/(p+q) and p/(p+q), respectively.

Specifically, the following parameters for the LTE MBMS channel simulations:

- MCS=9 and MCS=21 with 498 byte RLC-SDU size and 1332 byte RLC-SDU size.
- RLC-SDU distance of 10ms and 40ms for MCS=21
- RLC-SDU distance of 10ms for MCS=9
- Channel model with Markov model loss rate of 1%, 5%, 10% and 20% target BLER as introduced in TDoc R1-120831, Annex B, Table 1 section 3.2 with speed 3 kph. The table is duplicated below as Table 6 with a resolution of an inconsistency in the average BLER.
- Channel model with Markov model loss rate of 1%, 5%, 10% and 20% target BLER as introduced in TDoc R1-120831, Annex B, Table 2 section 3.2 with speed 120kph. The table is duplicated below as Table 7.

Table 6: Markov parameters for 3 km/h

Table 1 3 km/h				
	BLER = 1%	**BLER = 5%**	**BLER = 10%**	**BLER = 20%**
p	0.58%	1.80%	2.79%	4.61%
q	36.13%	24.01%	20.90%	16.80%
sg	98.42%	93.02%	88.23%	78.48%
sb	1.58%	6.98%	11.77%	21.52%
pg	0.03%	0.06%	0.56%	1.16%
pb	59.47%	70.54%	82.30%	89.20%
BLER	0.97%	4.98%	10.19%	20.12%
Tg (ms)	1724	555	359	217
Tb (ms)	28	42	48	60

Table 7: Markov parameters for 120 km/h

Table 2 120 km/h				
	BLER = 1%	**BLER = 5%**	**BLER = 10%**	**BLER = 20%**
p	6.06%	27.07%	46.48%	35.60%
q	94.30%	70.95%	50.95%	63.29%
sg	93.97%	72.39%	52.29%	64.00%
sb	6.03%	27.61%	47.71%	36.00%
pg	0.00%	0.00%	0.00%	9.72%
pb	17.31%	19.54%	22.33%	40.40%
BLER	1.05%	5.40%	10.66%	20.77%
Tg (ms)	165	37	22	28
Tb (ms)	11	14	20	16

Regarding the MCS selection, the optimum operating MCS strongly depends on the deployment scenario, including site-to-site distance, operating frequency, interference conditions at MBSFN area boundaries, etc. Therefore, one specific value is not suitable. Using two different MCS cases can give some diversity in the assumptions, hence a good approach to use the following two values:

- higher value MCS=21 resulting in RLC-SDU size of 1332 byte.
- lower value corresponding to 1 bit/s/Hz, with MCS=9 resulting in RLC-SDU size of 498 byte.

It is additionally from the following list of available simulation conditions the following were selected as a good candidate representative:

- RLC-SDU distance of 10 ms and 40ms for MCS=21
- RLC-SDU distance of 10 ms for MCS=9

This results in total in 24 different channel configurations as summarized in Table 8.

Table 8: Typical LTE MBMS bearer parameters

LTE eMBMS Bearer			
	Bearer bitrates	398.4 kbit/s,	266.4, 1.0656 Mbit/s
	RLC-SDU size	498 byte	1332 byte
	RLC-SDU frequency	10ms	40ms, 10ms
	MAC PDU loss pattern	Markov	Markov
	Speed	3 and 120 km/h	3 and 120 km/h
	MAC-PDU loss probability	1%, 5%, 10%, 20%	1%, 5%, 10%, 20%

6 FEC Evaluation Procedure

6.1 Introduction

An Evaluation Procedure is defined for FEC evaluation and selection. This includes procedures to measure theoretical FEC code performance, FEC performance in 3GPP services as well as high-level and detailed decoder performance.

6.2 Simulation Conditions

6.2.1 Simulation conditions and assumptions (UTRAN)

The simulation conditions for UTRAN-based MBMS are provided in Table 9.

Additional details on the simulation methodology are provided in Annex A and should be viewed as simulation guidelines in case there are any ambiguities.

Table 9: Simulation Conditions for UTRAN-based MBMS

UTRAN Download		
	Bearer rates	64 kbit/s, 128 kbit/s, 256 kbit/s
	RLC-PDU size	640 bytes, 1 280 bytes, 1 280 bytes respectively
	RLC-PDU BLER	1%, 5%, 10%, 15%, 20%, 30%
	RLC-PDU block loss pattern	Independent random loss
	Number of trials/users	At least 10,000 for files ≤ 512 KB, 3,000 for 3 072 KB
	File sizes	50 KB, 512 KB, 3 072 KB
	FLUTE payload size	456 bytes
	ROHC	No
	IPv4/UDP header	28 bytes
	FLUTE header	16 bytes
	FEC overhead	Varied in steps of X packets, where X=ceil(0.005N) and N is the number of packets containing source data
UTRAN Streaming		
	Bearer rates	64 kbit/s, 128 kbit/s and 256 kbit/s
	RLC PDU size	640 bytes (for 64 kbit/s bearer) 1280 bytes (for 128 kbit/s bearer) 1280 bytes (for 256 kbit/s bearer)
	RLC BLER	1 %, 5 %, 10 %, 15 %, 20 %, 30 %
	RLC block loss pattern	Independent random loss
	Content length	24 hours of media content
	Media rates	Varied by steps of 1 % of bearer rate, assuming only a single media stream with constant bitrate (see note 1)
	FEC overhead	Varied to sum FEC and Media to equal bearer rate
	Source packet RTP payload size	64 kbit/s: 456 bytes 128 kbit/s: 456 bytes 256 kbit/s: 768 bytes
	Repair packet RTP payload size	Minimum value supported by the FEC code which is not less than 470 (for 64 kbit/s and 128 kbit/s) and 782 (for 256 kbit/s) - (see note 2)
	Protection period	5 s, 20 s
	ROHC	No
	IPv4/UDP/RTP header	40
NOTE 1: In practice, multiple media streams may be carried within a single MBMS bearer. However, only a single media stream is considered for FEC simulation purposes for simplicity. NOTE 2: The last repair packet of a block may be shorter if supported by the FEC code in order to fit within the protection period.		

6.2.2 Simulation conditions and assumptions (LTE eMBMS)

The simulation conditions for LTE-based MBMS are provided in Table 10.

Additional details on the simulation methodology are provided in Annex A and should be used as guidelines for simulations.

Table 10: Simulation Conditions for LTE-based MBMS

LTE eMBMS Download		
	RLC-SDU	266.4 kbit/s, 398.4 kbit/s, 1.0656Mbit/s
	RLC-SDU size	498, 1332 byte
	Loss Model	Markov
	MCS	9, 21
	RLC-SDU period	40 ms, 10 ms
	Speed	3 km/h, 120 km/h
	MAC-PDU loss probability	1%, 5%, 10%, 20%
	Number of trials/users	At least 10,000 for files ≤ 1 MB, at least 3,000 otherwise
	File sizes	50kB, 1MB, 3MB, 128MB, 1.8GB
	FLUTE payload size	(RLC-PDU size - 44) bytes
	ROHC	No
	IPv4/UDP header	28 bytes
	FLUTE header	16 bytes
	FEC overhead	Varied in steps of X packets, where X=ceil(0.005N) and N is the number of packets containing source data
LTE eMBMS Streaming (based on DASH)		
	Bearer rates	266.4 kbit/s, 398.4 kbit/s, 1.0656Mbit/s
	RLC-SDU size	498, 1332 byte
	Loss Model	Markov
	MCS	9, 21
	RLC-SDU period	40 ms, 10 ms
	Speed	3 km/h, 120 km/h
	MAC-PDU loss probability	1%, 5%, 10%, 20%
	Content length	24 hours of media content
	Media rates	Varied by steps of FLUTE payload sizes, but constant
	FEC overhead	Varied to sum FEC and Media to equal bearer rate
	FLUTE payload size	(RLC-PDU size - 44) bytes
	Media Segment duration	1 s, 4 s
	Segment to FLUTE object mapping	Each Segment is mapped to one FLUTE object
	Maximum delivery delay of FLUTE object	media segment duration
	ROHC	No
	IPv4/UDP/FLUTE header	44

6.3 Code Performance

6.3.1 Introduction

For the evaluation of the code performance, two different methods are defined.

6.3.2 Method 1

6.3.2.1 Evaluation Procedure

Data to be transmitted is partitioned into K symbols. These K symbols are used to generate N total symbols to be transmitted, where $N>=K$. The N symbols are transmitted through an erasure channel with erasure probability P_e (on the FEC symbol level). The erasure channel is IID and it operates on the data symbol by symbol. The IID erasure channel is illustrated in Figure 8 and Figure 9. Successful decoding requires at least K symbols to be received, but in some cases additional received symbols may be necessary. Denote the number of symbols *received* in excess of K to be O. The decoding failure probability distribution is a function of O and is given as $P(O)$=Pr{decoding with O overhead symbols or less fails}.

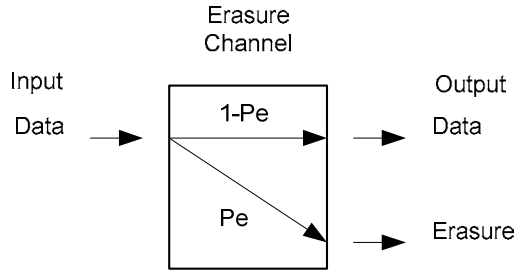


Figure 8: Illustration of the IID erasure channel. Data is passed through the channel with probability 1-Pe, and erased with probability Pe.

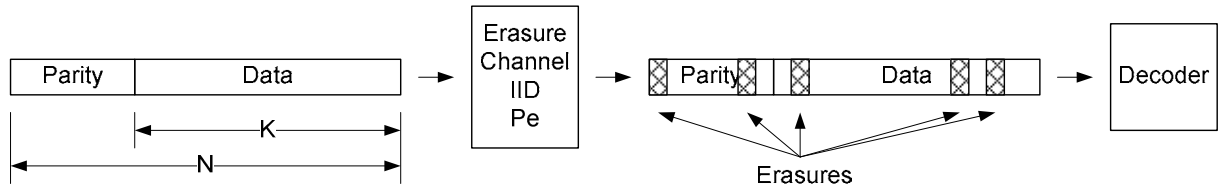


Figure 9: Data is passed through the IID erasure channel, with erasure probability of Pe. Data is delivered to the decoder in the order in which it was transmitted.

To obtain the distribution $P(O)$ a statistical evaluation procedure is proposed as follows:

1) Fix K, the number of encoded symbols
2) Fix N, the maximum number of symbols (systematic or repair) to be transmitted
3) Use an Erasure Channel with probability of error Pe for each symbol.
4) Loop over 5 to 10 for N_iterations=10,000
5) Set O= -1 and TX= -1
6) Set RX=0
7) While (RX < K)
    a. If (TX+1>N)
        i. Note the case as "undecodable"
        ii. Goto 5
    b. TX=TX+1
    c. Transmit a symbol through the Erasure Channel. If the symbol is delivered by the Erasure Channel
        i. RX = RX + 1
8) Attempt to Decode with the received symbols
9) If decoding is not successful
    a. If(TX+1>N)
        i. Note O and that the case was "undecodable"
        ii. Goto 5
    b. TX = TX+1

c. Transmit a symbol through the Erasure Channel. If the symbol is delivered by the Erasure Channel

i. O=O+1

d. Goto 8

10) If decoding is successful

a. Note O

b. Goto 5

6.3.2.2 Test Cases

The following test cases are determined for the purpose of evaluating the code performance.

Table 11: Test Cases for Code Performance

Number	*K*	*N*	Channel
CP1	32	39	IID P_e=5%
CP2	128	154	IID P_e=5%
CP3	256	282	IID P_e=5%
CP4	1024	1127	IID P_e=5%
CP5	8192	9012	IID P_e=5%
CP6	32	45	IID P_e=10%
CP7	128	180	IID P_e=10%
CP8	256	308	IID P_e=10%
CP9	1024	1229	IID P_e=10%
CP10	8192	9831	IID P_e=10%

6.3.2.3 Performance Metrics

For each of the above test cases the following performance metrics are reported for N_iterations=10,000:

- The probability that decoding is not successful with O = i symbols, P(O=i), where i=[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
- The probability that decoding is not successful P(undecodable).
- The probability that decoding is not successful with less than O=0 symbols $P_f(O=0)$,
- The necessary overhead O^* to achieve $P_f(O=O^*) <= 0.5$, O (P_f=0.5)
- The necessary overhead O^* to achieve $P_f(O=O^*) <= 0.1$, O (P_f=1e-1)
- The necessary overhead O^* to achieve $P_f(O=O^*) <= 0.01$, O (P_f=1e-2)
- The necessary overhead O^* to achieve $P_f(O=O^*) <= 0.001$, O (P_f=1e-3)
- The necessary overhead O^* to achieve $P_f(O=O^*) <=$ 1e-4, O (P_f=1e-4)
- The necessary overhead O^* to achieve $P_f(O=O^*) <=$ 1e-5, O (P_f=1e-5)
- The average symbol overhead $E\{O\}$ for the test case.

Table 12: Reporting format for Code Performance Method 1

Case	$P_f(O=0)$	O (P_f=0.5)	O (P_f=1e-1)	O (P_f=1e-2)	O (P_f=1e-3)	O (P_f=1e-4)	O (P_f=1e-5)	E{O}
CP1								
CP2								
CP3								
CP4								
CP5								
CP6								
CP7								
CP8								
CP9								
CP10								

Case	$P(O=0)$	$P(O=1)$	$P(O=2)$	$P(O=3)$	$P(O=4)$	$P(O=5)$	$P(O=6)$	$P(O=7)$	$P(O=8)$	$P(O=9)$
CP1										
CP2										
CP3										
CP4										
CP5										
CP6										
CP7										
CP8										
CP9										
CP10										

6.3.3 Method 2

6.3.3.1 Evaluation Procedure

The distribution of the code overhead O for different permutations of received symbols is a relevant measure for the code performance. Specifically, the failure probability distribution defined as $P_f(O)$= Pr{decoding with exactly O overhead symbols fails} is relevant and may be used to determine the code performance.

