ECN for 3GPP RRC

How ECN could be applied when specifying
3GPP RRC messages

Markku Turunen
ETSI STF 169

Contents

- Introduction
- RRC message requirements
- Problems with the current definitions
- Solutions
  - Addition of new messages
  - New message versions
  - Non-critical extensions
  - Spare values
  - Size optimiztions

Introduction

- Purpose
  - To show what kind of improvements can be made concerning the current RRC message definitions
  - To show how the ASN.1 ECN (Encoding Control Notation) can be used to simplify and clarify message definitions

- Note
  - Minor details of the ECN examples might be revised in the future
  - The ECN examples use features presented in the ballot comments

RRC message requirements

- Encoded messages must be compact
- Extensibility
  - New messages
    - addition shall have no effect on old messages
    - minimum overhead for new messages
  - New critical IEs => new message versions
    - receiver must be able to detect critical extensions
    - minimum overhead for new message versions
  - New non-critical IEs
    - minimum overhead for new non-critical IEs
  - New values in IEs
- Compression/specialization of encodings

Problems with the current definitions

- The vanilla ASN.1 + PER do not fulfil the requirements
  - generic extensibility => more bits consumed for control information, no good

⇒ The requirements affect ASN.1 definitions
  - Encoding specific issues are visible in message definitions

- The first message versions are manageable
- When messages are extended their complexity grows a lot

Solutions

- Separate abstract message contents and encoding specific issues
- The following slides show solution examples for the presented requirements

Addition of new messages

- Requirements
  - It must be possible to add new channel specific messages
  - Messages must be identified using as few bits as possible

- Current solution
  - There are explicit placeholders

```
-- v1
DL-DCCH-MessageType ::= CHOICE {
    activeSetUpdate         ActiveSetUpdate,
    cellUpdateConfirm       CellUpdateConfirm,
    -- etc
    extension               NULL
}
```

```
-- v2
DL-DCCH-MessageType ::= CHOICE {
    activeSetUpdate         ActiveSetUpdate,
    cellUpdateConfirm       CellUpdateConfirm,
    -- etc
    extension               CHOICE {
        newMessage3         NewMessage3,
        newMessage4         NewMessage4,
        extension           NULL
    }
}
```

Addition of new msgs - Improvements

- Simplify ASN.1 definitions
  - Remove encoding oriented parts from abstract message definition
  - Create a simple message wrapper type

- Separate message specific encoding definitions and generic encoding definitions
  - All the messages types share the same structure => specify it only once
  - Message identification is unique for each message type
    - fine-tune encoding if necessary

Addition of new msgs - ASN.1 definitions

- Simple ASN.1 definitions
- Special "extension" component
  - Used as a flag indicating presence of an extended message

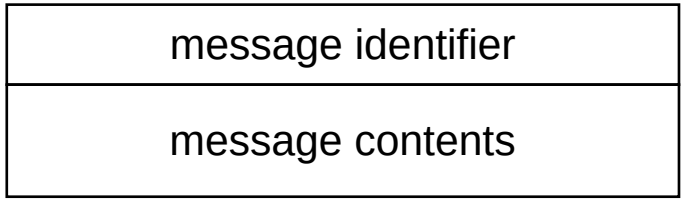
```
-- v1
DL-DCCH-MessageType ::= CHOICE {
    activeSetUpdate         ActiveSetUpdate,
    cellUpdateConfirm       CellUpdateConfirm,
    -- etc
    extension       NULL
}
```

```
-- v2
DL-DCCH-MessageType ::= CHOICE {
    activeSetUpdate         ActiveSetUpdate,
    cellUpdateConfirm       CellUpdateConfirm,
    -- etc
    newMessage3             NewMessage3,
    newMessage4             NewMessage4,
    extension               NULL
}
```

Addition of new msgs - Wanted encoding

- Two things must be specified
  - What are the bit-fields that encoding of a message type is composed of?
  - How the bit-fields are encoded?

