11

EPER : Efficient Packed Encoding Rules for ASN.1

Hiroki Horiuchi, Tetsuya Kuroki, Sadao Obana and Kenji Suzuki
KDD R&D Laboratories
2-1-15 Ohara Kamifukuoka-shi, Saitama 356, Japan
Tel.:+81 492 78 7326, email: hiroki@csg.lab.kdd.co.jp

Abstract

Data elements in OSI (Open Systems Interconnection) upper layer protocols are defined and encoded using ASN.1 (Abstract Syntax Notation One). Recently, PER (Packed Encoding Rules) have been standardized to realize more efficient ASN.1 encoding/decoding than the widely used BER (Basic Encoding Rules), by means of minimizing the length of encoded data.

This paper proposes new encoding rules, EPER (Efficient Packed Encoding Rules), to further improve PER, in which the encoding time and decoding time are 1.3 to 4.2 and 1.3 to 5.4 times, respectively, faster than those in PER, and the length of encoded data is shortened to 41% to 96% of that in PER. EPER have the following features: (1) they encode bit-aligned data and octet-aligned data in separate consecutive fields, to avoid the increase of encoding/decoding time due to many bit-shift operations and the increase of data length due to padding for octet-alignment in PER, (2) they encode integer type frequently used, so as to further eliminate redundant length octets in PER, (3) they have single transfer syntax, unlike PER that have multiple transfer syntax selected depending on communication partners. Furthermore, EPER optionally provide extended encoding rules which enable more efficient encoding when a value of a particular type is constrained by an ASN.1 sub-type definition.

EPER are highly efficient encoding rules applicable to wide range of network speeds and applications with single transfer syntax

Keywords

General Keywords: FDT-based system and protocol engineering; Languages: ASN.1;
Semantic models: Encoding Rules; Miscellaneous: Performance modeling and analysis;

1. INTRODUCTION

Data elements in OSI (Open Systems Interconnection) upper layer protocols and user parts of ISDN (Integrated Services for Digital Network) are defined and encoded using ASN.1 (Abstract Syntax Notation One) [ISO8824]. Currently, BER (Basic Encoding Rules) [ISO8825] are widely

R. Gotzhein et al. (eds.), *Formal Description Techniques IX*

used as encoding rules for ASN.1. However, BER have defects of increasing encoding/decoding time and of increasing length of encoded data, since encoded data in BER always has redundant identifier octets (ID) and length octets (LI) for each data type [Rose, Huitema].

In order to resolve these defects, two approaches have been studied. One approach is to realize fast encoding/decoding in BER by means of a hardware encoder/decoder [Bilgic] or by optimizing implementation of a software encoder/decoder [Sample]. Another approach is to introduce new encoding rules in place of BER, such as Packed Encoding Rules (PER) [ISO8825-2] which eliminate redundant IDs and LIs, application specific encoding rules [Cardoso] and Light Weight Encoding Rules (LWER) [Huitema, ISON6131] which are based on computer processing oriented data representation.

Among these encoding rules, PER have been standardized to realize more efficient encoding/decoding than BER by minimizing the length of encoded data. However, PER still have problems which are (1) increase of encoding/decoding time due to many bit-shift operations, (2) increase of data length due to padding for octet-alignment, (3) redundant LIs in encoding of Integer type and (4) multiple transfer syntax to be selected depending on communication partners.

This paper proposes new encoding rules, EPER (Efficient Packed Encoding Rules) to resolve these problems in PER. In section 2, an overview of PER and its problems are given. Then, we propose EPER to resolve the problems in PER in section 3 and their extension which enables more efficient encoding utilizing sub-type definitions in section 4, respectively. In section 5 and 6, EPER are evaluated and discussed from the viewpoint of encoding/decoding time and the length of encoded data.