To obtain the distribution $P_f(O)$ a statistical evaluation procedure is proposed based on the following four parameters:

- the source block size K providing the total number of source symbols
- the maximum encoding symbol ID (ESI) N for any repair symbol

Given these numbers the following procedure is proposed to obtain the O for one experiment:

1. Generate a source block with K symbols
2. Generate N-K repair symbols with ESI=K+1, ..., N
3. Randomly pick K among the N symbols
4. Set O to 0
5. Attempt decoding using the available K+O encoding symbols. The symbols are ordered in sequence for decoding.
6. If decoding is not successful then
    a. pick one additional not yet included encoding symbol randomly chosen from the N symbols.
    b. Set O to O+1,
    c. If K+O == N+1 then goto 7, else goto 5
7. Report O as the overhead result for this experiment

To obtain the distribution for the necessary overhead O at least 10,000 of the above experiments are carried out.

6.3.3.2 Test Cases

The following test cases are determined for the purpose of evaluating the code performance.

Table 13: Test Cases for Code Performance

Number	*K*	*N*
CP11	32	34
CP12	32	38
CP13	32	128
CP14	256	269
CP15	256	307
CP16	256	1024
CP17	1024	1075
CP18	1024	1229
CP19	1024	3072
CP20	8192	8601
CP21	8192	9830
CP22	8192	30000

Notes that a code does not necessarily have to provide N different encoding symbols, but the code may have less symbols N'. To use such codes in an environment were N symbols are required, the code with $N'<N$ independent encoding symbols may repeat encoding symbols to generate N symbols in total.

6.3.3.3 Performance Metrics

For the above test cases CP11-CP22 the following performance metrics are reported for at least N_iterations=10,000:

- The probability that decoding is not successful with $O = i$ symbols, P(O=i), where i=[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
- The probability that decoding is not successful with less than O=0 symbols $P_f(O=0)$,
- The necessary overhead O^* to achieve $P_f(O=O^*) <= 0.5$, O (P_f=0.5)
- The necessary overhead O^* to achieve $P_f(O=O^*) <= 0.1$, O (P_f=1e-1)
- The necessary overhead O^* to achieve $P_f(O=O^*) <= 0.01$, O (P_f=1e-2)
- The necessary overhead O^* to achieve $P_f(O=O^*) <= 0.001$, O (P_f=1e-3)
- The necessary overhead O^* to achieve $P_f(O=O^*) <= 1e-4$, O (P_f=1e-4)
- The necessary overhead O^* to achieve $P_f(O=O^*) <= 1e-5$, O (P_f=1e-5)
- The average symbol overhead E{O} for the test case.

Table 14: Reporting format for Code Performance Method 2

Case	$P_f(O=0)$	O (P_f=0.5)	O (P_f=1e-1)	O (P_f=1e-2)	O (P_f=1e-3)	O (P_f=1e-4)	O (P_f=1e-5)	E{O}
CP11								
CP12								
CP13								
CP14								
CP15								
CP16								
CP17								
CP18								
CP19								
CP20								
CP21								
CP22								

Case	*P*(*O*=0)	*P*(*O*=1)	*P*(*O*=2)	*P*(*O*=3)	*P*(*O*=4)	*P*(*O*=5)	*P*(*O*=6)	*P*(*O*=7)	*P*(*O*=8)	*P*(*O*=9)
CP11										
CP12										
CP13										
CP14										
CP15										
CP16										
CP17										
CP18										
CP19										
CP20										
CP21										
CP22										

6.4 Download Performance

6.4.1 Performance Metrics

For download delivery, the FEC Overhead required to achieve 99% probability of recovery of the file provides a very good indication for the system level performance.

The FEC Overhead required for 99 % probability of recovery is computed the Transmission overhead as described in Annex A.1 of this document.

In addition, the following parameters are be reported:

- The symbol size, T, in bytes
- The total number of symbols required to represent the source data of the object, Kt
- The number of source blocks, Z
- The number of sub-blocks in each source block, Ns
- The maximum number of symbols to be transported in a single packet, G

For details refer to RFC 3926 [4] and RFC 5053 [5].

6.4.2 Download Performance over UTRAN

Table 15 provides a reporting format for UTRAN test cases.

Table 15: FEC Overhead required for 99 % probability for UTRAN download test cases

Test Case	Error conditions	File size	Bitrate kbit/s	Fec Overhead	[T; Kt; Z; Ns; G]
UD1	Low (1% BLER)	Small (50KB)	64		
UD2		Medium (512KB)	64		
UD3		Large (3072KB)	64		
UD4	Medium (5% BLER)	Small (50KB)	64		
UD5		Medium (512KB)	64		
UD6		Large (3072KB)	64		
UD7	High (10% BLER)	Small (50KB)	64		
UD8		Medium (512KB)	64		
UD9		Large (3072KB)	64		
UD10	15% BLER	Small (50KB)	64		
UD11		Medium (512KB)	64		
UD12		Large (3072KB)	64		
UD13	20% BLER	Small (50KB)	64		
UD14		Medium (512KB)	64		
UD15		Large (3072KB)	64		
UD16	30% BLER	Small (50KB)	64		
UD17		Medium (512KB)	64		
UD18		Large (3072KB)	64		
UD19	Low (1% BLER)	Small (50KB)	128/256		
UD20		Medium (512KB)	128/256		
UD21		Large (3072KB)	128/256		
UD22	Medium (5% BLER)	Small (50KB)	128/256		
UD23		Medium (512KB)	128/256		
UD24		Large (3072KB)	128/256		
UD25	High (10% BLER)	Small (50KB)	128/256		
UD26		Medium (512KB)	128/256		
UD27		Large (3072KB)	128/256		
UD28	15% BLER	Small (50KB)	128/256		
UD29		Medium (512KB)	128/256		
UD30		Large (3072KB)	128/256		
UD31	20% BLER	Small (50KB)	128/256		
UD32		Medium (512KB)	128/256		
UD33		Large (3072KB)	128/256		
UD34	30% BLER	Small (50KB)	128/256		
UD35		Medium (512KB)	128/256		
UD36		Large (3072KB)	128/256		

6.4.3 Download Performance over LTE

Table 16 provides a reporting format for LTE test cases.

Table 16: FEC Overhead required for 99 % probability for LTE download delivery test cases

Test Case	Error conditions	Bitrate kbit/s	File size	FEC Overhead	[T; Kt; Z; Ns; G]
LD1	Markov, 3 km/h, 1%	266.4	50 kB		
LD2		266.4	Audio (1 MB)		
LD3		266.4	Clip(3 MB)		
LD4		266.4	SD (128 MB)		
LD5		266.4	HD(1.8 GB)		
LD6	Markov, 3 km/h, 5%	266.4	50 kB		
LD7		266.4	Audio (1 MB)		
LD8		266.4	Clip(3 MB)		
LD9		266.4	SD (128 MB)		
LD10		266.4	HD(1.8 GB)		
LD11	Markov, 3 km/h, 10%	266.4	50 kB		
LD12		266.4	Audio (1 MB)		
LD13		266.4	Clip(3 MB)		
LD14		266.4	SD (128 MB)		
LD15		266.4	HD(1.8 GB)		
LD16	Markov, 3 km/h, 20%	266.4	50 kB		
LD17		266.4	Audio (1 MB)		
LD18		266.4	Clip(3 MB)		
LD19		266.4	SD (128 MB)		
LD20		266.4	HD(1.8 GB)		
LD21	Markov, 3 km/h, 1%	398.4	50 kB		
LD22		398.4	Audio (1 MB)		
LD23		398.4	Clip(3 MB)		
LD24		398.4	SD (128 MB)		
LD25		398.4	HD(1.8 GB)		
LD26	Markov, 3 km/h, 5%	398.4	50 kB		
LD27		398.4	Audio (1 MB)		
LD28		398.4	Clip(3 MB)		
LD29		398.4	SD (128 MB)		
LD30		398.4	HD(1.8 GB)		
LD31	Markov, 3 km/h, 10%	398.4	50 kB		
LD32		398.4	Audio (1 MB)		
LD33		398.4	Clip(3 MB)		
LD34		398.4	SD (128 MB)		
LD35		398.4	HD(1.8 GB)		
LD36	Markov, 3 km/h, 20%	398.4	50 kB		
LD37		398.4	Audio (1 MB)		
LD38		398.4	Clip(3 MB)		
LD39		398.4	SD (128 MB)		
LD40		398.4	HD(1.8 GB)		
LD41	Markov, 3 km/h, 1%	1065.6	50 kB		
LD42		1065.6	Audio (1 MB)		
LD43		1065.6	Clip(3 MB)		
LD44		1065.6	SD (128 MB)		
LD45		1065.6	HD(1.8 GB)		
LD46	Markov, 3 km/h, 5%	1065.6	50 kB		
LD47		1065.6	Audio (1 MB)		
LD48		1065.6	Clip(3 MB)		
LD49		1065.6	SD (128 MB)		
LD50		1065.6	HD(1.8 GB)		
LD51	Markov, 3 km/h, 10%	1065.6	50 kB		
LD52		1065.6	Audio (1 MB)		
LD53		1065.6	Clip(3 MB)		
LD54		1065.6	SD (128 MB)		
LD55		1065.6	HD(1.8 GB)		
LD56	Markov, 3 km/h, 20%	1065.6	50 kB		
LD57		1065.6	Audio (1 MB)		
LD58		1065.6	Clip(3 MB)		
LD59		1065.6	SD (128 MB)		
LD60		1065.6	HD(1.8 GB)		
LD61	Markov, 120 km/h, 1%	266.4	50 kB		
LD62		266.4	Audio (1 MB)		
LD63		266.4	Clip(3 MB)		

LD64		266.4	SD (128 MB)		
LD65		266.4	HD(1.8 GB)		
LD66	Markov, 120 km/h, 5%	266.4	50 kB		
LD67		266.4	Audio (1 MB)		
LD68		266.4	Clip(3 MB)		
LD69		266.4	SD (128 MB)		
LD10		266.4	HD(1.8 GB)		
LD71	Markov, 120 km/h, 10%	266.4	50 kB		
LD72		266.4	Audio (1 MB)		
LD73		266.4	Clip(3 MB)		
LD74		266.4	SD (128 MB)		
LD75		266.4	HD(1.8 GB)		
LD76	Markov, 120 km/h, 20%	266.4	50 kB		
LD77		266.4	Audio (1 MB)		
LD78		266.4	Clip(3 MB)		
LD79		266.4	SD (128 MB)		
LD80		266.4	HD(1.8 GB)		
LD81	Markov, 120 km/h, 1%	398.4	50 kB		
LD82		398.4	Audio (1 MB)		
LD83		398.4	Clip(3 MB)		
LD84		398.4	SD (128 MB)		
LD85		398.4	HD(1.8 GB)		
LD86	Markov, 120 km/h, 5%	398.4	50 kB		
LD87		398.4	Audio (1 MB)		
LD88		398.4	Clip(3 MB)		
LD89		398.4	SD (128 MB)		
LD90		398.4	HD(1.8 GB)		
LD91	Markov, 120 km/h, 10%	398.4	50 kB		
LD92		398.4	Audio (1 MB)		
LD93		398.4	Clip(3 MB)		
LD94		398.4	SD (128 MB)		
LD95		398.4	HD(1.8 GB)		
LD96	Markov, 120 km/h, 20%	398.4	50 kB		
LD97		398.4	Audio (1 MB)		
LD98		398.4	Clip(3 MB)		
LD99		398.4	SD (128 MB)		
LD100		398.4	HD(1.8 GB)		
LD101	Markov, 120 km/h, 1%	1065.6	50 kB		
LD102		1065.6	Audio (1 MB)		
LD103		1065.6	Clip(3 MB)		
LD104		1065.6	SD (128 MB)		
LD105		1065.6	HD(1.8 GB)		
LD106	Markov, 120 km/h, 5%	1065.6	50 kB		
LD107		1065.6	Audio (1 MB)		
LD108		1065.6	Clip(3 MB)		
LD109		1065.6	SD (128 MB)		
LD110		1065.6	HD(1.8 GB)		
LD111	Markov, 120 km/h, 10%	1065.6	50 kB		
LD112		1065.6	Audio (1 MB)		
LD113		1065.6	Clip(3 MB)		
LD114		1065.6	SD (128 MB)		
LD115		1065.6	HD(1.8 GB)		
LD116	Markov, 120 km/h, 20%	1065.6	50 kB		
LD117		1065.6	Audio (1 MB)		
LD118		1065.6	Clip(3 MB)		
LD119		1065.6	SD (128 MB)		
LD120		1065.6	HD(1.8 GB)		

6.5 UTRAN Streaming Performance

For RTP-based streaming delivery, as a suitable measure it was considered to evaluate the maximum supported Media Rate (kbit/s) for Mean Time Between FEC Block Loss of 1 hour.

For streaming services simulation we assume the following:

- All source RTP packets and UDP repair packets have the same total SDU size (500 bytes for 64/128 kbit/s, 800 bytes for 256 kbit/s) and number of symbols *G*: this is not exactly true, but it is considered sufficient FEC code evaluation.

- Receiver working memory is large enough to decode the highest bitrate with the longest protection period.

- Total bitrate of source data plus repair is always matched to the bearer rate. Consequently the SDU loss transcript is always the same for a given stream duration and fixed SDU size, only amount of repair and the associated maximum possible streaming rate are changing.

Results following the mode as provided in Annex A of TR 26.946 are expected.

The simulation conditions as provided in Annex A.2 of the present document are be applied.

In addition, the following parameters are reported:

- The symbol size, T, in bytes
- The total number of symbols within a protection period, N'
- The number of symbols per packet, G
- The source block size K

The stream total duration is 24 hours and target Mean Time Between Failure (MTBF) is set to 1 block error per hours. In addition, the MBTF over the source block rate may be reported as well. This translates into a maximum of 24 errors over a 24 hour period.