The default PER encoding structure

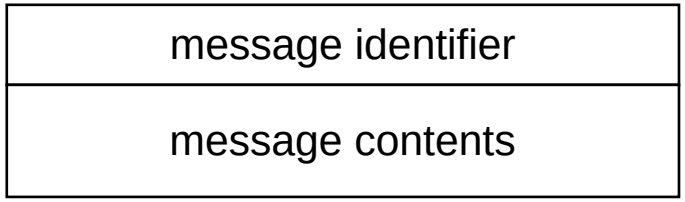


The default PER encoding for the fields

Fixed length integer field

The message determined by the id field

Wanted encoding structure



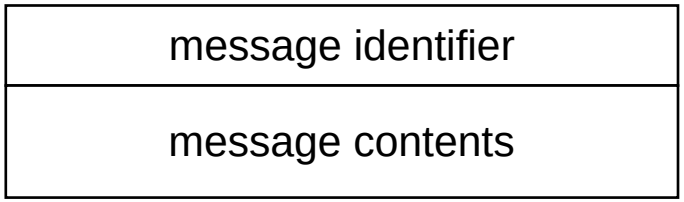
Wanted encoding for the fields

Variable length integer field optimized for each message type

The message determined by the id field

Addition of new msgs - Generic structure

- Generic encoding structure for message types



Variable length identifier field

The message determined by the id field

- Encoding of message identifier varies for different message types
- Encoding of message contents varies for different message types
- However the general structure is the same

⇒ Capture the general structure in a generic parameterized encoding structure

⇒ Provide message type specific parts as parameters

Addition of new msgs - Replacement

- The steps from a specific message type to generic structure are as follows:
- Step 1: Replace the encoding structure of a message type with a generic encoding structure

  - Replacement maps fields of DL-DCCH-MesageType to the fields of RRC–MessageType{<>}

- Step 2: Specifiy how fields of RRC-MessageType{<>} are encoded

Addition of new msgs - Replacement

- Encoding definition for a message type
  - Replace encoding structure (bit-fields) with the generic structure
    - "#DL-DCCH-MessageType" will be the value of the "#Msgs" parameter
  - Provide parameters for the generic encoding object
    - How message identifier is encoded

```
dL-DCCH-MessageType-encoding #DL-DCCH-MessageType ::= {
    REPLACE     ENTIRE-STRUCTURE
            WITH            #RRC-MessageType-struct
            ENCODING        dL-DCCH-MessageType-struct-encoding
}
```

Use this structure

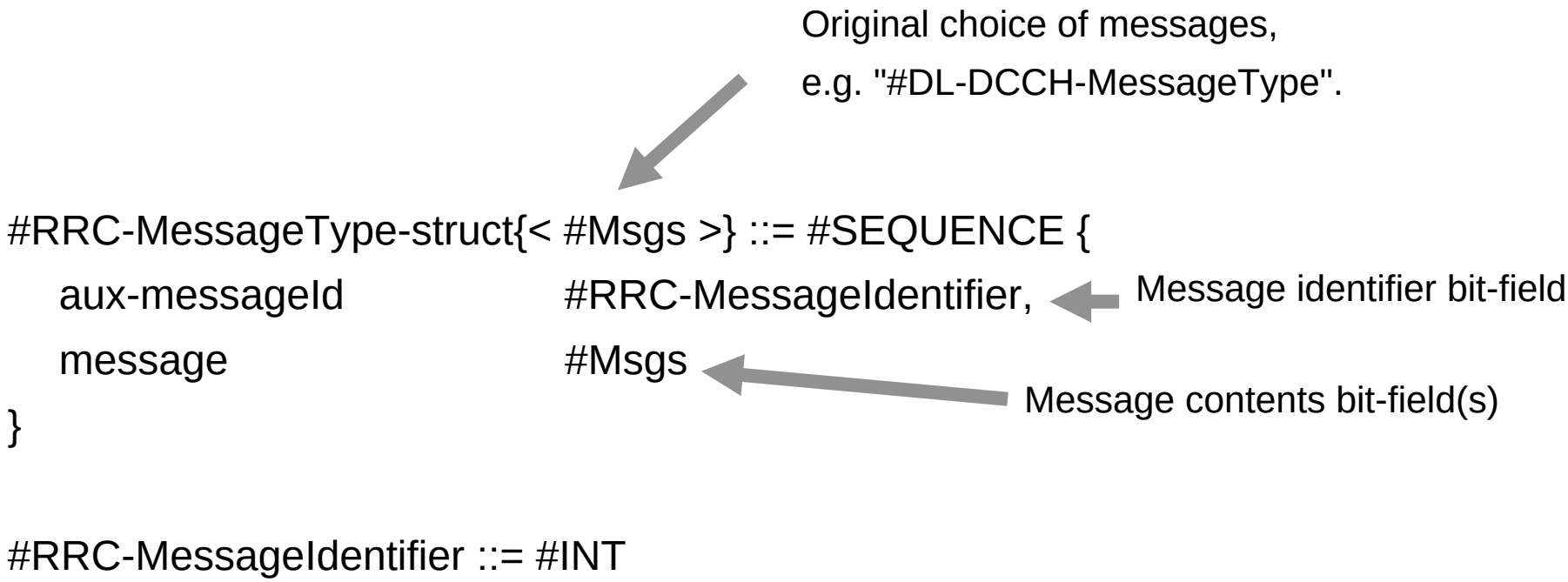
Use this encoding object

```
dL-DCCH-MessageType-struct-encoding{< #Msgs >} #RRC-MessageType-struct{< #Msgs >} ::=
    rrc-MessageType-struct-encoding{< #Msgs, rrc-messageIdentifier-2-encoding >}
```