2. OVERVIEW OF PER AND ITS PROBLEMS

2.1 Overview of PER

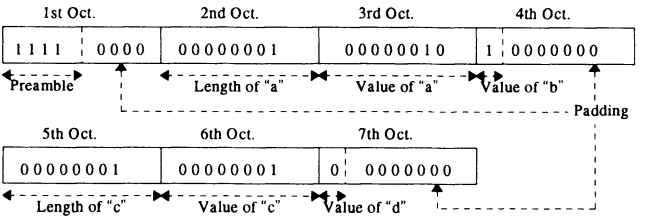
Basic Encoding Rules (BER) [ISO8825] are redundant since length octets (LI) and identifier octets (ID) for a type are always added to encoded data. In order to resolve these problems, Packed Encoding Rules (PER) [ISO8825-2] have been developed by ISO (International Organization for Standardization). PER have the following features.

(1) Identifier octets (ID) for a type are not added.
(2) Length octets (LI) are added only when the value has a variable length. LI are constructed of octet units (octet data).
(3) Encoding of Boolean type or BitString type is performed in bit units (bit data).
(4) Encoding does not distinguish between Sequence type and Set type, except that each component in Set type is encoded in order from smaller tag number (canonical order), and each component in Sequence type in order of appearance of the type in abstract syntax definition.
(5) The use of an optional or a default component in Sequence type or Set type is indicated by bit data. This bit is referred to as *preamble*.
(6) The component selected in Choice type is indicated by bit data or octet data. The data is referred to as *index*.
(7) Two kinds of transfer syntax are provided. One is *Aligned* transfer syntax, in which padding is added to the bit data in the encoded data to adjust the octet boundary. Another is

Unaligned transfer syntax, in which no padding is added.

An example of encoded data in PER is shown in Figure 1. An abstract syntax definition of Sequence type is shown in Figure 1 (a), and the values of the type shown in Figure 1 (b) are encoded in *Aligned* transfer syntax as shown in Figure 1 (c). In Figure 1 (c), the *preamble* showing the use of optional components is encoded by four bits of the first octet, the value of Boolean type is set by each first bit of the fourth octet and the seventh octet, the value of Integer type is encoded by other octets, padding is added to the first, fourth, and seventh octets, and the total encoded data length is 7 octets (56 bits).

```
A::= SEQUENCE {                                    { a 2, b TRUE, c 1, d FALSE }
  a [1] INTEGER OPTIONAL, b [2] BOOLEAN OPTIONAL,        (b) Data value
  c [3] INTEGER OPTIONAL, d [4] BOOLEAN OPTIONAL }
          (a) Abstract syntax definition
```



(c) Encoded data in *Aligned* transfer syntax

Figure 1 Example of encoded data in PER.

2.2 Problems in PER

PER have problems as shown in (i) to (iv) below which still have to be improved.

(i) Increase in encoding and decoding time in *Unaligned* transfer syntax:

Encoded data in *Unaligned* transfer syntax where no padding is added, depending on the type, contains both bit data (data in bit units) and octet data (data in octet units). Therefore, when octet data exists following bit data, components for each type in encoded data are often not aligned with the octet boundary.

In case the encoded data is not aligned with an octet boundary, frequent bit operation such as shift or logical AND must be performed on the encoded data, which causes increase of encoding/decoding time.

(ii) Increase in encoded data length in *Aligned* transfer syntax:

The length of encoded data in *Aligned* transfer syntax is increased since padding is added to bit data. For example, in the encoded data shown in Figure 1, padding of the first, fourth, and seventh octets corresponding to the *preamble* and Boolean type is a total of 18 bits, which is about 1/3 of the total encoded data length.

In fact, since Sequence type with optional or default components, or Choice type are included in data elements of ordinary OSI application layer protocol, bit data such as *preamble* or *index* often occur in the encoded data in PER, causing increase of encoded data length due to padding.

(iii) Redundancy of LI in encoding of Integer type:

A small value is ordinarily allocated to values of Integer types such as operation values of

ROS (Remote operations) or error codes in OSI application layer protocol. It is redundant that LI is always more than 1 octet in such an Integer type.
(iv) Complexity of selecting transfer syntax of several kinds:

Aligned or *Unaligned* transfer syntax must be selected depending on communication partners by the application program.