Table 17 provides a reporting format for UTRAN streaming test cases.

Table 17: Maximum supported Media Rate (kbit/s)
for Mean Time Between FEC Block Loss of 1 hour for UTRAN streaming test cases

Test Case	Error rates	Bearer rate	Protection Period	Performance	[T; N'; G; K]
US1	Low (1% BLER)	Low (64 kbit/s)	5 sec		
US2			20 sec		
US3		Medium (128 kbit/s)	5 sec		
US4			20 sec		
US5		High (256 kbit/s)	5 sec		
US6			20 sec		
US7	Medium (5% BLER)	Low (64 kbit/s)	5 sec		
US8			20 sec		
US9		Medium (128 kbit/s)	5 sec		
US10			20 sec		
US11		High (256 kbit/s)	5 sec		
US12			20 sec		
US13	High (10% BLER)	Low (64 kbit/s)	5 sec		
US14			20 sec		
US15		Medium (128 kbit/s)	5 sec		
US16			20 sec		
US17		High (256 kbit/s)	5 sec		
US18			20 sec		

6.6 Streaming Performance over LTE

For DASH-based streaming delivery, as a similarly suitable measure it is considered to evaluate the media rate to support a Mean Time Between FEC Block Loss of 1 hour.

Test cases are considered for 1, 2 and 4 seconds segment duration as well as bearer bitrates of 260 kbit/s and 1 MBit/s.

The simulation conditions as provided in Annex A.2 of this document are applied.

In addition, the following parameters are reported:

- The symbol size, T, in bytes
- The total number of symbols within a protection period, N'
- The number of symbols per packet, G

- The source block size K

The stream total duration is 24 hours and target Mean Time Between Failure (MTBF) is set to 1 block error per hours. In addition, the MBTF over the source block rate may be reported as well. This translates into a maximum of 24 errors over a 24 hour period.

Table 18 provides a reporting format for LTE streaming test cases.

Table 18: Media Bitrate in kbit/s
for Mean Time Between FEC Block Loss of 1 hour for LTE use cases

Test Case	Error conditions	Segment Duration in seconds	Bearer Bitrate kbit/s	Supported Media Bitrate	[T; K; N'; G]
LS1	Markov, 3 km/h, 1%	1	266.4		
LS2		1	398.4		
LS3		1	1065.6		
LS4		4	266.4		
LS5		4	398.4		
LS6		4	1065.6		
LS7	Markov, 3 km/h, 5%	1	266.4		
LS8		1	398.4		
LS9		1	1065.6		
LS10		4	266.4		
LS11		4	398.4		
LS12		4	1065.6		
LS13	Markov, 3 km/h, 10%	1	266.4		
LS14		1	398.4		
LS15		1	1065.6		
LS16		4	266.4		
LS17		4	398.4		
LS18		4	1065.6		
LS19	Markov, 3 km/h, 20%	1	266.4		
LS20		1	398.4		
LS21		1	1065.6		
LS22		4	266.4		
LS23		4	398.4		
LS24		4	1065.6		
LS25	Markov, 120 km/h, 1%	1	266.4		
LS26		1	398.4		
LS27		1	1065.6		
LS28		4	266.4		
LS29		4	398.4		
LS30		4	1065.6		
LS31	Markov, 120 km/h, 5%	1	266.4		
LS32		1	398.4		
LS33		1	1065.6		
LS34		4	266.4		
LS35		4	398.4		
LS36		4	1065.6		
LS37	Markov, 120 km/h, 10%	1	266.4		
LS38		1	398.4		
LS39		1	1065.6		
LS40		4	266.4		
LS41		4	398.4		
LS42		4	1065.6		
LS43	Markov, 120 km/h, 20%	1	266.4		
LS44		1	398.4		
LS45		1	1065.6		
LS46		4	266.4		
LS47		4	398.4		
LS48		4	1065.6		
LS52	Markov, 3 km/h, 1%	2	266.4		
LS53		2	398.4		
LS54		2	1065.6		
LS55	Markov, 3 km/h, 5%	2	266.4		
LS56		2	398.4		
LS57		2	1065.6		
LS58	Markov, 3 km/h, 10%	2	266.4		
LS59		2	398.4		
LS60		2	1065.6		
LS61	Markov, 3 km/h, 20%	2	266.4		
LS62		2	398.4		
LS49		2	1065.6		
LS63	Markov, 120 km/h, 1%	2	266.4		
LS64		2	398.4		

Test Case	Error conditions	Segment Duration in seconds	Bearer Bitrate kbit/s	Supported Media Bitrate	[T; K; N'; G]
LS65		2	1065.6		
LS66	Markov, 120 km/h, 5%	2	266.4		
LS67		2	398.4		
LS50		2	1065.6		
LS68	Markov, 120 km/h, 10%	2	266.4		
LS69		2	398.4		
LS70		2	1065.6		
LS71	Markov, 120 km/h, 20%	2	266.4		
LS72		2	398.4		
LS51		2	1065.6		

6.6a Implementation-specific Performance Metrics

Codes not only differ in terms of the code efficiency but also in other performance criteria. Two important aspects are the required memory for decoding in the MBMS client as well as the complexity of the considered decoding algorithm.

Therefore, to judge the *complexity* of a decoding algorithm, the decoding speed in terms of bit/s on top of a recognized mobile processor platform running a recognized mobile operating system can provide good insight into the feasibility of executing the code for mobile applications.

Another important aspect is the global latency of the global system (i.e. From the video making to the video rendering on the device). Thus, encoding complexity is also considered. Therefore, to judge the *complexity* of an encoding algorithm, the encoding speed in terms of bit/s on top of a recognized PC platform running a recognized PC operating system can provide good insight into the impact of the encoding on the global latency.

In terms of *memory requirements*, a reasonable measure is the required random access memory in the MBMS client to decode large files, such as considered in the video delivery use cases from above.

Another performance metric for successful integration into mobile platforms is the *library footprint* of the code and the footprint of hardware functions if any.

The complexity and memory requirements is in particular provided for the following use cases:

- 1.8 GByte at 20% Markov model error rate;
- 4sec @ 1MBit/s streaming at the 20% Markov model error rate;
- 20sec protection period for RTP based streaming at 384 kbit/s and the 20% error rate.

6.7 Device-based Complexity Evaluation

6.7.1 Introduction

This clause provides a test plan for testing device-based evaluation. For all definitions and acronyms here that are not explicitly made in this document, please see TS 26.346 [3] (e.g. for OTI, FDT, FLUTE).

6.7.2 Test Cases

The following use cases are considered for performing (for details refer to clause 6.2).

LTE Download Delivery

Note that the file size are as follows:

- Clip: 3 * 1024 * 1024 Byte = 3145728 Bytes,
- SD: 128 * 1024 * 1024 Byte = 134217728 Bytes,
- HD: 1800 * 1024 * 1024 Byte = 1887436800 Bytes.

Test Case	Error conditions	Bitrate kbit/s	File	File size (in bytes)	Repetition
LD60	Markov, 3 km/h, 20%	1065.6	HD	1887436800	1
LD108	Markov, 120 km/h, 5%	1065.6	Clip	3145728	20
LD109		1065.6	SD	134217728	5
LD110		1065.6	HD	1887436800	1
LD118	Markov, 120 km/h, 20%	1065.6	Clip	3145728	20
LD119		1065.6	SD	134217728	5

DASH-based Streaming Delivery over LTE

Test Case	Error conditions	Segment Duration in seconds	Bearer Bitrate kbit/s	Duration in seconds
LS21	Markov, 3 km/h, 20%	1	1065.6	1800
LS49		2	1065.6	1800
LS24		4	1065.6	1800
LS33	Markov, 120 km/h, 5%	1	1065.6	1800
LS50		2	1065.6	1800
LS36		4	1065.6	1800
LS45	Markov, 120 km/h, 20%	1	1065.6	1800
LS51		2	1065.6	1800
LS48		4	1065.6	1800

6.7.3 Test Conditions & Test Procedure

6.7.3.1 Overview Test Platform and Operation Conditions

Figure 10 shows the considered test platform that is to be used.

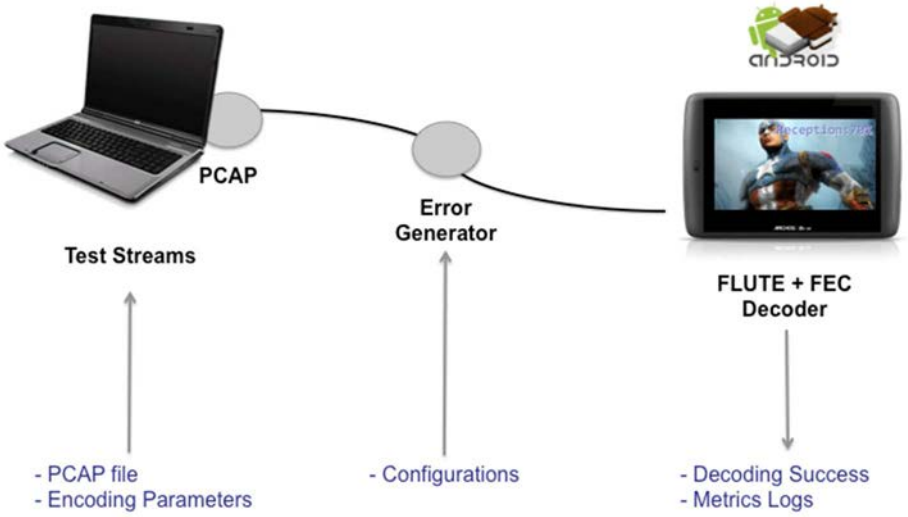


Figure 10: Test Platform

Figure 10 may suggest that data would be transmitted from laptop to device and experience errors over the connection. Despite this may be considered conceptually, in practice a local procedure on the PC is applied to go from the original PCAP file to an errored PCAP file. Prior knowledge of the error traces was not used by the FEC encoder or decoder.

6.7.3.2 Download Delivery

6.7.3.2.1 Summary Test Cases

The following parameters for each test case parameters are specified:

- FS is the file size in bytes
- T' is the FEC payload size.
- T is the symbol size. Typically T = T' unless there are multiple symbols per packet
- Kt is the total number of source symbols, i.e., Kt = ceil(FS/T)
- Z is the total number of source blocks
- O is the transmission overhead in percent according to the table provided by the proponents
- Nt is the resulting number of total symbols defined as Kt*(1+O/100)
- The code specific FEC-OTI (see TS 26.346, clause 7.2.9), e.g. the partitioning and sub-blocking parameters
- SeSt is the sending strategy with IL = Interleaved, n/a not applicable and SQ sequential:
    - Sequential = send all packets for the first source block, followed by all packets for the second source block, followed by all symbols for the third source block, etc. In addition, send all packets in order of the ESI.
    - Interleaved = send a first packet for each of the Z source blocks, followed by a second packet for each of the Z source blocks, followed by a third packet for each of the Z source blocks, etc.
    - Unless otherwise noted the symbols within each source block are assumed sent in order of increasing ESI-value starting with the first source symbol. If any other sending order for symbols within each source block is utilized it should be explicitly noted under Notes.

It is further expected that of the Z source blocks:

- the first Z1 have source block size K1 = ceil(Kt/Z)
- the remaining Z2 have source block size K2 = floor(Kt/Z)
- and Z1 = Kt – K2*Z and Z2=Z-Z1.

The test cases are summarized in Table 19. The test cases LD60_110, LD118_108 and LD119_109 are done to apply the error streams of lower loss rates to higher overhead streams. The test cases LD60_110, LD_118_108 and LD119_109 are optional.

Table 19: Parameters for Download Test Case

Common					Code-Specific						
Test Case	**Error conditions**	**File size FS**	**T'**	**Kt**	**Z**	**T**	**OTI**	**O**	**Nt**	**SeSt**	**Notes**
LD60	Markov, 3 km/h, 20%	HD	1288	1465402							
LD108	Markov, 120 km/h, 5%	Clip	1288	2443							
LD109		SD	1288	104207							
LD110		HD	1288	1465402							
LD118	Markov 120 km/h, 20%	Clip	1288	2443							
LD119		SD	1288	104207							
LD60_110	Markov 120 km/h, 5%	HD	1288	1465402							
LD118_108		Clip	1288	2443							
LD119_109		SD	1288	104207							

6.7.3.2.2 Generate FLUTE Packet Test Streams

6.7.3.2.2.1 Process

To generate the FLUTE packet test streams, the following actions are applied on the host. Some UNIX operation system is assumed with basic UNIX commands available.

- Download the following file `http://media.xiph.org/ED/ed-pixlet.mov`
- for each test case LDX according to Table 19

- generate segments and MD5
  - generate temporary file of size FS:
    `head -c <file size> ed-pixlet.mov > data.tmp`
  - create the MD5 for the file:
    `cat data.tmp | openssl md5 | awk '{ print $2 }' > ldX.md5`
- FEC encode to PCAP file as follows
  - Put FDT for the file in first packet specifying at least the following parameters
    - TOI
    - FEC-OTI

NOTE: Content-Location and Content-Length may not be added as they are not necessary. Transfer-Length in the FEC-OTI is sufficient.

- encode file into ALC/LCT packets using the test case parameters according to Table 19 for the candidate. The end of session and end of object transmission signalling may be used by setting the A and B flag in the LCT header.
- provide packets with UDP payload size according to Table 19. The ALC/LCT/UDP/IPv4 header is in total 44 bytes.

6.7.3.2.2.2 Output

The output from this process is, for each test case:

- TOI and MD5 for the file. Note that the TOI and MD5 are not code specific. Note that the TOI in this case is typically 1.
- PCAP file that contains encoded file preceded with an FDT (for details refer to TS 26.346 [3], clause 7.2.9). The PCAP file name for an example code with code name X is provided in Table 20 along with the total number of packets.