This encoding object specifies how message id is encoded

Addition of new msgs - Generic structure

- "#RRC-MessageType-struct" is an encoding structure
  - It specifies the bit-fields that comprise encoding of "RRC-MessageType"



Addition of new msgs - Generic structure

- The "rrc-MessageType-struct-encoding" is an encding object
  - It specifies how the bit-fields of "#RRC-MessageType-struct" are encoded

```
rrc-MessageType-struct-encoding{< #Msgs, #RRC-MessageIdentifier : msgId-encoding >}
    #RRC-MessageType-struct{< #Msgs >} ::=
{
    ENCODE STRUCTURE {
        aux-messageId           msgId-encoding,
        message                 choice-with-aux-determinant-encoding{< aux-messageId >}
    }
    WITH PER-BASIC-UNALIGNED
}
```

Encoding for the message identifier

The message identifier is the selector for the message

The rest is encoded using PER

Addition of new msgs - Message id

- "rrc-messageIdentifier-2-encoding" is an encoding object for message ids

```
rrc-messageIdentifier-2-encoding #RRC-MessageIdentifier ::= {
    USE             #BITS
    MAPPING TO BITS {
        0 .. 1      TO  '00'B .. '01'B,     <- 00 - Message1, 01 - Message2
        2           TO  '1'B                <- 1  - extensions
    }
    WITH self-delimiting-bits
}
rrc-messageIdentifier-2-2-encoding #RRC-MessageIdentifier ::= {
    USE             #BITS
    MAPPING TO BITS {
        0 .. 1      TO  '00'B .. '01'B,     <- 00  - Message1, 01  - Message2
        2 .. 3      TO  '100'B .. '101'B,   <- 100 - Message3, 101 - Message4
        4           TO  '110'B              <- 110 - extensions
    }
    WITH self-delimiting-bits
}
```

Addition of new msgs - Summary

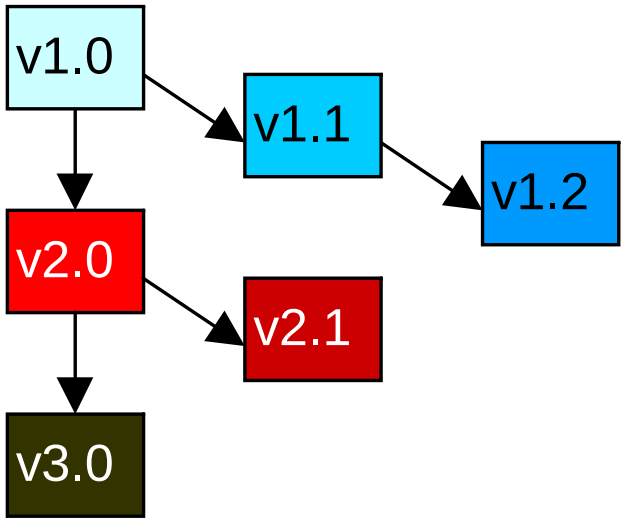
- ASN.1 definitions can be simplified
- Generic encoding structure for the messages
  - #RRC-MessageType-struct
- Generic encoding for the bit-fields of the message structure
  - rrc-MessageType-struct-encoding
- Encoding of message identifiers can be separated and specialized as wanted
  - rrc-messageIdentifier-2-encoding