3. NEW ENCODING RULES (EPER)

We propose new encoding rules, EPER (Efficient Packed Encoding Rules), to resolve the problems in PER discussed in section 2. First, we adopt the following principles in the design of EPER :
(1) Components as bit data and components as octet data are encoded in each fields, to resolve problems (i) and (ii) in PER. The components are encoded in order of appearance of the types in the abstract syntax definition.
(2) The redundancy of LIs, which is always more than 1 octet (problem (iii)), is eliminated. For example, a value which can be expressed by 6 bits is encoded with 2 bits for LI as 1 octet.
(3) Only single transfer syntax is provided, in order to avoid selection of multiple transfer syntaxes (*Aligned* and *Unaligned*) for an individual communication partner (problem(iv)).
(4) When a value of type is constrained by the definition of a sub-type, encoding is performed by packing more efficiently utilizing these constraints. EPER optionally provide extended encoding rules utilizing sub-type definitions. The details of extended rules are discussed in section 4.

3.1 Structure of Encoded Data

As shown in Figure 2, the structure of encoded data comprises three fields: a bit field (referred to as BIF), an octet field (referred to as OCF) following the BIF, and an offset field (referred to as OFF) conditionally added before the BIF. In the BIF, bit data is set in order of appearance of type in abstract syntax definition. In the OCF, octet data is set in order of appearance of type in abstract syntax definition. The OFF indicates the length of the BIF and exists only when the length of the BIF is not determined from the abstract syntax definition, beforehand. At the end of the BIF, padding is added so that the total length of the OFF and the BIF is an integer multiple of 8 bits.

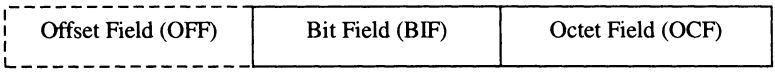


Figure 2 Structure of encoded data in EPER.

3.2 Encoding of Offset-Field (OFF)

3.2.1 Condition for Adding OFF

Conditions for adding OFF are as follows. In these cases, since the length of the BIF varies depending on a value set in the abstract syntax definition, OFF must be added.

(1) Type of value is BitString type.
(2) Type of value is Sequence type or Set type, and includes bit data of an optional or default component.
(3) Type of value is Choice type, and includes bit data of components in the choices.
(4) Type of value is Sequence-Of type or Set-Of type, and type of repeated component includes bit data.

3.2.2 Encoding of OFF

When the OFF is added, the length information for the BIF is set as follows.
(1) When the total length of BIF is 1 to 7 bits, the first bit of the first octet is set to "0"B as OFF. The remaining 7 bits in the first octet are used as BIF.
(2) When the total length of BIF is 0 bit, or 1 to 63 octets, 1 octet is allocated to OFF. In the OFF, the first and second bits are set to "10"B, and the remaining 6 bits are set to the number of octets of BIF.
(3) When the total length of BIF is 64 octets or more, 2 or more octets are allocated to OFF. In the OFF, the first and the second bits of the first octet are set to "11"B, the remaining 6 bits of the first octet are set to the number of octets of OFF, and the octets following the first octet are set to the number of octets of BIF.

3.3 Encoding Rules for Each Type

Table 1 shows encoding rules and use of fields for each type in EPER.
(1) Use of Bit-Field (BIF) and Octet-Field (OCF)

In the following cases (i) to (v), since the type of the value is bit data, the BIF is used, and bit data of each type is encoded in the order of appearance in the abstract syntax definition.
(i) Type of value is Boolean type.
(ii) Type of value is Enumerated type, and the value is within a range from 2 to 128.
(iii) Extra bit when type of the value is BitString type, and the length of the value is not an integer multiple of 8 bits.
(iv) *Preamble* when the type of the value is Sequence type or Set type and shows use of optional and default components.
(v) Type of value is Choice type, is an *index* showing a selected component and the value is within a range from 2 to 128.