Table 20: PCAP files and Segment List for a virtual code X

Test Case	Error conditions	PCAP file	Number of Packets (Code-specific)	MD5 file
LD60	Markov, 3 km/h, 20%	`ld060_codeX.cap`		`ld060.md5`
LD108	Markov, 120 km/h, 5%	`ld108_codeX.cap`		`ld108.md5`
LD109		`ld109_codeX.cap`		`ld109.md5`
LD110		`ld110_codeX.cap`		`ld110.md5`
LD118	Markov, 120 km/h, 20%	`ld118_codeX.cap`		`ld118.md5`
LD119		`ld119_codeX.cap`		`ld119.md5`

6.7.3.2.3 Generate Erroneous Packet Streams

6.7.3.2.3.1 LTE Traces

Several LTE Error Trace are provided for each test case in the attached package. The files are named `error_trace_ld<testcase>_<trno>.txt`. The details are summarized in Table 21.

The format of the error traces is as follows

```
<Number L of loss/received events in ASCII>[newline]
L x {01}
```

where *L* is the maximum number of packets in ASCII that the input PCAP file may have followed on the next line with a string of length *L* made of ASCII characters '0' (packet received) and '1' (packet is lost). One example would be:

```
12
001011100100
```

i.e. the length of the of string of 0s and 1s is given by the integer on the first line.

Table 21: Error traces for download test cases with losses and loss statistics. These are accumulated.

Test Case	Error conditions	File size	S	PCAP file	Length N	Loss Percentage
LD60	Markov, 3 km/h, 20%	HD	1	error_trace_ld60_<trno>.cap	2000000	20.14
LD108	Markov, 120 km/h, 5%	Clip	20	error_trace_ld108_<trno>.cap	3400	5.47
LD109		SD	5	error_trace_ld109_<trno>.cap	150000	5.46
LD110		HD	1	error_trace_ld110_<trno>.cap	2000000	5.48
LD118	Markov, 120 km/h, 20%	Clip	20	error_trace_ld118_<trno>.cap	3400	20.85
LD119		SD	5	error_trace_ld119_<trno>.cap	150000	20.81
LD60_110	Markov, 120 km/h, 5%	HD	1	error_trace_ld110_<trno>.cap	2000000	5.47
LD118_108		Clip	20	error_trace_ld1108_<trno>.cap	3400	5.46
LD119_109		SD	5	error_trace_ld109_<trno>.cap	150000	5.48

A process for generating the error traces independently is provided in Annex B.2.

6.7.3.2.3.2 Apply to LTE traces to PCAP streams

In order to introduce loss into a controlled manner to the PCAP files using the Markov error traces, a tool called `pcaploss`, available in source code form, is available and attached in the package (including Makefile). This tool takes a `pcap` file as input and transforms it into another altered `pcap`. The usage message for pcaploss is:

```
pcaploss: Usage: ./pcaploss <pcap_in> <pcap_out> <loss_file> [<#pkts>]
```

where the format of the loss trace file is according to the format introduced in section 6.7.3.3.2.1. If the optional integer argument `#pkts` is present, only the number of packets indicated by `#pkts` will be read in from `pcap_in` before `pcaploss` closes the output file and stops.

The `pcap` for transmission may be prepped with the right MAC/IP addresses for both sender and receiver. On the sender side MAC and IP can be obtained with command '`ipconfig/ all`' on Windows, e.g.:

```
Ethernet adapter Local Area Connection 4:

   Connection-specific DNS Suffix  . :
   Description . . . . . . . . . . . : SAMSUNG Mobile USB Remote NDIS Network Device
   Physical Address. . . . . . . . . : 02-65-64-60-6E-0B
   DHCP Enabled. . . . . . . . . . . : Yes
   Autoconfiguration Enabled . . . . : Yes
   Link-local IPv6 Address . . . . . : fe80::117:1bc9:34df:dd76%26(Preferred)
   IPv4 Address. . . . . . . . . . . : 192.168.42.149(Preferred)
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Lease Obtained. . . . . . . . . . : Monday, July 16, 2012 3:37:43 PM
   Lease Expires . . . . . . . . . . : Monday, July 16, 2012 4:37:50 PM
   Default Gateway . . . . . . . . . : 192.168.42.129
   DHCP Server . . . . . . . . . . . : 192.168.42.129
   DHCPv6 IAID . . . . . . . . . . . : 855795044
   DHCPv6 Client DUID. . . . . . . . : 00-01-00-01-14-97-F4-E0-F4-CE-46-AC-6F-32
   DNS Servers . . . . . . . . . . . : 192.168.42.129
   NetBIOS over Tcpip. . . . . . . . : Enabled
```

where hardware and IP addresses are `02:65:64:60:6E:0B` and `192.168.42.149` respectively. On the receiver side a multicast IP address and associated MAC could be `230.20.20.10` and `01:00:5e:66:14:14:0a`.

With the information above and for each test case LDY in Table 26 and each trace number `trno`, the following process is applied:

```
./tcprewrite --distipmap=0.0.0.0/0:230.20.20.10 --enet-dmac= 01:00:5e:66:14:14:0a --
srcipmap=0.0.0.0/0: 192.168.42.149 --enet-smac=02:65:64:60:6E:0B --fixcsum -i
ldY_codeX.cap -o temp.cap

./pcaploss temp.cap ldY_codeX_ldZ_<trno>.cap errortrace_ldZ_<trno>.txt
```

Note that the integration of the Ethernet and IP addresses with `tcprewrite` is optional and may only be done absence of any other knowledge. `tcprewrite` is included in the `TCPreplay` suite, for details refer to section B.6.2.

6.7.3.2.3.3 Output

The outputs of this process are *S* PCAP file for each test case. The PCAP files are summarized in Table 22. The length of the PCAP file depends on the loss statistics.

Table 22: PCAP files for a virtual code X after applying channel that maps to specific channel model

Test Case	Error conditions	File size	S	PCAP file
LD60	Markov, 3 km/h, 20%	HD	1	`ld060_codeX_ld060_<trno>.cap`
LD108	Markov, 120 km/h, 5%	Clip	20	`ld108_codeX_ld108_<trno>.cap`
LD109		SD	5	`ld109_codeX_ld109_<trno>.cap`
LD110		HD	1	`ld110_codeX_ld110_<trno>.cap`
LD118	Markov, 120 km/h, 20%	Clip	20	`ld118_codeX_ld118_<trno>.cap`
LD119		SD	5	`ld119_codeX_ld119_<trno>.cap`
LD60_110	Markov, 120 km/h, 5%	HD	1	`ld060_codeX_ld110_<trno>.cap`
LD118_108		Clip	20	`ld118_codeX_ld108_<trno>.cap`
LD119_109		SD	5	`ld119_codeX_ld109_<trno>.cap`

6.7.3.2.4 Generate Device Performance Measures

6.7.3.2.4.1 Setup

The following device/operating conditions are used:

NOTE: "Trade name(s) of product(s)] are an example(s) of a suitable product(s) available commercially. This information is given for the convenience of users of the present document and does not constitute an endorsement by 3GPP of these product(s)."

- Device:
  - Samsung Galaxy S2™ (GT-I9100) Smartphone, running Android 4.0.3. The processor is a Dual-core Exynos 4210 1.2GHz processor ARM Cortex-A9.
  - Samsung MB-MSBGA™ Flash memory card - 32 GB microSDHC - 1 x microSDHC SD Card (Class 10)
  - Root access is applied to the device, for details see Annex B.3.
  - `network2sd` executable for reading packets from network interface and writing it in a suitable manner to the SD card in order optimize reading while decoding. For details on functionalities, see section 6.7.3.2.4.1.1.
  - `ld_decoder` executable for FEC decoding based on data on the SD card of the device and for writing subblock data to SD card. For details on functionalities, see section 6.7.3.2.4.1.2.
  - push the Unix `'time'` command on the device, for details see Annex B.4.
  - an ssh server is installed and running on the device to get shell access while USB tethering is active. See Annex B.7 for details.
- The host PC:
  - can be any OS, but typically Windows or Linux
  - The host PC is connected to the Device using USB tethering through an interface. It is assumed that the interface has assigned name `Samsung`.
  - the host does have a functionality installed that permits to push the stored PCAP files to the device. For details, see Annex B.6. In the following it is assumed that the ColaSoft Packet Player is available.
- The details of connecting device and host PC are provided in Annex B.5.

6.7.3.2.4.1 Code-specific Tools

6.7.3.2.4.1.1 Read from network and write to SD

The `network2sd` executable for reading packets from network interface and writing it in a suitable manner to the SD card in order optimize reading while decoding. The `network2sd` writes some information to `stdout`, which is used by `ld_decoder` as input to locate the relevant information. The executable synchronises all buffers with the SD card before exiting (e.g. via sync() system call).

For the purpose of implementing receiving payload data reading and writing to flash/disk, standard Android procedures and functions are used.

6.7.3.2.4.1.2 Decoding from and to SD card

The `ld_decoder` executable reads input data from SD card and writes it back to SD card sub-block by sub-block. The `ld_decoder` receives information from the `network2sd` process in order to locate the relevant data. The executable are synchronised all buffers with the SD card before exiting (e.g. via the sync() system call).

6.7.3.2.4.2 Process

For each test case LDX from Table 19 and each `<trno>`, the following processes are carried out in the following sequence:

- On the device start the following process in directory `/data/data/berserker.android.apps.sshdroid/home` with device Wifi IP of 192.168.2.102 an ssh server running on port 2222

```
1. ssh -p 2222 root@192.168.2.102
2. When asked for passwd, type: "admin"
3. Use rm to clear all disk space on SD card
4. time -v ./network2sd info.txt 2> time1.txt
```

- On the host start the Colasoft Packet Player with the following
  - Adapter: Samsung
  - Packet File: Add -> File of type: libpcap (*.cap)
  - Select file `ldY_codeX_ldZ_<trno>.cap`
  - Click button "Play"
- After termination at the device, the following is carried on the device

```
5. echo 1 > /proc/sys/vm/drop_caches (# this is for clearing caches)
6. time -v ./ld_decoder info.txt 2> time2.txt
7. (generate md5 and TOI > out.txt)
```

- After termination at the device, the following is carried out on the host

```
scp –P 2222
root@192.168.2.102:/data/data/berserker.android.apps.sshdroid/home/out.txt
ldY_codeX_ldZ_<trno>.out

scp –P 2222
root@192.168.2.102:/data/data/berserker.android.apps.sshdroid/home/time1.txt
ldY_codeX_ldZ_<trno>.time1

scp –P 2222
root@192.168.2.102:/data/data/berserker.android.apps.sshdroid/home/time2.txt
ldY_codeX_ldZ_<trno>.time2
```

6.7.3.2.4.2 Error Free Process

In order to understand the influence of supplementary processes to the FEC decoding, the same process as described in clause 6.7.3.2.4.1 may be carried out for the error-free `pcap` files. To do so, all files `ldY_codeX_ldZ_<trno>.*` can be replaced by `ldY_codeX.*`.

6.7.3.2.4.3 Output

The output of this process is one performance file and one result file for each test case. The files are summarized in Table 23.

Table 23: Performance and result file for a virtual code X after decoding

Test Case	Error-Free Performance (optional)	S	Result	Performance files
LD60	ld060_codeX.time	1	ld060_codeX_ld060_<trno>.out	ld060_codeX_ld060_<trno>.time
LD108	ld108_codeX.time	20	ld108_codeX_ld108_<trno>.out	ld108_codeX_ld108_<trno>.time
LD109	ld109_codeX.time	5	ld109_codeX_ld109_<trno>.out	ld109_codeX_ld109_<trno>.time
LD110	ld110_codeX.time	1	ld110_codeX_ld110_<trno>.out	ld110_codeX_ld110_<trno>.time
LD118	ld118_codeX.time	20	ld118_codeX_ld118_<trno>.out	ld118_codeX_ld118_<trno>.time
LD119	ld119_codeX.time	5	ld119_codeX_ld119_<trno>.out	ld119_codeX_ld119_<trno>.time
LD60_110	ld060_codeX.time	1	ld060_codeX_ld110_<trno>.out	ld060_codeX_ld110_<trno>.time
LD118_108	ld108_codeX.time	20	ld118_codeX_ld108_<trno>.out	ld118_codeX_ld108_<trno>.time
LD119_109	ld109_codeX.time	5	ld119_codeX_ld109_<trno>.out	ld119_codeX_ld109_<trno>.time

6.7.3.2.5 Evaluation

6.7.3.2.5.1 General

After all test cases are completed the output files as presented in Table 23 are available. These files may be moved back to the host for evaluation.

6.7.3.2.5.2 Correct Decoding

To verify that decoding was successful for each test case or to identify the number of unsuccessful attempts, the result files `ldY_codeX_ldZ_<trno>.out` are collected and for each one it is compared if the TOI and MD5 are identical with `ldY.md5`. If not identical, one error event is recorded.

6.7.3.2.5.3 Performance Evaluation

The output will then be extracted from the two files which include the output from the time command similar as seen below:

```
Command being timed: "ld_decoder"
User time (seconds): 1.49
System time (seconds): 0.36
Percent of CPU this job got: 73%
Elapsed (wall clock) time (h:mm:ss or m:ss): 0m 2.52s
Average shared text size (kbytes): 0
Average unshared data size (kbytes): 0
Average stack size (kbytes): 0
Average total size (kbytes): 0
Maximum resident set size (kbytes): 165456
Average resident set size (kbytes): 0
Major (requiring I/O) page faults: 1
Minor (reclaiming a frame) page faults: 21740
Voluntary context switches: 9659
Involuntary context switches: 10442
Swaps: 0
File system inputs: 0
File system outputs: 0
Socket messages sent: 0
Socket messages received: 0
Signals delivered: 0
Page size (bytes): 4096
```

```
    Exit status: 0
```

The relevant entries here are "`system time`", "`user time`" (the sum of which is to be reported as the processing cost), and "`Maximum resident set size`". The memory usage to be reported is 1/4 of that given as the "`Maximum resident set size`" in an unpatched busybox 1.19.0. The reason for this division by 4 is that busybox has a bug which causes it to overestimate memory usage by a factor of 4, just like the GNU time utility from which it is presumably inheriting this mistake. See the bug report here (see note).