! Only small amount of message type specific ECN definitions are needed

- dL-DCCH-MessageType-encoding - use replacement
- dL-DCCH-MessageType-struct-encoding - encoding for replacement

New message versions

- Addition of new critical IEs creates new message versions
  - E.g. v1.0 => v2.0 => v3.0
- Addition of new non-critical IEs creates new message sub-versions
  - E.g. v1.0 => v.1.1 => 1.2



New message versions

- Current solution
  - Placeholders for critical and non-critical IEs

```
ActiveSetUpdate ::= CHOICE {
    ies             SEQUENCE {
        ies                 ActiveSetUpdate-v1-IEs,
        nonCriticalExtensions
                            SEQUENCE {} OPTIONAL
    },
    criticalExtensions  NULL
}
```

```
ActiveSetUpdate ::= CHOICE {
    ies                 SEQUENCE {                          <-- v1.0
        ies                     ActiveSetUpdate-v1-IEs,
        nonCriticalExtensions SEQUENCE {                    <-- v1.1
            ies                 ActiveSetUpdate-v1-v2-exts,
            nonCriticalExtensions
                                SEQUENCE {} OPTIONAL
        } OPTIONAL
    },
    criticalExtensions   CHOICE {
        ies             SEQUENCE {                          <-- v2.0
            ies                 ActiveSetUpdate-v2-exts,
            nonCriticalExtensions
                                SEQUENCE {} OPTIONAL
        },
        criticalExtensions      NULL
    }
}
```

New msg versions - Improvements

- Simplify ASN.1 definitions
  - Make versioning explicit
  - Group all sub-versions in a single version structure

- Make encoding definitions for messages and message versions
  - All messages share the same structure
  - Identification of message versions is the same for all the messages

  $\Rightarrow$ Specify them only once

New msg versions - ASN.1 defs

- Simple ASN.1 definitions
  - Explicit versioning + flag for unknown message versions

```
-- Message
ActiveSetUpdate ::= CHOICE {
    v1                  ActiveSetUpdate-v1,
    criticalExtension NULL
}

-- Message version
ActiveSetUpdate-v1 ::= SEQUENCE {
    ies                 ActiveSetUpdate-v1-IEs
}
```

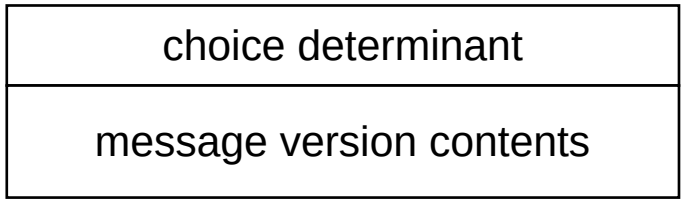
```
-- Message
ActiveSetUpdate ::= CHOICE {
    v1                  ActiveSetUpdate-v1,
    v2                  ActiveSetUpdate-v2,
    criticalExtension  NULL
}

ActiveSetUpdate-v1 ::= SEQUENCE {
    ies                 ActiveSetUpdate-v1-IEs,          <- v1.0
    ext-v2              ActiveSetUpdate-v1-v2-exts       <- v1.1
                        OPTIONAL
}

ActiveSetUpdate-v2 ::= SEQUENCE {
    ies                 ActiveSetUpdate-v2-exts          <- v2.0
}
```

New msg versions - Generic structure

- Generic structure for different message versions



Variable length determinant field

The message version determined by the choice determiant field

- Encoding of version identifier is the same for all the messages
- Encoding of message version contents varies for different message versions

⇒ Make a generic #CHOICE structure which is distinct from other #CHOICE structures
  - has its own "coloring"

⇒ Replace message encoding structures with the special structure

⇒ Make one encoding object for the new encoding structure
  - affects all messages

New msg versions - "Colors" and "shapes"

- The steps from a specific message to generic structure are as follows:
- Step 1: Mark messages as having the property ("color") of "RRC-Message-v1"