In other than the above cases (i) to (v) using BIF, the OCF is used and each octet data is encoded in the order of appearance in the abstract syntax definition.
(2) Encoding for Integer type

Table 2 shows the encoding rules for integer type. The encoded data consists of a length indicator (referred to as LIN) and values which are set to OCF. The LIN is represented by 2 bits for $2^5 \leq$ value (V) $\leq 2^5-1$, 4 bits for $-2^{35} \leq V \leq -(2^5+1)$ or $2^5 \leq$ value $\leq 2^{35}-1$, 4 bits for $-2^{67} \leq V \leq -(2^{35}+1)$ or $2^{35} \leq$ value $\leq 2^{67}-1$, 1 octet for $-2^{(40\times8+4)} \leq V \leq -(2^{67}+1)$ or $2^{67} \leq V \leq 2^{(40\times8+4)}-1$ and 2 or more octets for $V \leq -2^{(40\times8+4)}$ or $2^{(40\times8+4)} \leq V$.

The following are taken into consideration to define encoding rules for Integer type.

(i) The number of bits and octets for a LIN and a value should be made as small as possible.
(ii) Integer represented in up to 64-bits notation, which is currently used in computers, should be encoded efficiently.
(iii) It should be possible to represent a very large integer value more than 64 bits.

Table 1 Encoding rules for each type in EPER

Type	Encoding Rules	Use of Field	
		BIF	OCF
Boolean	Value is encoded in 1 bit to BIF and LI is not encoded. Value is either True (1) or False (0).	M	-
Enumerated	Depending on the number N of enumerated values, encoding is not performed for N = 1, the value is set to BIF for 2 ≤ N ≤ 128, and the value is set to OCF for 129 ≤ N. LI is not encoded.	C	C
Null	Encoding is not performed except when OP or DF in Sequence type or Set type is Null type, where the presence of such components is set to BIF as *preamble*.	-	-
Integer	See Table 2.	-	M
Real	Encoded data consists of LI and the value in OCF. The value is encoded in the same rules as BER.	-	M
Octet Str. / Character Str.	Encoded data consists of LI and the value in OCF. The value is encoded in the same rules as BER.	-	M
BitString	Encoded data consists of LI and the value. LI is set to OCF, extra bit which is not an integer multiple of 8 bits of the value is set to BIF, and others are set to OCF.	C	M
Object ID	Encoded data consists of LI and the value. The value is encoded in the same rules as BER. LI and the value are both set to OCF.	-	M
Any	Encoded data consists of LI and the value in OCF. The value is the encoded data of data type separately defined by ASN.1 and have the structure of encoded data shown in Figure 2.	-	M
Sequence/Set	Encoded data consists of preamble and one or more components. When there is not any OP and DF in the types, the preamble is not encoded. For the *preamble*, a value, present(1)/absent(0), is set to BIF. The components are encoded according to rules for the type. In addition, the components in Set type are encoded in the canonical order.	C	C
Sequence-Of / Set-Of	Encoded data consists of number of repeated components and zero or more components. The number is encoded to OCF and the components are encoded according to the encoding rules for the type.	C	M
Choice	Encoded data consists of *index* and selected component. For the *index*, encoding is not performed when the number of components is one, for 2 to 128, encoding is performed on BIF in number of bit necessary for expressing the number of (components minus 1) and for 129 or more, encoding is set to OCF. The selected component is encoded according to the encoding rules for the type .	C	C

Note) BIF: Bit Field, **OCF**: Octet Field, **LI**: Length Information,
OP: optional component, **DF**: default component,
C: conditionally use the field, **M**: use the field, -: not use the field