NOTE: http://lists.gnu.org/archive/html/bug-gnu-utils/2008-12/msg00047.html

The following performance data measurement is proposed:

- Generate the numbers from above for the considered test case
- Generate the numbers from above for a zero loss trace
- Report the following numbers for each test case and the zero loss trace:
    - *U*: User time (seconds)
    - *S*: System time (seconds)
    - *P*: Percent of CPU this job got
    - *W*: Elapsed (wall clock) time (h:mm:ss or m:ss):
    - *M*: Maximum resident set size (kbytes)
- Generate the following numbers for performance evaluation based on the above results and the object size *F* (in bytes) for each test case and trace number:
    - `Speed`: Average decoding speed (in MBit/s): *F*8*/(1000000*(*U*+*S*))
    - `Time1`: Decoding time (in s): *U*+*S*
    - `Time2`: weighted elapsed time (in s): *P***W*/100
    - `Memory`: Peak memory usage (in MBytes): *M*/4096

6.7.3.2.5.4 Performance Documentation

The following values are to be reported for each test case be using the results from each **`trno`** = 0, ..., *S*-1 and the error free decoding:

- *Np* the total number of packets used for decoding
- *E* the total number of file delivery attempts that failed (should be 0)
- `AvSpeed` the average speed over all *S* decoding attempts
- `AvTime1` the average decode time over all *S* decoding attempts
- `AvTime2` the weighted elapsed time over all *S* decoding attempts
- `MinSpeed` the minimum speed over all *S* decoding attempts
- `MaxTime1` the maximum decoding over all *S* decoding attempts
- `MaxTime2` the weighted elapsed time over all *S* decoding attempts
- `MaxMem` the maximum memory over all *S* decoding attempts
- `AvCPU` the average value of *P* over all *S* decoding attempts
- `EfSpeed` the speed for error-free decoding attempt
- `EfTime1` the Time for error-free decoding attempt
- `EfTime2` the Time for error-free decoding attempt
- `EfMem` the Memory for error-free decoding attempt

Note that

- the error-free results are not required, but recommended to be provided.

- the data are obviously expected to be provided for the `network2sd` (in Table 25) and `ld_decoder` process (in Table 24).
- The test cases LD60_110, LD118_108 and LD119_109 are optional

Table 24: Performance Data for Download Delivery Test Cases for ld_decoder

Test Case	S	*Np*	*E*	AvSpeed (MBit/s)	AvTime1 (sec)	AvTime2 (sec)	MinSpeed (MBit/s)	MaxTime2 (sec)	MaxTime2 (sec)	MaxMem (MByte)	AvCPU	EfSpeed (MBit/s)	EfTime1 (sec)	EfTime2 (sec)	EfMem (MByte)
LD60	1														
LD108	20														
LD109	5														
LD110	1														
LD118	20														
LD119	5														
LD60_110	1														
LD118_108	20														
LD119_109	5														

Table 25: Performance Data for Download Delivery Test Cases for network2sd

Test Case	AvSpeed (MBit/s)	AvTime1 (sec)	AvTime2 (sec)	MinSpeed (MBit/s)	MaxTime2 (sec)	MaxTime2 (sec)	MaxMem (MByte)	AvCPU	EfSpeed (MBit/s)	EfTime1 (sec)	EfTime2 (sec)	EfMem (MByte)
LD60												
LD108												
LD109												
LD110												
LD118												
LD119												
LD60_110												
LD118_108												
LD119_109												

6.7.3.3 Streaming Delivery

6.7.3.3.1 Summary Test Cases

Table 26 summarizes the streaming test cases for device-based evaluation. The test cases LS45_33, LS51_50 and LS48_36 are done to apply the error streams of lower loss rates to higher overhead streams. These test cases LS45_33, LS51_50 and LS48_36 are optional.

Table 26: Parameters for Streaming Test Case

Common Parameters							Code-specific Parameters (SHOWN ARE PARAMETERS FOR OFFICIAL TRACES WITH IDEAL CODE)			
Test Case	**Error conditions**	**Segment Duration**	**T'**	**N'**	**Packet Interval**	**Number Segments Y (time=30min)**	**G**	**K**	**Segment Size S**	**Media Rate**
LS21	Markov, 3 km/h, 20%	1s	1288	100	10ms	1800	1	35	45080	360.6
LS49		2s	1288	200	10ms	900	1	103	132664	530.7
LS24		4s	1288	400	10ms	450	1	248	319424	638.8
LS33	Markov, 120 km/h, 5%	1s	1288	100	10ms	1800	1	85	109480	875.8
LS50		2s	1288	200	10ms	900	1	177	227976	911.9
LS36		4s	1288	400	10ms	450	1	363	467544	935.1
LS45	Markov, 120 km/h, 20%	1s	1288	100	10ms	1800	1	65	83720	669.8
LS51		2s	1288	200	10ms	900	1	139	179032	716.1
LS48		4s	1288	400	10ms	450	1	291	374808	749.6
LS45_33	Markov, 120 km/h, 5%	1s	1288	100	10ms	1800	1	65	83720	669.8
LS51_50		2s	1288	200	10ms	900	1	139	179032	716.1
LS48_36		4s	1288	400	10ms	450	1	291	374808	749.6

6.7.3.3.2 Generate FLUTE Packet Test Streams

6.7.3.3.2.1 Process

In order to generate the FLUTE Packet Test streams, the following actions are to be applied on the host

- Download the following file `http://media.xiph.org/ED/ed-pixlet.mov`
- for each test case LSX according to Table 26

- generate segments and MD5
  - split the file in to Y segments, each of size S
  - create the MD5 for each of the segments and create a file that lists the TOI and the MD5
  - the shell script in section 0 can be used for this purpose. It creates as output the segment number as well as the MD5 for the segment
- FEC encode to PCAP file as follows:
  - Provide FDT for each segment just before first packet of a segment specifying at least the following parameters
    - TOI
    - FEC-OTI

    Note:
    - Content-Location and Content-Length are not added as they are not necessary. Transfer Encoding is sufficient.
  - encode each segment sequentially with increasing TOI numbers *1* ... *Y* into ALC/LCT packets using the test case parameters according to Table 26 for the candidate
    - number of source symbols K,
    - number of transmitted symbols N,
    - symbol size T,
    - sub-blocking parameters if needed

    Note: End of session and end of object transmission signalling may be used by setting the A and B flag in the LCT header.
  - for all ALC/LCT packets with TOI not equal to 0,
    - provide packets with UDP payload size according to Table 26. The ALC/LCT/UDP/IPv4 header is in total 44 bytes.
    - If in doubt or unclear what to use, include the timing for the real-time bitrate, i.e. 1 packet every according to the packet interval in Table 26. Note that the tool `pcaploss` rewrites correctly the packet timestamps with the right transmission time interval.
  - for all ALC/LCT packets with TOI equal to 0, i.e. FDT packets
    - provide packets with UDP payload size according to Table 26. The ALC/LCT/UDP/IPv4 header is in total 44 bytes.
    - If in doubt or unclear what to use, include a timing that is 50% of the packet interval in Table 26 earlier than the one in the first packet of the object with the TOI included in this FDT. Note that the tool `pcaploss` rewrites correctly the packet timestamps with the right transmission time interval.

6.7.3.3.2.2 Output

The output from this process is for each test case:

- File that contains TOI and MD5 for each of the segments
- PCAP file that contains a sequence of segments prefixed with a single multi-packet FDT that summarizes the entire sequence. The PCAP file name for a code with code name X is provided in Table 27 along with the total number of packets

Table 27: PCAP files and Segment List for a virtual code X

Test Case	Error conditions	PCAP file	Number of Data Packets	Segment list
LS21	Markov, 3 km/h, 20%	ls21_codeX.cap	180000	ls21.md5
LS49		ls49_codeX.cap	180000	ls49.md5
LS24		ls24_codeX.cap	180000	ls24.md5
LS33	Markov, 120 km/h, 5%	ls33_codeX.cap	180000	ls33.md5
LS50		ls50_codeX.cap	180000	ls50.md5
LS36		ls36_codeX.cap	180000	ls36.md5
LS45	Markov, 120 km/h, 20%	ls45_codeX.cap	180000	ls45.md5
LS51		ls51_codeX.cap	180000	ls51.md5
LS48		ls48_codeX.cap	180000	ls48.md5
LS45	Markov, 120 km/h, 5%	ls45_codeX.cap	180000	ls45.md5
LS51		ls51_codeX.cap	180000	ls51.md5
LS48		ls48_codeX.cap	180000	ls48.md5

6.7.3.3.3 Generate Erroneous Packet Streams

6.7.3.3.3.1 LTE Traces

One LTE Error Trace is provided for each test case in the attached package. The files are named error_trace_ls<testcase>.txt. The details are summarizes in Table 28.

The format of the error traces is as follows

```
<Number L of loss/received events in ASCII>[newline]
L x {01}
```

where *L* is the maximum number of packets that the input pcap file may have followed on the next line with a string made of characters '0' (packet received) and '1' (packet is lost) of length L. One example would be:

```
12
001011100100
```

i.e. the length of the of string of 0s and 1s is given by the integer on the first line.

Table 28: Error traces for streaming test cases with losses and loss percentage

Test Case	Error conditions	Error Trace	Length N	Loss Percentage
LS21	Markov, 3 km/h, 20%	errortrace_ls21.txt	180000	19.94
LS49		errortrace_ls49.txt	180000	19.94
LS24		errortrace_ls24.txt	180000	19.94
LS33	Markov, 120 km/h, 5%	errortrace_ls33.txt	180000	5.41
LS50		errortrace_ls50.txt	180000	5.41
LS36		errortrace_ls36.txt	180000	5.41
LS45	Markov, 120 km/h, 20%	errortrace_ls45.txt	180000	20.80
LS51		errortrace_ls51.txt	180000	20.80
LS48		errortrace_ls48.txt	180000	20.80
LS45_33	Markov, 120 km/h, 5%	errortrace_ls33.txt	180000	5.41
LS51_50		errortrace_ls50.txt	180000	5.41
LS48_36		errortrace_ls36.txt	180000	5.41

A process for generating the error traces independently is provided in Annex B.

6.7.3.3.3.2 Apply to LTE traces to PCAP streams

In order to introduce loss into a controlled manner to the PCAP files using the Markov error traces, a tool called `pcaploss`, available in source code form, is available and attached in the package (including Makefile). This tool takes a `pcap` file as input and transforms it into another altered `pcap`. The usage message for pcaploss is:

```
pcaploss: Usage: ./pcaploss <pcap_in> <pcap_out> <loss_file> [<#pkts>]
```

where the format of the loss trace file is according to the format introduced in clause 6.7.3.3.2.1. If the optional integer argument `#pkts` is present, only the number of packets indicated by `#pkts` will be read in from pcap_in before pcaploss closes the output file and stops.

The `pcap` for transmission may be prepped with the right MAC/IP addresses for both sender and receiver. On the sender side MAC and IP can be obtained with command '`ipconfig/ all`' on Windows, e.g.:

```
Ethernet adapter Local Area Connection 4:

   Connection-specific DNS Suffix  . :
   Description . . . . . . . . . . . : SAMSUNG Mobile USB Remote NDIS Network Device
   Physical Address. . . . . . . . . : 02-65-64-60-6E-0B
   DHCP Enabled. . . . . . . . . . . : Yes
   Autoconfiguration Enabled . . . . : Yes
   Link-local IPv6 Address . . . . . : fe80::117:1bc9:34df:dd76%26(Preferred)
   IPv4 Address. . . . . . . . . . . : 192.168.42.149(Preferred)
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Lease Obtained. . . . . . . . . . : Monday, July 16, 2012 3:37:43 PM
   Lease Expires . . . . . . . . . . : Monday, July 16, 2012 4:37:50 PM
   Default Gateway . . . . . . . . . : 192.168.42.129
   DHCP Server . . . . . . . . . . . : 192.168.42.129
   DHCPv6 IAID . . . . . . . . . . . : 855795044
   DHCPv6 Client DUID. . . . . . . . : 00-01-00-01-14-97-F4-E0-F4-CE-46-AC-6F-32
   DNS Servers . . . . . . . . . . . : 192.168.42.129
   NetBIOS over Tcpip. . . . . . . . : Enabled
```

where hardware and IP addresses are `02-65-64-60-6E-0B` and `192.168.42.149` respectively. On the receiver side a multicast IP address and associated MAC could be `230.20.20.10` and `01:00:5e:66:14:14:0a`.

With the information above and for each test case LSY(_Z) in Table 26, the following process is applied:

```
./tcprewrite --distip=0.0.0.0/0:230.20.20.10 --enet-dmac= 01:00:5e:66:14:14:0a --
srcip=0.0.0.0/0:192.168.42.149 --enet-smac=02:65:64:60:6E:0B  --fixcsum -i
ldY_codeX.cap -o temp.cap

./pcaploss temp.cap ldY_codeX_ldZ_<trno>.cap errortrace_ldZ_<trno>.txt
```

Note that the integration of the Ethernet and IP addresses with `tcprewrite` is optional and may only be done absence of any other knowledge. `tcprewrite` is included in the `TCPreplay` suite, for details refer to Annex B.6.2.

6.7.3.3.3.3 Output

The output of this process is one PCAP file for each test case. The PCAP files are summarized in Table 29. The length of the PCAP file depends on the loss statistics.