- Step 2: Change the encoding structure ("shape") to be the wanted generic encoding structure

- Step 3: Specify encoding for the generic encoding structure

Step 1: "Coloring"

- Mark messages to be replaced with "#RRC-Message-v1"
  - One encoding object for "#RRC-Message-v1" => encoding for messages

```
GENERATES-AND-EXPORTS
    REPLACE  #CHOICE WITH #RRC-Message-v1
    IN    #ActiveSetUpdate,
          #ActiveSetUpdateComplete,
          #ActiveSetUpdateFailure
          -- etc
FROM PDU-definitions;
```

Use this structure and whatever encoding is specified for it

- "#RRC-Message-v1" is a synonym for #CHOICE

```
#RRC-Message-v1 ::= #CHOICE
```

Step 2: "Shaping"

- "#RRC-Message-struct" will replace "#RRC-Message-v1"

```
#RRC-Message-struct{< #MsgsVersions >} ::= #SEQUENCE {
    aux-version          #RRC-MessageVersionDeterminant,
    messageVersion       #MsgsVersions
}

#RRC-MessageVersionDeterminant ::= #INT
```

Version identifier bit-field

Message version contents bit-field(s)

Step 2: "Shaping"

- "rrc-Message-v1-encoding" is an encoding object
  - It specifies how the encoding structure "#RRC-Message-v1" is replaced by "#RRC-Message-struct"
  - The encoding object "rrc-Message-struct-v1-encoding" specifies how the replaced structure is then encoded

```
rrc-Message-v1-encoding #RRC-Message-v1 ::= {
    REPLACE ENTIRE-STRUCTURE
        WITH            #RRC-Message-struct
        ENCODING    rrc-Message-struct-v1-encoding
}
```

Step 3: Encoding for the new "shape"

- "rrc-Message-struct-v1-encoding" is an encoding object
  - It specifies how bit-fields of "#RRC-Message-struct" are encoded
  - Only one message version has been specified

```
rrc-Message-struct-v1-encoding{< #MsgVers >} #RRC-Message-struct{< #MsgVers >} ::= {
    ENCODE STRUCTURE {
        -- Components
        aux-version             rrc-MessageVersionDeterminant-v1-encoding,
        messageVersion          choice-with-aux-determinant-encoding{< aux-version >}

        -- Structure
        STRUCTURED WITH     per-seq-encoding
    }
}
```

New msg versions - Version determinant

- "rrc-MessageVersionDeterminant-v1-encoding" is an encoding object for message versions

```
rrc-MessageVersionDeterminant-v1-encoding #RRC-MessageVersionDeterminant ::= {
        USE     #BITS
        MAPPING TO BITS {
                0  TO '0'B,         -- v1
                1  TO '1'B          -- unknown critical extension
        }
        WITH self-delimiting-bits
}

rrc-MessageVersionDeterminant-v2-encoding #RRC-MessageVersionDeterminant ::= {
        USE     #BITS
        MAPPING TO BITS {
                0  TO '0'B,         -- v1
                1  TO '10'B,        -- v2
                2  TO '11'B         -- unknown critical extension
        }
        WITH self-delimiting-bits
}
```

Non-critical IEs - Step 1: "Coloring"

- Mark message versions to be replaced with "#RRC-MessageVersion"
  - One encoding object for "#RRC-MessageVersion" => encoding for message versions
  - Non-critical extensions in message versions are encoded using the special encoding

```
GENERATES-AND-EXPORTS
    REPLACE #SEQUENCE WITH #RRC-MessageVersion
    IN    #ActiveSetUpdate-v1,
          #ActiveSetUpdateComplete-v1,
          #ActiveSetUpdateFailure-v1
          -- etc
FROM PDU-definitions;

#RRC-MessageVersion ::= #SEQUENCE
```

Use this structure and whatever encoding is specified for it

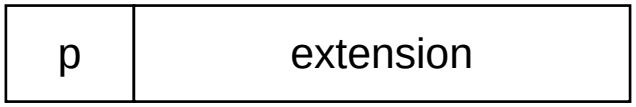
New IEs - Step 2: "Shaping"