Table 2 Encoding Rules for Integer Type

Value Range	Encoding Rules
$-2^5 \leq V \leq 2^5-1$	Encoding is performed in 1 octet. The first 2 bits are allocated to LIN, where "00"B is set. The remaining 6 bits in the octet are the value in binary notation.
$-2^{35} \leq V \leq -(2^5+1)$ $2^5 \leq V \leq 2^{35}-1$	Encoding is performed within a range from 2 to 5 octets. LIN is assigned to first two bits of the first and second octets. As the LIN, "01"B are set to first two bits of the first octet, and "nn"B representing the number of octets following the second octet is set to the first two bits of the second octet. The remaining 6 bits of the first octet and the second octet, and subsequent octets are the value in binary notation.
$-2^{67} \leq V \leq -(2^{35}+1)$ $2^{35} \leq V \leq 2^{67}-1$	Encoding is performed within a range from 6 to 9 octets. LIN is assigned to first 4 bits in the first octet. As the LIN, "10"B is set to the first and second bits and "nn"B representing the number of octets following the sixth octet is set to the third and fourth bits. The remaining 4 bits of the first octet and subsequent octets are the value in binary notation.
$-2^{(40\times8+4)} \leq V \leq -(2^{67}+1)$ $2^{67} \leq V \leq 2^{(40\times8+4)}-1$	Encoding is performed within a range from 10 to 41 octets. LIN is assigned to the entire first octet. As the LIN, "110"B showing use of 10 to 41 octets is set to the first 3 bits of the first octet, and "mmmmm"B representing the number of octets following the tenth octet is set to the remaining 5 bits. The octets following the first octet are the value in binary notation.
$V \leq -2^{(40\times8+4)}$ $2^{(40\times8+4)} \leq V$	Encoding is performed in 42 octets or more. LIN is assigned to two or more octets. As the LIN, "111"B showing use of 42 octets or more is set to the first 3 bits of the first octet, and "mmmmm"B representing the number of octets in LIN following the first octet is set to the remaining 5 bits of the first octet. The value of LIN is set in octets following the first octets. The value in Integer type is set in binary notation following these octets.

Note) V : Value of Integer type, **LIN** : Length Indicator for Integer type
"nn"B : represent "00"B to "11"B, **"mmmmm"B :** represent "00000"B to "11111"B.

3.4 Example of Encoding

Figure 3 shows an example of encoded data in EPER when the abstract syntax definition in Figure 1(a) and value in Figure 1(b) are applied. In Figure 3, the OFF is allocated to the first bit of the first octet, the *preamble* and the values of Boolean type are encoded in the following six bits of the first octet as BIF, the values of Integer type are encoded in the second and third octets as OCF, and the total encoded data length is 3 octets. In comparison with PER, the total encoded data length is considerably reduced from the example of encoded data in PER shown in Figure 1(c).

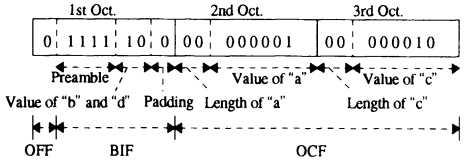


Figure 3 Example of encoded data in EPER.

4. EXTENSION OF ENCODING RULES FOR SUB-TYPE

In ASN.1, the range of value in Integer type, the range of octet length in the value of OctetString type, the range of bit length in the value of BitString type, and the range of number of components in Sequence-Of type or Set-Of type can be constrained by using the definition of sub-type, e.g. an abstract syntax "OCTET STRING (SIZE (16))" shows that the length of value is fixed 16 octets, and "INTEGER (0..128)" shows that the value is between the lower limit 0 and upper limit 128, inclusive. When such a constraint is imposed by a sub-type definition, more efficient encoding can be performed by utilizing these constraints. This section proposes the extension of encoding rules for utilizing subtypes.

4.1 Extended Encoding Rules for Sub-type Definition

Table 3 shows the extended encoding rules for Integer type, Octet String/Character String/BitString type and Sequence-Of/Set-Of types. These rules are added to and override the encoding rules described in section 3, and are applied only when a sub-type definition is used in an abstract syntax definition.