Table 29: PCAP files for a virtual code X after applying channel that maps to specific channel model

Test Case	Error conditions	Error Trace	Number of Packets (CODE DEPENDENT)
LS21	Markov, 3 km/h, 20%	ls21_codeX_ls21.cap	
LS49		ls49_codeX_ls49.cap	
LS24		ls24_codeX_ls24.cap	
LS33	Markov, 120 km/h, 5%	ls33_codeX_ls33.cap	
LS50		ls50_codeX_ls50.cap	
LS36		ls36_codeX_ls36.cap	
LS45	Markov, 120 km/h, 20%	ls45_codeX_ls45.cap	
LS51		ls51_codeX_ls51.cap	
LS48		ls48_codeX_ls48.cap	
LS45_33	Markov, 120 km/h, 5%	ls45_codeX_ls45.cap	
LS51_50		ls51_codeX_ls51.cap	
LS48_36		ls48_codeX_ls48.cap	

6.7.3.3.4 Generate Device Performance Measures

6.7.3.3.4.1 Setup

The following device/operating conditions are used:

NOTE: "Trade name(s) of product(s)] are an example(s) of a suitable product(s) available commercially. This information is given for the convenience of users of the present document and does not constitute an endorsement by 3GPP of these product(s)."

- Device:
  - Samsung Galaxy S2™ (GT-I9100) Smartphone, running Android 4.0.3. The processor is a Dual-core Exynos 4210 1.2GHz processor ARM Cortex-A9.
  - Root access is applied to the device, for details see Annex B.3.
  - `ls_decoder` executable for FEC decoding available on the device, for details on functionalities, see clause 6.7.3.3.3.2.
  - `verifysegm` for generating the md5 or a received segment push the data to `stdout` with TOI and length. For details on functionalities, see Annex B.8.
  - push the Unix `'time'` command on the device, for details see Annex B.4.
- The host PC:
  - can be any OS, but typically Windows or Linux
  - The host PC is connected to the Device using USB tethering through an interface. It is assumed that the interface has assigned name `Samsung`.
  - the host does have a functionality installed that permits to push the stored PCAP files to the device. For details, see Annex B.6. In the following it is assumed that ColaSoft Packet Player is available.
- The details of connecting device and host PC are provided in Annex B.5.

6.7.3.3.4.2 Decoder

The `ls_ecoder` executable receives its input data via the network interface card (UDP/ALC/LCT packets) and writes on `stdout` decoded source block.

If correction of the segment is successful, this application writes on `stdout`:

```
[ TOI (32-bit) |  length (32-bit) | <sequence of segment bytes>]
```

where `TOI` is the segment Transport Object Identifier followed by the length of the decoded segment in bytes and the actual recovered segment data. `TOI` and `length` are in network-byte order.

Note that the proponent need not use the provided `verifysegm`, but provide its own verification program. In this case the interface between the decoder and the verification program may for example use the segment name instead of the TOI.

6.7.3.3.4.3 Process

For each test case LSX from Table 26, the following processes are carried out in the following sequence:

- On the device start the following process in directory `/data/data/berserker.android.apps.sshdroid/home` with device Wifi IP of 192.168.2.102 and an ssh server running on port 2222:

```
1. ssh -p 2222 root@192.168.2.102
2. When asked for password, type 'admin'
3. time -v ls_decoder 2> time.txt | time -v verifysegm > out.txt
```

- On the host start the Colasoft Packet Player with the following:
  - Adapter: Samsung
  - Packet File: Add -> File of type: libpcap (*.cap)
  - Select file `lsY_codeX_lsZ_<trno>.cap`
  - Click button Play
- After termination at the device, the following is carried out on the host:

```
scp –P 2222
root@192.168.2.102:/data/data/berserker.android.apps.sshdroid/home/out.txt
lsY_codeX_lsZ.out

scp –P 2222
root@192.168.2.102:/data/data/berserker.android.apps.sshdroid/home/time.txt
lsY_codeX_lsZ.time
```

6.7.3.3.4.4 Error-Free Process

The same process as described in section 6.7.3.3.3 is carried out for the error-free `pcap` files. To do so, all files `lsY_codeX_lsZ.*` are replaces by `lsY_codeX.*`.

6.7.3.3.4.5 Output

The output of this process is one performance file and one result file for each test case. The files are summarized in Table B.1.

Table 30: Performance and result file for a virtual code X after decoding

Test Case	Error conditions	Result	Performance	Error-Free Performance
LS21	Markov, 3 km/h, 20%	`ls21_codeX_ls21.out`	`ls21_codeX_ls21.time`	`ls21_codeX.time`
LS49		`ls49_codeX_ls49.out`	`ls49_codeX_ls49.time`	`ls49_codeX.time`
LS24		`ls24_codeX_ls24.out`	`ls24_codeX_ls24.time`	`ls24_codeX.time`
LS33	Markov, 120 km/h, 5%	`ls33_codeX_ls33.out`	`ls33_codeX_ls33.time`	`ls33_codeX.time`
LS50		`ls50_codeX_ls50.out`	`ls50_codeX_ls50.time`	`ls50_codeX.time`
LS36		`ls36_codeX_ls36.out`	`ls36_codeX_ls36.time`	`ls36_codeX.time`
LS45	Markov, 120 km/h, 20%	`ls45_codeX_ls45.out`	`ls45_codeX_ls45.time`	`ls45_codeX.time`
LS51		`ls51_codeX_ls51.out`	`ls51_codeX_ls51.time`	`ls51_codeX.time`
LS48		`ls48_codeX_ls48.out`	`ls48_codeX_ls48.time`	`ls48_codeX.time`
LS45_33	Markov, 120 km/h, 5%	`ls45_codeX_ls33.out`	`ls45_codeX_ls33.time`	`ls45_codeX.time`
LS51_50		`ls51_codeX_ls50.out`	`ls51_codeX_ls50.time`	`ls51_codeX.time`
LS48_36		`ls48_codeX_ls36.out`	`ls48_codeX_ls36.time`	`ls48_codeX.time`

6.7.3.3.5 Evaluation

6.7.3.3.5.1 General

After all test cases are completed the output files as presented in Table 30 are available. These files are moved back to the host for evaluation.

6.7.3.3.5.2 Correct Decoding

The number of successfully decoded segments can be computed as follows (on a UNIX machine):

```
cat lsY_codeX.md5 lsY_codeX.out | sort | uniq –d | wc –l
```

6.7.3.3.5.3 Performance Evaluation

The output of `lsY_codeX(_lsY).time` will be something like this:

```
Command being timed: "ls_decoder"
User time (seconds): 1.49
System time (seconds): 0.36
Percent of CPU this job got: 73%
Elapsed (wall clock) time (h:mm:ss or m:ss): 0m 2.52s
Average shared text size (kbytes): 0
Average unshared data size (kbytes): 0
Average stack size (kbytes): 0
Average total size (kbytes): 0
Maximum resident set size (kbytes): 165456
Average resident set size (kbytes): 0
Major (requiring I/O) page faults: 1
Minor (reclaiming a frame) page faults: 21740
Voluntary context switches: 9659
Involuntary context switches: 10442
Swaps: 0
File system inputs: 0
File system outputs: 0
Socket messages sent: 0
Socket messages received: 0
Signals delivered: 0
Page size (bytes): 4096
Exit status: 0
```

The relevant entries here are `system time`, `user time` (the sum of which is to be reported as the processing cost), and `Maximum resident set size`. The memory usage to be reported is 1/4 of that given as the `Maximum resident set size` in an unpatched busybox 1.19.0. The reason for this division by 4 is that busybox has a bug which causes it to overestimate memory usage by a factor of 4, just like the GNU time utility from which it is presumably inheriting this mistake. See the bug report here (see note).

NOTE: http://lists.gnu.org/archive/html/bug-gnu-utils/2008-12/msg00047.htm

The following performance data measurement is proposed:

- Generate the numbers from above for the considered test case
- Generate the numbers from above for a zero loss trace
- Report the following numbers for each test case and the zero loss trace:
  - *U*: `User time (seconds)`
  - *S*: `System time (seconds)`
  - *P*: `Percent of CPU this job got`
  - *W*: `Elapsed (wall clock) time (h:mm:ss or m:ss)`:
  - *M*: `Maximum resident set size (kbytes)`
- Generate the following numbers for performance evaluation based on the above results and the segment duration *D* (in seconds), the media bitrate *R* (in kBit/s), and the duration of the media data *t* (in seconds):
  - `Speed`: Average decoding speed (in MBit/s): *R***t*/(1000*(*U*+*S*))

- `Latency`: Average decoding latency (in ms): $D*(1000*(U+S))/t$
- `Memory`: Peak memory usage (in MBytes): $M/4096$

6.7.3.3.5.4 Performance Documentation

The performance should be documented according to Table 31. The right three columns document the performance for error-free transmission. Also the values for *U*, *S*, *P*, *W* and *M* should be provided.

Table 31: Performance Data for Streaming Test Cases

Test Case	Error conditions	G	K	E	Speed (MBit/s)	Latency (ms)	Memory (MByte)	EF-Speed (MBit/s)	EF-Latency (ms)	EF-Memory (MByte)
LS21	Markov, 3 km/h, 20%									
LS49										
LS24										
LS33	Markov, 120 km/h, 5%									
LS50										
LS36										
LS45	Markov, 120 km/h, 20%									
LS51										
LS48										
LS45_33	Markov, 120 km/h, 5%									
LS51_50										
LS48_36										

6.7.4 Tools

The attachment `Attachment-1-Tools.zip` contains the following files attached to this Technical Report specifically for the purpose of the test plan:

- `LossGenerator.zip`: A packet to generate the relevant Markov error traces.
- `pcaploss.zip`: Package that includes a tcprewrite functionality to generate PCAP traces with losses according to a Markov trace.
- `Traces.zip`: all relevant error traces for conducting the tests
- `verifysegm.zip`: verification tool to generate MD5 for generated segment.

6.7.5 Verification Process

A detailed verification process had been defined.

- Each candidate provides the availability to access pcap files and executables ld_decoder, network2sd and ls_decoder for verification.
- Any 3GPP member can repeat the tests according to the test plans in section 6 and indicate that the verification was not successful. Collaboration with the candidate proponent to resolve verification is encouraged.

7 FEC Candidates

7.1 Introduction

Based on review of the self-evaluation data of all submitted candidates, the following complete EFEC candidates have been agreed to pass the Qualification Criteria to be considered as a Qualifying Candidate in the EMM-EFEC selection procedure.

- RS+LDPC: The summary of the submitters is provided in clause 7.3
- Supercharged Codes: The summary of the submitters is provided in clause 7.4
- 6330 code: The summary of the submitters is provided in clause 7.5

In the evaluation of the candidate codes, benchmark codes are used. These are documented in clause 7.2.

7.2 Benchmark Codes

7.2.1 Ideal Code

A code is generally capable to handle one or a few or many of the following parameters

- T: source symbol size
- K: source block size and number of source symbols
- N: word length and number of encoding symbols

An ideal code with parameters (K, N, T) can reconstruct the K source symbols from any set of K of the N encoding symbols. Ideal codes exists, but are usually very complex in encoding and decoding, especially if K is not small or if N needs to be large.

7.2.2 MBMS FEC RFC 5053

The code is fully specified in IETF RFC 5053 and is also used in MBMS TS26.346.

7.3 RS+LDPC

The RS+LDPC code and its proposed application as MBMS application layer FEC is documented in the attachment `Attachment-6-RS+LDPC.zip` as provided by the proponent.

7.4 Supercharged Codes

The Supercharged Code provides application layer FEC protection.

7.5 6330 Code

The 6330 code and its proposed application as MBMS application layer FEC is documented in the attachment `Attachment-5-6330.zip` as provided by the proponent.

8 Performance of FEC Codes

8.1 Benchmark Codes: Ideal Code and RFC 5053

Attached to this document is an excel sheet named `Attachment-2-Benchmark-Codes.xls`. It contains all results of the benchmark codes.

Specifically it contains the following tabs:

- LTE-Download - 5053&Ideal: this tab provides the results for all 120 LTE download cases for both codes with the following details:
    - Kt: the total number of source symbols for the test cases for RFC 5053
    - FEC Overhead 5053: The FEC symbol overhead necessary to fulfil the criteria in percent for RFC 5053
    - Nt: the total number of overhead symbols for the RFC 5053 code
    - T: The symbol size in bytes for the RFC 5053 code
    - Z: the total number of source blocks for the RFC 5053 code
    - G: the number of symbols for each packet for the RFC 5053 code
    - Sending strategy:

        - n/a not applicable as a single source block
        - IL: source blocks are sent fully interleaved
        - SQ: source blocks are sent sequential
    - FEC Overhead Ideal (single SB): The FEC symbol overhead necessary to fulfil the criteria in percent when using an ideal code with a single source block.
    - Nt ideal (single SB): the total number of overhead symbols for the ideal (single source block) code
    - Kt with G=1: the total number of source symbols for the test cases for the ideal (single source block) code
    - T: The symbol size in bytes for the ideal (single source block)
    - Difference. The difference in overhead between the RFC 5053 code and the ideal (single source block)
- LTE-Streaming - 5053&Ideal: this tab provides the results for all 72 LTE streaming cases for both codes with the following details:
    - Supported Media rate: the media bitrate supported by RFC 5053 that fulfils the criteria
    - T: The symbol size in bytes for the RFC 5053 code
    - N: The total number of symbols for the RFC 5053 code
    - G: the number of symbols for each packet for the RFC 5053 code
    - K: the number of source symbols for a DASH segment for RFC 5053
    - Supported Media rate: the media bitrate supported the ideal (single source block) code that fulfils the criteria
    - T: The symbol size in bytes for the ideal (single source block) code
    - N: The total number of symbols for the ideal (single source block) code
    - G: the number of symbols for each packet for the ideal (single source block) code
    - K: the number of source symbols for a DASH segment the ideal (single source block) code
    - Difference % media rate: the degradation in media rate of the RFC5053 compared to the ideal (single source block) code
- UTRAN-Streaming -5053&Ideal: this tab provides the results for all 18 UMTS streaming cases for both codes with the following details:
    - Performance: the media bitrate supported by RFC 5053 that fulfils the criteria
    - T: The symbol size in bytes for the RFC 5053 code
    - N: The total number of symbols for the RFC 5053 code
    - G: the number of symbols for each packet for the RFC 5053 code
    - K: the number of source symbols for a source block for RFC 5053
    - Performance ideal: the media bitrate supported by ideal code that fulfils the criteria
    - K: the number of source symbols for a source block for the ideal code
- UTRAN-Download - 5053&Ideal: this tab provides the results for all 36 UMTS download cases for both codes with the following details:
    - FEC Overhead: The FEC symbol overhead necessary to fulfil the criteria in percent for RFC 5053

- T: The symbol size in bytes for the RFC 5053 code
- Kt: the total number of source symbols for the test cases for RFC 5053
- Z: the total number of source blocks for the RFC 5053 code
- Ns: the number of subblocks for the RFC 5053 code
- G: the number of symbols for each packet for the RFC 5053 code
- FEC Overhead Ideal (single SB): The FEC symbol overhead necessary to fulfil the criteria in percent when using an ideal code with a single source block.
- T: The symbol size in bytes for the ideal (single source block)
- Kt with G=1: the total number of source symbols for the test cases for the ideal (single source block) code
- Z: the total number of source blocks for the ideal code (always 1)
- G: the number of symbols for each packet for the ideal code (always 1)

- Code Performance 5053: this tab provides the results for the code performance for the RFC 5053 code
- Device-Download - 5053: this tab provides the results for two configurations of the device download for all six test cases
  - Configuration 1: Focus on traces with low memory usage by applying subblocking. The memory is kept below 8 MByte
  - Configuration 2: Focus on traces without subblocking.
- Device-Streaming - 5053: this tab provides the results for two configurations of the device streaming for all nine test cases

8.2 Candidate Results

The excel sheets for these documents are attached included in the attachment `Attachment-3-Submission.zip` package. The excel sheets for the codes from the self-evaluation numbers are attached to this document as Submission-<code>.xlsx.