- "rrc-MessageVersion-encoding" specifies that all optional IE groups within one message version structure (i.e. non-critical extensions) have special encoding

```
rrc-messageVersion-encoding #RRC-MessageVersion ::= {
    REPLACE OPTIONAL-COMPONENTS
                WITH            #RRC-NonCriticalExtension
                ENCODING    rrc-NonCriticalExtensionEncoding
}
```

Non-critical IEs - Step 2: "Shaping"

- "#RRC-NonCriticalExtension" is an encoding structure
  - It specifies how a non-critical extension is encoded
  - An extension is present if there are more bits in a container AND the following presence bit "p" is 1



Original message extension,

e.g. "#ActiveSetUpdate-v1-v2-exts".

```
#RRC-NonCriticalExtension{< #Ext >} ::= #SEQUENCE {
        extensionWrapper      #SEQUENCE {
                extension                #Ext    #OPTIONAL
        } #OPTIONAL
}
```

A wrapper takes care of end-of container

Encapsulated extension

Non-critical IEs - Step 3: Encoding

- "rrc-NonCriticalExtension-encoding" is an encoding object
  - It specifies that if there are more bits in a container then the extension wrapper must be present
  - The extension wrapper is encoded using normal PER => there is one presence bit for "extension"

```
rrc-NonCriticalExtension-encoding{< #Ext >} #RRC-NonCriticalExtension{< #Ext >} ::= {
        ENCODE STRUCTURE {
                extensionWrapper per-seq-encoding
                        OPTIONAL-ENCODING present-if-not-end-of-container
        }
        WITH PER-BASIC-UNALIGNED
}
```

New msg versions and IEs - Summary

- ASN.1 definitions can be simplified
- Generic encoding structure for the message versions
  - #RRC-Message-v1 + #RRC-Message-struct
  - #RRC-MessageVersion + #RRC-NonCriticalExtension
- Generic encoding for the bit-fields of the message structure
  - rrc-Message-v1-encoding
  - rrc-MessageVersion-encoding

! Only small amount of message specific ECN definitions are needed

  - One line specifying that encoding of the underlying type is replaced with special encoding

Extension of IE values

- Requirements
  - It must be possible to specify which IEs are extensible
  - Minimize bits used for extensibility

- Current solution
  - Spare values are listed in comments

```
-- TABULAR: Used range in Release99 is 1..224,
-- values 225-256 are spare values
MaxPhysChPerFrame ::= INTEGER (1..256)
```

- It must be specified what to do with received spare values during decoding
  - Treat them as illegal values; or
  - Treat them as normal allowed values and let the application take care of them

Reject spare values

- ASN.1 definitions shall contain only allowed values

```
-- Values 225-256 are spare values
MaxPhysChPerFrame ::= INTEGER (1..224)
```

- Encoding definitions shall specify the spare values

```
maxPhysChPerFrame-encoding #MaxPhysChPerFrame ::= {
    USE             #INT (1..256)
    MAPPING         ORDERED VALUES
    WITH            per-int-encoding
}
```

- If a spare value is received a decoding error is raised

Ignore spare values

- ASN.1 definitions contain also spare values, as in the current specification

```
-- Values 225-256 are spare values
MaxPhysChPerFrame ::= INTEGER (1..256)
```

- It is not allowed to send spare values but it is allowed to receive them

```
maxPhysChPerFrame-encoding #MaxPhysChPerFrame ::= {
    ENCODE-DECODE {
            USE         #INT (1..224)       -- no padding bits needed
            MAPPING     ORDERED VALUES
            WITH        per-int-encoding
    }
    DECODE-AS-IF per-int-encoding
}
```

Size optimization

- Requirements
  - There are many IEs which are optimized for size
  - Result: complex ASN.1 definitions

```
BitModeRLC-SizeInfo ::= CHOICE {
        sizeType1           INTEGER (1..127),
        sizeType2           SEQUENCE {                  -- Actual size = (part1 * 8) + 128 + part2
           part1                        INTEGER (0..15),
           part2                        INTEGER (1..7)     OPTIONAL
        },
        sizeType3           SEQUENCE {                  -- Actual size = (part1 * 16) + 256 + part2
           part1                        INTEGER (0..47),
           part2                        INTEGER (1..15) OPTIONAL
        },
        sizeType4           SEQUENCE {                   -- Actual size = (part1 * 64) + 1024 + part2
           part1                        INTEGER (0..62),
           part2                        INTEGER (1..63) OPTIONAL
        }
}
```