Table 3 Extended encoding rules for utilizing sub-type definition

Type	Encoding Rules	Use of Field	
		BIF	OCF
Integer with range of value	(1) When a fixed value is given, encoding is not performed. (2) When either lower limit or upper limit is given, a difference is encoded according to the rules of Integer type shown in Table 2. (3) When a range is less than or equal to 64K, LIN is not encoded and the difference between the value and the lower limit is encoded to BIF (In the case of 2 < Range ≤ 128) or OCF(In the case of 129≤ Range) with minimum bits expressing the range. (4) When a range is greater than 64K, the same rules as (2) are applied.	C	C
Octet Str. / Char. Str. / Bit String with range of length	(1)When a fixed length is given, LIN is not encoded, but only value is encoded in OCF. (2) When a range is less than 64K, the difference between the length and lower limit is encoded using the same rules as in Integer type (3) , and the value is encoded in OCF or BIF. (3) When a range is greater than or equal to 64K, the same rules as in Table 1 are applied.	C	C
Sequence-Of / Set-Of with range of number of components	(1)When a fixed number is given, the number is not encoded. (2) When the range is less than 64K, the difference between the number and lower limit is encoded using the same rules as in Integer type (3) , and components are encoded in OCF or BIF. (3) When the range is greater than or equal to 64K, the same rules as in Table 1 are applied.	C	C

Note) BIF : Bit Field, **OCF** : Octet Field, **LIN** : Length Indicator for Integer type,
C : conditionally use the field, **M** : use the field, **-** : not use the field.

(1) Principles

Principles of above extension utilizing sub-type definition are as follows:

(i) Fixed Value

When a value is constrained to a fixed value, e.g. abstract syntaxes "INTEGER (32)", encoding is not performed. When length or number of components are constrained to fixed values (e.g. abstract syntaxes "OCTET STRING (SIZE (16))" and "SEQUENCE SIZE (29) OF"), encoding of length or number of components is not performed.

(ii) Value Range

When a value or a length is constrained to a range (e.g. abstract syntaxes "INTEGER (0..128)", "OCTET STRING (SIZE (1..16))" and "SEQUENCE SIZE (2..128) OF"), the difference between the lower (or upper) limit and the value is encoded.

(2) Use of BIF (Bit-Field)

When the range of the value of Integer type is constrained to $2 \leq R \leq 128$, when the range of size in OctetString type or BitString type is constrained to $2 \leq R \leq 128$, or when the range of number of components of Sequence-Of type or Set-Of type is constrained to $2 \leq R \leq 128$, the value, size, and number are encoded as bit data in BIF.

(3) Use of OFF (Offset-Field)

When components using the above BIF are components of Choice type, optional components of Sequence type or Set type, or repeated components of Sequence-Of type or Set-Of type, OFF is added to set the length information of the BIF, since the length of bit field is not determined from the abstract syntax definition.

4.2 Example of Encoding

Figure 4 shows an example of encoded data in extended encoding rules. Figure 4(a), Figure 4(c) and Figure 4(d) are abstract syntax definition, data value and encoded data without sub-type definition, respectively. Figure 4(b), Figure 4(c) and Figure 4(e) are those with sub-type definition. In the example, length of encoded data with sub-type is shortened to 40% of that without sub-type definition.

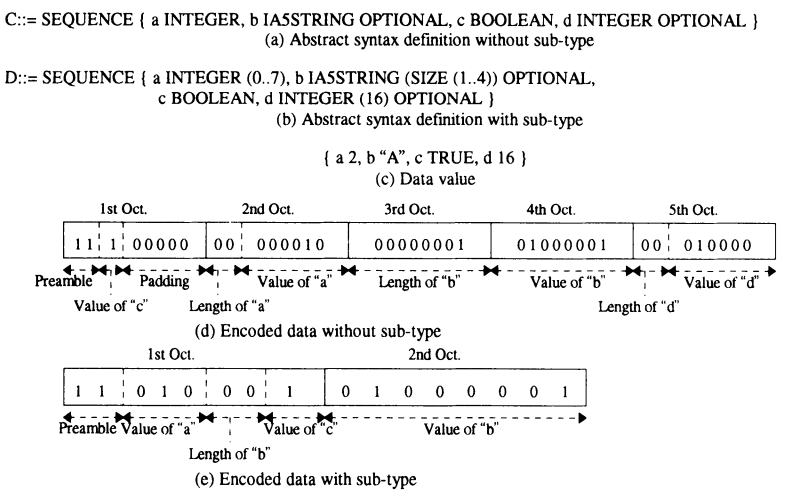


Figure 4 Example of encoded data in extended encoding rules in EPER.