8.3 Verification

8.3.0 Introduction

All verification data is included in the package `Attachment-4-Verification.zip` attached to this document.

8.3.1 Verification of RS+LDPC Code

The following companies performed verification for the RS+LDPC code:

- Nomor: The consolidated excel sheet of the verification is attached to this document as Verification-RS+LDPC-nomor.xlsx. All results are verified.
- Huawei: The consolidated excel sheet of the verification is attached to this document as Verification-RS+LDPC-Huawei.xlsx. All download and streaming cases are verified.

8.3.2 Verification of 6330 Code

The following companies performed verification for the 6330 code:

- Nomor: The consolidated excel sheet of the verification is attached to this document as Verification-6330-nomor.xlsx. All download and streaming cases are verified.

- Huawei: The consolidated excel sheet of the verification is attached to this document as Verification-6330-huawei.xlsx. No non-verified test cases are reported.
- Expway: The consolidated excel sheet of the verification is attached to this document as Verification-6330-expway.xlsx.

8.3.3 Verification of Supercharged Code

The following companies performed verification for the Supercharged code:

- Nomor: The consolidated excel sheet of the verification is attached to the present document as Verification-Supercharged-nomor.xlsx. The numbers in the submission could be verified.
- Huawei: The consolidated excel sheet of the verification is attached to the present document as Verification-Supercharged-huawei.xlsx.

9 Other FEC Enhancements

9.1 Introduction

During the work item on EMM-EFEC other technologies beyond pure codes were submitted. The technologies are documented in this clause.

9.2 Graceful Degradation (GD) - FEC

9.2.1 Introduction

Graceful Degradation (GD)-FEC sub-layer is performed at the first process in Transport layer and directly applied for the received data from the Media layer. GD-FEC mechanism provides unequal error protection (UEP) technique that protects important parts of media bit stream(s) more strongly than others. FEC layer is located in the lower and at the RTP/RCTP or SRTP sections to fully protect the packets and headers generated in the upper sub-layers. GD-FEC sub-layer may also be located in above FLUTE protocol layer when it used for DASH-based streaming service over FLUTE.

9.2.2 GD-FEC Operations and Requirements

When a system needs to supply a certain level of service quality for radio impaired mobiles that are suffered frequent burst losses in broadcasting, it may adopt GD-FEC and in this case, the system should inform all mobiles about the following information using in-band and/or out-band signalling:

- The kind of media data that is be protected by GD-FEC (e.g. audio, text, …)
- The location of GD-FEC source and repair packets in the source flows
- The GD-FEC format and encoded method

The portion of the source data protected by GD-FEC (referred to as target source) should be relatively small (e.g. audio data can be chosen for the target source, because the size of audio data is typically below 10% of that of video data) so that the required amount of repair packets can be small. With an enough amount of repair data, GD-FEC may provide a media protection with a low burden in computational complexity.

In GD-FEC, the encoding delay introduced by the GD-FEC encoder may increase, but may still be good enough for typical media coding systems. However, for receivers ignoring the GD-FEC system should not add additional decoding delay. Next section describes an example of GD-FEC implementation which fulfils these requirements.

In fact, GD-FEC can be flexibly adopted as: Good receiving status mobiles can be configured to not use GD-FEC decoding, however, bad receiving status mobiles may use GD-FEC repair packets when GD-FEC is deployed in media systems at the expense of possibly increased delay.

9.2.3 GD-FEC Encoding/Decoding Examples

This section describes an example of GD-FEC implementation which fulfils the requirements described in previous section. In this example, the audio packets are selected as the target source data for GD-FEC protection.

There are two important terminologies for GD-FEC: Encoded multimedia data group (EMDG) and GD-FEC encoding group (GDEG) as shown in figure 11. EMDG is a packet group that contains a group of media data providing a certain amount of information (e.g. all media data in one picture frame unit). GDEG is defined as a group of L EMDGs where L=1, 2, … (e.g. L=4 in figure 11).

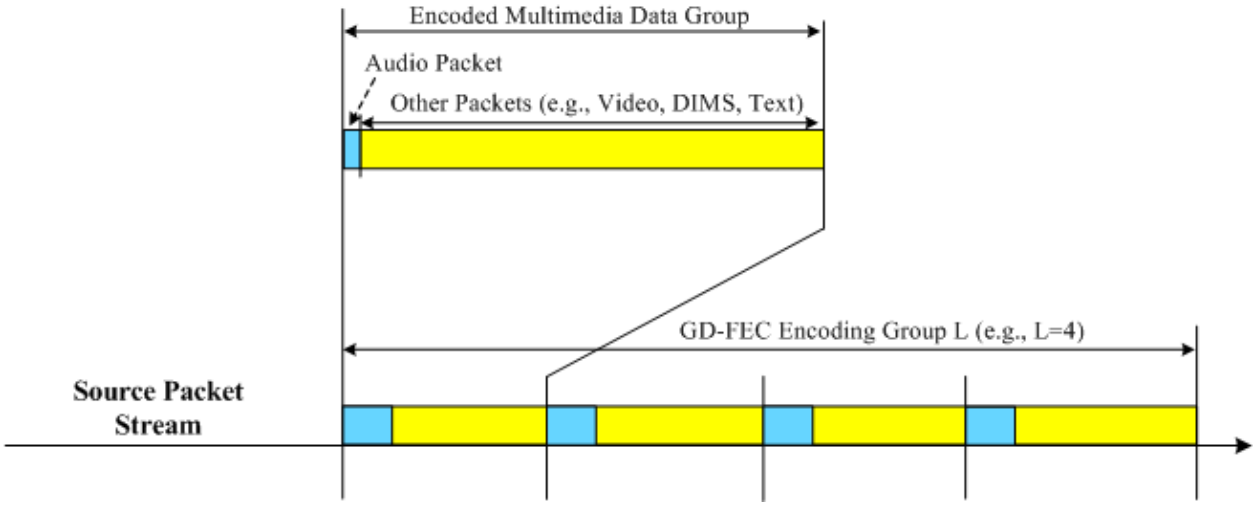


Figure 11: Encoded Multimedia Data Group (EMDG) and GD-FEC Encoding Group (GDEG)

It should be noted that, an important requirement for the GD-FEC described in previous section is the decoding delay of GD-FEC. Figure 12 shows none or minimum delay for the GD-FEC decoding whereas enough delay is yielded in the GD-FEC encoding (in the case of L=4).

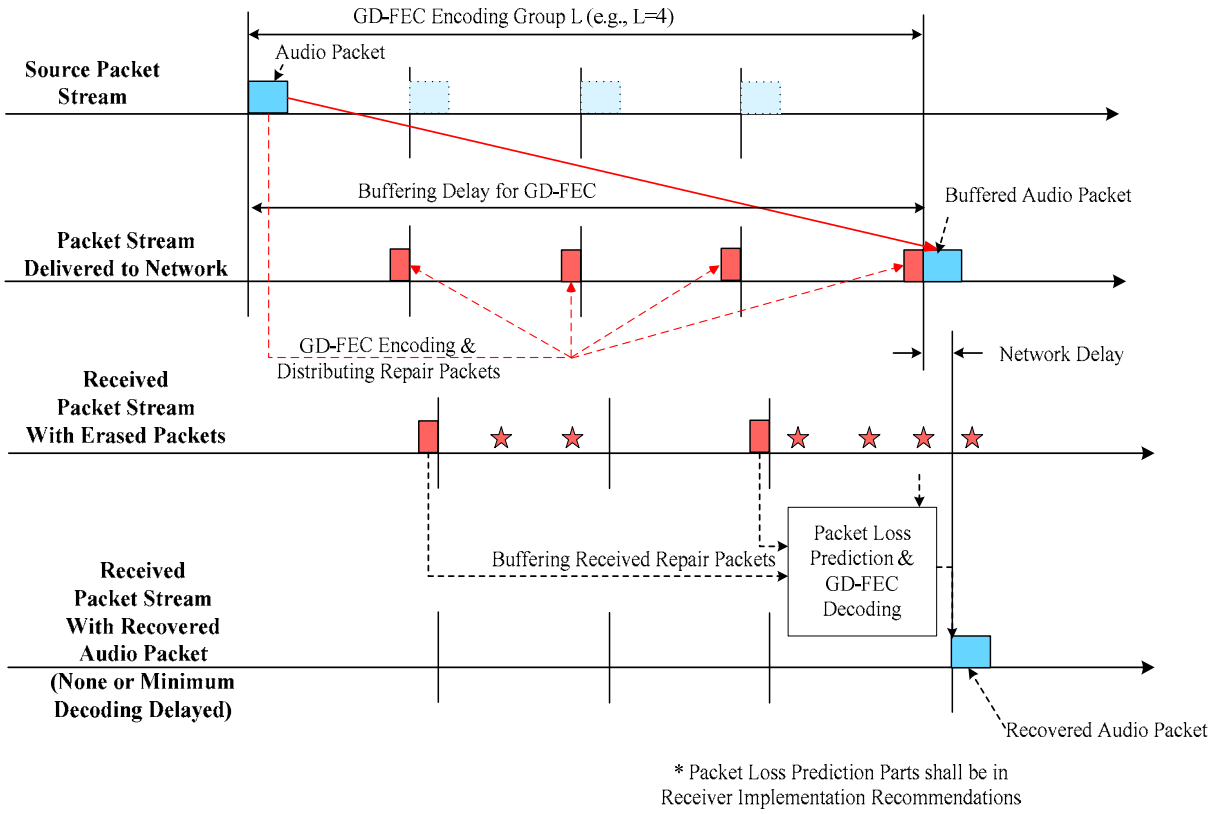


Figure 12: None or Minimum Delay GD-FEC Decoding Whereas Enough Delay Yielded by GD-FEC Encoding (e.g. L=4)

Another example of the low latency GD-FEC scheme is shown in Figure 13. In the figure, the duration of a GD-FEC encoding group is the same as that required to process a FLUTE segment file. With this alignment, an additional processing delay for the GD-FEC encoding is not required because it can be performed in the time slot of the FLUTE segment file process. Using the same reasoning, an additional processing delay for the GD-FEC decoding is also not required. The FLUTE segment packet units are smaller units than FLUTE file segments. Interleaving can be performed with these FLUTE segment packet units (e.g. predetermined random order sending). In the total streaming file for the FLUTE, the source data packets have headers and these headers are more important than others; furthermore, they can be GD-FEC target sources for stronger protection (e.g. UEP for the headers).

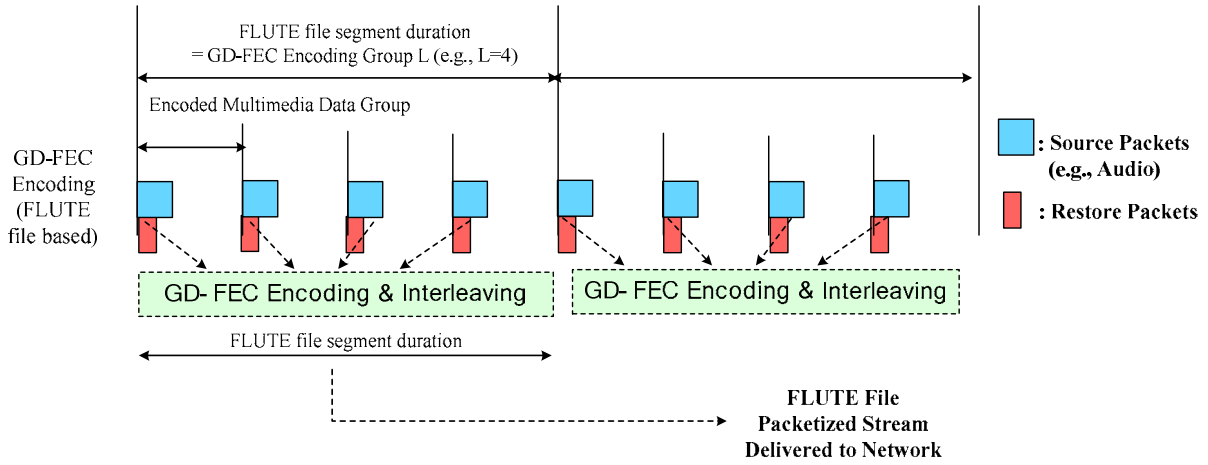


Figure 13: GD-FEC encoding delay aligned with the FLUTE segment file processing delay (e.g. the duration of the FLUTE file segment equals that of the GD-FEC encoding group)

9.2.4 Conclusion on GD-FEC

Although the study of GD-FEC during the EMM-EFEC work item produced results that showed positive benefits in some use cases and environments, it was considered that this technology could not be adopted within a normative specification at this time. Further consideration of GD-FEC may be made during later releases as a solution to various use cases including, but not limited to:

- Small segment delivery over FLUTE
- Low latency for good covered mobiles and increased latency for UEs in worse coverage
- Fast start-up
- OTA (over the air overheads)

10 Conclusions

There were significant investigations into the capabilities, performance and suitability of the qualifying EMM-EFEC candidates. The attached results and verification documents present a full view of the work performed in this area.