Size optimization

- Encoding sizes
  - sizeType1 2 + 7 bits = 9 bits
  - sizeType2 2 + 1 + 4 [+3] = 7 [10] bits
  - sizeType3 2 + 1 + 6 [+4] = 9 [13] bits
  - sizeType4 2 + 1 + 6 [+6] = 9 [15] bits

  - PER: 13 bits

Size optimization - Improvements

- Separate specification of information contents and its encoding
- Simple ASN.1 definitions

```
BitModeRLC-SizeInfo ::= INTEGER (1..5055)
```

- Encoding definitions have the complexity

```
bitModeRLC-SizeInfo-encoding #BitModeRLC-SizeInfo ::= {
    USE                     #BitModeRLC-SizeInfo-struct
    MAPPING DISTRIBUTION {
      1 .. 127              TO sizeType1,
      128 .. 255            TO sizeType2,
      256 .. 1023           TO sizeType3,
      1024 .. 5055          TO sizeType4
    }
    WITH      bitModeRLC-SizeInfo-struct-encoding
}
```

Size optimization - Improvements

- An integer is mapped to a choice
  - Value distributions are mapped to different alternatives

```
#BitModeRLC-SizeInfo-struct ::= #CHOICE {
        sizeType1           SizeType1,
        sizeType2           SizeType2,
        sizeType3           SizeType3,
        sizeType4           SizeType4
}

SizeType1 ::= #INT (1 .. 127),
SizeType2 ::= #INT (128 .. 255),
SizeType3 ::= #INT (256 .. 1023),
SizeType4 ::= #INT (1024 .. 5055)
```

Size optimization - Improvements

- Second mapping
  - Optimize segment sizes

```
bitModeRLC-SizeInfo-struct-encoding #BitModeRLC-SizeInfo-struct ::= {
    ENCODE STRUCTURE {
      -- sizeType1 is ok as is
      sizeType2         sizeType2-encoding,
      sizeType3         sizeType3-encoding,
      sizeType4         sizeType4-encoding
    }
    WITH PER-BASIC-UNALIGNED
}
```

Size optimization - Improvements

```
sizeType2-encoding #SizeType2 ::= {
    USE             #SEQUENCE {
                            part1               #INT (0..15),
                            part2               #INT (1..7)         #OPTIONAL
                    }
    MAPPING     TRANSFORMS { sizeType2-transformation }
    WITH            per-seq-encoding
}

sizeType2-transformation #TRANSFORM ::=
    USER-FUNCTION-BEGIN
        -- Actual size = (part1 * 8) + 128 + part2
        -- If part2 is absent then part2 value is considered to be 0.
    USER-FUNCTION-END
```

GSM specific parts in msgs

- There are messages with GSM specfic parts

```
InterSystemHandoverCommand-GSM-v1-IEs ::= SEQUENCE {
        -- Some IEs omitted...
        message-and-extension           CHOICE {
            gsm-Message                         SEQUENCE {},
            -- In this case, what follows the basic production is a variable length bit string
            -- with no length field, containing the GSM message including GSM padding
            -- up to end of container, to be analysed according to GSM specifications
          with-extension                        SEQUENCE {
                    messages                            GSM-MessageList
            }
        }
}
```

GSM specific parts

- Include GSM parts for example as follows

```
InterSystemHandoverCommand-GSM-v1-IEs ::= SEQUENCE {
        -- Some IEs omitted...
        message-and-extension           CHOICE {
            gsm-Message                         GSM-Message,
            -- gsm-Message contains a GSM message including GSM padding,
            -- and is to be analysed according to GSM specifications
          with-extension                        SEQUENCE {
                    messages                            GSM-MessageList
            }
        }
}

GSM-Message ::= BIT STRING
```