5. EVALUATION OF EPER

In order to evaluate EPER, the encoding/decoding time and the data length are measured using encoding/decoding functions (or programs) generated by an ASN.1 compiler. For this purpose, we have implemented a compiler for EPER, PER and LWER. For BER, we used the compiler currently available [Hasegawa], which generates fast encoding/decoding functions in BER.

5.1 ASN.1 Compiler for EPER, PER and LWER

The compiler generates data type definition in C language according to ASN.1 data type and encoding/decoding functions for EPER, PER and LWER (Figure 5).

The generated encoding/decoding functions provide a single transfer syntax in EPER, *Aligned* and *Unaligned* transfer syntax in PER and six kinds of transfer syntax for LWER. Transfer syntaxes in LWER consist of the combination of "word-length (16, 32 or 64 bits/word)" and "byte-order (Big-Endian or Little-Endian)" [ISON6131].

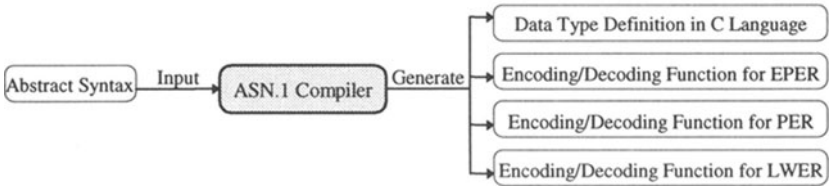


Figure 5 ASN.1 Compiler for EPER, PER and LWER.

5.1.1 Generated Data Type Definition in C Language

Figure 6 shows an example of generated data type definition in C language (referred to as C type). In the example, C types "struct Person", "struct Name" and "struct Children" are generated. The generated encoding/decoding functions encode and decode through variables with these generated C types.

```
Person ::= SEQUENCE { number INTEGER, name Name,
          maleOrFemale BOOLEAN, age INTEGER,
          single BOOLEAN, children Children OPTIONAL }
Name ::= SEQUENCE { first IA5String, last IA5String }
Children ::= SEQUENCE OF Name
```

(a) Abstract syntax definition

```
struct field { int length; char *field ; };
struct Name { struct field first; struct field last; };
struct Children { int number; struct Name *name; };
struct Person { int number; struct Name name;
          int maleOrFemale; int age;
          int single; struct Children *children; };
```

(b) Generated data type definition in C language

Figure 6 Example of abstract syntax definition and generated data type definition.

5.1.2 Procedures in Encoding/Decoding Functions

The following show the outline of procedures in encoding/decoding functions for EPER, PER and LWER, to clarify the difference among these functions. These functions adopt algorithms to make encoding/decoding for each encoding rules as fast as possible.

(1) EPER

Procedures of encoding functions are the following:

(i) Total size of encoded data and size for each BIF and OCF are calculated. Further, size of OFF is calculated according to size of BIF in case that conditions for adding OFF described in section 3.2.1 are satisfied,

(ii) Encoded data are set in BIF or OCF according to encoding rules of each data type in ASN.1, tracing the values of variables in order of appearance of the type in abstract syntax definition.

(2) PER

Encoding functions sequentially encode the values of variables from the beginning of encoded data in the order of appearance of the type in abstract syntax definition. In the encoding procedures for *Aligned* transfer syntax, padding is added to bit data. In the encoding procedures for *Unaligned* transfer syntax, bit operations such as shift are performed when octet boundary is not aligned. In both transfer syntaxes, decoding functions set values of variables, tracing the encoded data.

(3) LWER

In the case of transfer syntax with the same word-length and byte-order as those of computer used, the procedures in encoding/decoding functions are the following :

(i) The values of variables are copied to encoded data, tracing variables.

(ii) Encoded data corresponding to the value of pointers in variables are converted to offset values.

(iii) The values with variable length and optional or default components are encoded at the end of encoded data.

Decoding functions substitute the address of the beginning of encoded data to that of variables without data copy and convert offsets in the encoded data to pointer values in the variables.