6330 and RS+LDPC each received significant support as MBMS Application Layer FEC (AL-FEC).

As a final conclusion of the work, no new AL-FEC code was adopted.

The existing FEC provides sufficient features and performance to ensure successful operation of MBMS User Services.

Annex A:
Simulation Conditions

A.1 Simulation Procedure for download delivery

For file downloads simulations the following assumptions are made:

- All source blocks have the same size, i.e. the size the largest source blocks (this would slightly overestimate FEC overhead but simplifies simulation code)
- The working size memory is 256KB for UTRAN MBMS and 1MB for LTE MBMS

The download procedure is:

- Generate IP packet loss transcripts, one per user, with mapping algorithm according to the access technology and the IP packet size according to the table. The transcript length has to be long enough to cover transmission of the biggest file subject to maximum simulated loss and transmission overhead to meet target success rate.
- Using the following as input: file size F, payload size P, receiver memory size WS. Then compute the number of source blocks Z and their size in symbols KT, the number of symbols per packet G (always 1 for Ideal) according to the following schemes per FEC:
  - In the case of Ideal code, there is always a single source block with symbol size T=P with a total of K=ceil(F/P) symbols.
  - In the case of Raptor, the parameters are computed using Section 9.1 Block Partitioning Algorithm of RFC 5052.
  - In case of other codes, the algorithm for computing the different parameters should be provided

- For each user U do
  - Encoding symbol index I = 0
  - Until all Z source block are received
    - For each Z source block
      1. Add a received symbol of ESI I for the block if not lost according to lost transcript A for user
      2. Move loss transcript pointer to next item
      3. If the block is not decoded and number of received symbols is equal or bigger than K*T do:
         - Try decoding with the set of received ESIs
         - If successful, mark block as decoded, record number of symbols necessary for this block
    - I = I + 1
  - Find maximum of necessary symbols maxSymbol across Z blocks for user U, report Transmission overhead as (maxSymbol*T *Z/ F) in percent
- Rank all users according to their Transmission overhead
- If X is the target success rate, keep the last (1 – X) * N last users where N is the number of simulated users
- Report Transmission overhead (reported as FEC overhead in TR26.346) of the first user (i.e. with lowest FEC overhead) from remaining users of step 5.

A.2 Simulation Procedure for streaming delivery

The streaming simulation procedure is:

- Select a streaming service with source data rate and stream duration (24 hours)

- Generate IP packet loss transcripts, one per user, with mapping algorithm according to the access technology and the IP packet size according to the table. The transcript length has to be long enough to cover transmission of the whole stream duration.
- Compute number of symbols N per protection period for FEC under consideration (for RFC5053, this is the number of packets if G > 1)
- R = 0, the number of repair symbols
- Loop 1: Until number of segment in error E is less than target error maxE do:
    - K = N – R, where K = number of symbols for block
    - For all segments in stream do:
        - For ESI = 0 up to N-1 do:
            - If SDU is received according to loss transcript A, record ESI as received
        - Try decoding with set of received ESI
        - If not successful, E = E + 1
        - If E > maxE, R = R + 1, restart Loop 1
- Record last value of K as maxK
- Report maximum streaming rate as (G*K*T*8 / protection period) where T is the symbol size.

Annex B:
Tools for device-based evaluation

B.1 Split file into segments and generate MD5

This Unix script creates <total> smaller segments, each of size <bytes> from file <file> and names the segments with <prefix>08%d.

```
#!/bin/sh

# Split large file segments and create md5

if [ $# -eq 4 ]
then
    rm -f $3*
    head -c 231840000 $2 > /tmp/temp.mov
    split -d -a 4 -b $1 /tmp/temp.mov $3
    rm -f /tmp/temp.mov
    j=0
    for i in `ls -1 $3*`;
    do
        j=`expr $j + 1`;
        if [ $j -le $4 ]
        then
                x=`echo $3 $j | awk '{ printf ("%s%08d", $1, $2) }'`
                mv $i $x;
          MD5=`cat $x |  md5sum | awk '{ print $1 }'`
          echo '$j $MD5'
        else
               rm $i
        fi
    done
else
    echo $#  'usage: split_with_numbers.sh <bytes> <file> <prefix> <total>'
fi
```

B.2 Generate Markov Traces

The attached java code "LossGenerator.java" and "Random.java" may be used to generate the loss traces independently. The java trace file can be executed as follows:

```
java LossVectorGenerator p q gBLER bBLER subsamp n seed offset vectorfile
```

with:

```
p (transition probability from good to bad state)

q (transition probability from bad to good state)

gBLER (BLER for the good markov state)

bBLER (BLER for the bad markov state)

subsamp (subsampling for markov trace)

n (length of the vector to be generated)

seed (for the prng)

offset (iterate n times before generating the vector)
```

```
vectorFile (file name where to output the vector)
```

Table B.1 provides the instructions how to generate the error traces for the streaming test cases.

Table B.1 Markov Trace generation for streaming test cases

Test Case	Error conditions	Test Script parameters
LS21	Markov, 3 km/h, 20%	`0.0461 0.1680 0.0016 0.8920 1 180000 0 0 errortrace_ls21.txt`
LS49		`0.0461 0.1680 0.0016 0.8920 1 180000 0 0 errortrace_ls49.txt`
LS24		`0.0461 0.1680 0.0016 0.8920 1 180000 0 0 errortrace_ls24.txt`
LS33	Markov, 120 km/h, 5%	`0.2707 0.7095 0.0000 0.1954 1 180000 0 0 errortrace_ls33.txt`
LS50		`0.2707 0.7095 0.0000 0.1954 1 180000 0 0 errortrace_ls50.txt`
LS36		`0.2707 0.7095 0.0000 0.1954 1 180000 0 0 errortrace_ls36.txt`
LS45	Markov, 120 km/h, 20%	`0.3560 0.6329 0.0972 0.4040 1 180000 0 0 errortrace_ls45.txt`
LS51		`0.3560 0.6329 0.0972 0.4040 1 180000 0 0 errortrace_ls51.txt`
LS48		`0.3560 0.6329 0.0972 0.4040 1 180000 0 0 errortrace_ls48.txt`

Table B.2 provides the instructions how to generate the error traces for the download test cases. The <length> corresponds to the Length in the table and the <offset> is the trace number minus one multiplied by the length. The trno runs from 1 to S.

Table B.2 Markov Trace generation for download test cases

Test Case	S	Length	PCAP file
LD60	1	2000000	`0.0461 0.1680 0.0016 0.8920 1 <length> 0 <offset> error_trace_ld60_<trno>.pcap`
LD108	20	3400	`0.2707 0.7095 0.0000 0.1954 1 <length> 0 <offset> error_trace_ld108_<trno>.pcap`
LD109	5	150000	`0.2707 0.7095 0.0000 0.1954 1 <length> 0 <offset> error_trace_ld109_<trno>.pcap`
LD110	1	2000000	`0.2707 0.7095 0.0000 0.1954 1 <length> 0 <offset> error_trace_ld110_<trno>.pcap`
LD118	20	3400	`0.3560 0.6329 0.0972 0.4040 1 <length> 0 <offset> error_trace_ld118_<trno>.pcap`
LD119	5	150000	`0.3560 0.6329 0.0972 0.4040 1 <length> 0 <offset> error_trace_ld119_<trno>.pcap`

B.3 Root access for Galaxy S2

Here is procedure to root the Samsung S2:

- http://forum.xda-developers.com/showthread.php?t=1501719

Once the phone is rooted, to turn on performance mode and disable the second CPU core:

- `cd /sys/devices/system/cpu/cpu0/cpufreq`
- `cat scaling_governor`
  - this will tell the current mode (on-demand or performance)
- `echo performance > scaling_governor`
  - turn on performance mode. `echo ondemand` *to turn off*
  - NOT a sticky command i.e. value resets to ondemand after reset
    - note: *performance* mode will keep it at 1.5GHz, even at idle
    - In *ondemand* mode - at idle, without a data transfer or anything else running on the device, cpu0 should be running at much lower speed
- `cat scaling_cur_freq`
  - display current clock frequency in kHz
- `cd /sys/devices/system/cpu/cpu1/cpufreq`

  - to check the settings for cpu1
    - NOTE: if core 1 is not on, the cpufreq directory won't exist
- `cd /sys/devices/system/cpu/cpu1; cat online`
  - if it outputs 1, cpu1 is still up
- `echo 0 > /sys/devices/system/cpu/cpu1/online`
  - shuts a given cpu down
- `chmod 444 /sys/devices/system/cpu/cpu1/online`
  - ensures that the cpu is not restarted again (needs to be finally verified)

B.4 Time Command on Android Device

To enable the time command on an android device, the Busybox needs to be installed.

- ARM pre-compiled busybox can be downloaded from http://busybox.net/downloads/binaries/1.19.0/ (the ARMv6l works well on Android).
- Then push it on the phone by
  - renaming it 'time': adb push busybox-armv6l /data/local/tmp/time
  - make sure it's executable (adb shell chmod 0777 /data/local/tmp/time).

B.5 USB tethering of Android Devices

B.5.1 Requirements

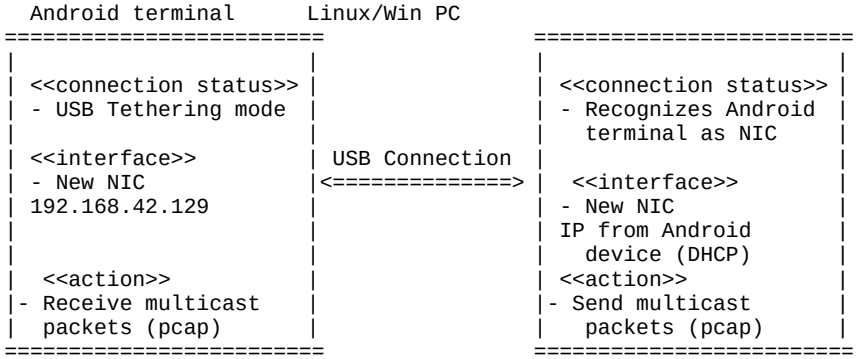
Android device running 2.2 Froyo or higher

B.5.2 Enable USB tethering on Android

- Switch ON "Tethering" option in "Setting->Wireless and Networks.

You can check the IP address of the newly created interface using the "`adb`" tool from the Android SDK. Once in the Android shell use the "`netcfg`" command. The IP address should be "192.168.42.129" (Hardcoded in Android source code).

B.5.3 Network structure



B.6 Play a PCAP

B.6.1 Windows

In order to play a PCAP file on a Windows based host, one can use the following tools:

- `http://www.colasoft.com/packet_player/`

B.6.2 Unix & Win32/Cygwin

In order to play a pcap file on a Unix based host, one can use the following tools:

- TCP Replay as available here: `http://tcpreplay.synfin.net/`
- or here as source `http://sourceforge.net/projects/tcpreplay/`

B.7 Android SSH server

An SSH server for Android is SSHDroid available from Google Marketplace. Search for 'SSHDroid'. Once installed, make sure to configure port 2222 in its settings. For some reason when SSHDroid defaults to port 22 when running in root mode, it is not possible to `ssh` in. Port 2222 has no such restriction.

B.8 Verify Segment Decoding

This tool is attached in source code in `verifysegm.zip` with compilation instructions for Android.

The tool reads from stdin a repeated sequence

```
[ TOI (32-bit) |  length (32-bit) | <sequence of segment bytes>]
```

where `TOI` is the segment Transport Object Identifier followed by the length of the decoded segment in bytes and the actual recovered segment data. `TOI` and `length` are in network-byte order.

For each such triplet, the output is

```
<TOI as a human readable integer> <one space> <human readable hex MD5> <newline>
```

The output is human readable, unlike the input. Exactly one such line is printed to `stdout` per TOI (assuming the TOI is received a single time).

Example input in hex:

```
00 00 00 01 00 00 00 03 a0 a1 a2 00 00 00 02 00 00 00 01 b0

(end of file after that.)
```

This corresponds to two objects, first having TOI 1, and a length of 3 bytes, the file content being (in hex) a0 a1 a2, and the second one being TOI 1 the file containing a single byte b0.

The output produced by that should be:

```
1 b33326d4c1d789e9651d526f420b6801

2 ec655b6da8b9264a7c7c5e1a70642fa7
```

and no other line.

Annex C:
Change history

Change history							
Date	**TSG #**	**TSG Doc.**	**CR**	**Rev**	**Subject/Comment**	**Old**	**New**
2012-06	56	SP-120225			Presented at TSG SA#56 (for information)		1.0.0
2013-03	59	SP-130018			Presented at TSG SA#59 (for approval)	1.0.0	2.0.0
2013-03					Version for Release 11	2.0.0	11.0.0
2014-09	65				Version for Release 12	11.0.0	12.0.0
2015-12	70				Version for Release 13	12.0.0	13.0.0

Change history							
Date	**Meeting**	**TDoc**	**CR**	**Rev**	**Cat**	**Subject/Comment**	**New version**
2017-03	75					Version for Release 14	14.0.0
2018-06	80					Version for Release 15	15.0.0

History

Document history		
V15.0.0	July 2018	Publication