GSM specific parts

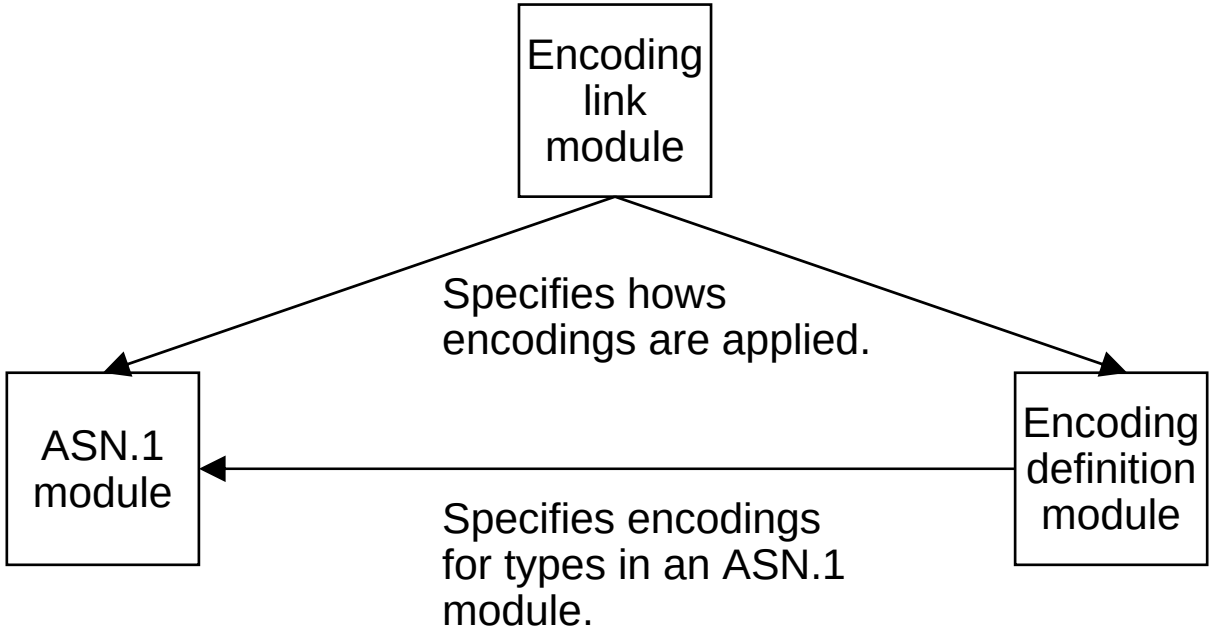
- Special encoding for "GSM-Message"
  - Is specifies that encoding of "GSM-Message" continues until the end of container

```
gsm-Message-encoding #GSM-Message ::= {
    ENCODING
            ENCODING-SPACE AS container
            CONTAINED IN end-of-encoding : NULL
}
```

- Or:
  - ASN.1 + ECN definitions for the GSM message

Putting all together

- The encoding definitions must be applied for ASN.1 definitions
  - Collect encoding objects
  - Apply encodings in an encoding link module



Encoding object set

- Collect encoding objects in one encoding object set

```
RRC-encodings #ENOCDINGS ::= {
    -- Trailing bits
    outer-encoding                          |

    -- Message type
    dL-DCCH-MessageType-encoding            |
    uL-DCCH-MessageType-encoding            |
    -- etc.

    -- Messages and message versions
    rrc-Message-v1-encoding                 |
    rrc-MessageVersion-encoding
}
```

Encoding link module

```
RRC-Encoding-Link-Definitions LINK-DEFNITIONS ::=
BEGIN

IMPORTS
    RRC-encodings                                   -- Encoding object set
FROM RRC-Encoding-Definitions;

-- Message types
ENCODE Class-definitions.DL-DCCH-MessageType
    WITH RRC-encodings COMPLETED BY PER-BASIC-UNALIGNED

-- Messages
ENCODE PDU-definitions.ActiveSetUpdate
    WITH RRC-encodings COMPLETED BY PER-BASIC-UNALIGNED

-- Message versions
ENCODE PDU-definitions.ActiveSetUpdate-v1
    WITH RRC-encodings COMPLETED BY PER-BASIC-UNALIGNED

END
```

Summary

- ASN.1 definitions can be simplified
- Encoding specific definitions can be separated from ASN.1 definitions
- Complexity of encoding can be centralized in generic definitions
  - Specify the generic definitions only once
- Message specific encoding definitions can be simple
  - Specify message specific definitions in terms of generic definitions