In the case of transfer syntax with different word-length and/or byte-order from those of the computer used, conversion procedures for the difference are further added to the above procedures.

5.2 Evaluation Results

The encoding/decoding time and data length for EPER, PER, LWER and BER are measured. In the measurement, encoding or decoding functions are invoked ten thousand times and average elapsed time is measured on SUN SPARC 670 MP(*word-length: 32bits, byte-order:Big-Endian*) under SUN OS 4.1.3.

5.2.1 Applying Abstract Syntax Related to Problems in PER

EPER, PER, and BER are applied to the encoding of the abstract syntax definition "Example-1"

and "Example-2" shown in Figure 7. They are abstract syntax definitions which clearly show the problems of PER described in section 2.2. In these cases, there are 20 Sequence-Of components. The encoding/decoding time and encoded data length are shown in Table 4.

```
Example-1 ::= SEQUENCE OF SEQUENCE { a BOOLEAN, b INTEGER }
Example-2 ::= SEQUENCE OF SEQUENCE { a BOOLEAN, b OCTET STRING }
```

Figure 7 Abstract syntax definition for evaluation.

Table 4 Evaluation results in abstract syntax related to problems in PER

Encoding Rules	Example 1			Example 2		
(Transfer Syntax)	Enc.(ms)	Dec.(ms)	Len.(oct.)	Enc.(ms)	Dec. (ms)	Len. (oct.)
EPER	0.025	0.018	25	0.039	0.025	385
PER (*Aligned*)	0.043	0.037	61	0.051	0.034	401
PER (*Unaligned*)	0.048	0.038	44	0.165	0.136	384
BER	0.175	0.195	163	0.178	0.227	504
LWER (*16bit Big-Endian*)	0.039	0.035	84	0.069	0.057	484
LWER (*16bitLittle -Endian*)	0.057	0.062	84	0.096	0.096	484
LWER (*32bit Big-Endian*)	0.020	0.008	168	0.044	0.010	648
LWER (*32bit Little-Endian*)	0.055	0.035	168	0.105	0.051	648
LWER (*64bit Big-Endian*)	0.048	0.050	336	0.101	0.079	976
LWER (*64bit Little-Endian*)	0.094	0.113	336	0.168	0.167	976

Note) - The values for Integer type in Example-1 are less than 32.
- The length of values for Octet String type in Example-2 is 10 to 20 octets.

5.2.2 Applying OSI Application Layer Protocol

The data structures often used in OSI application protocol data units are generally classified into the following three groups.

[A] A constructor type which consists of many primitive types whose encoding lengths are short.

[B] A constructor type which includes many primitive types and data lengths in some of them is long.

[C] A constructor type which consists of primitive types whose encoding length are short and the number of primitive types is small.

According to the above classification, abstract syntax definitions of "GetResult" in OSI System Management for [A], "IM-UAPDU" in MHS (1984 version) for [B] and "GetArgument" in OSI System Management for [C] are used, and encoding/decoding time and encoding data length are shown in Table 5.

Table 5 Evaluation results in OSI application layer protocol data units

Encoding Rules (Transfer Syntax)	[A] GetResult			[B] IM-UAPDU			[C] GetArgument		
	Enc.(ms)	Dec.(ms)	Len.(oct.)	Enc.(ms)	Dec.(ms)	Len.(oct.)	Enc.(ms)	Dec.(ms)	Len.(oct.)
EPER	0.141	0.095	509	0.110	0.097	429	0.041	0.032	124
PER (*Aligned*)	0.197	0.149	602	0.139	0.127	438	0.054	0.044	134
PER (*Unaligned*)	0.339	0.273	558	0.288	0.260	430	0.075	0.058	128
BER	0.470	0.700	758	0.248	0.425	481	0.191	0.196	160
LWER (*16 bit Big-Endian*)	0.254	0.190	912	0.192	0.181	742	0.076	0.076	228
LWER(*16 bitLittle -Endian*)	0.342	0.327	912	0.263	0.290	742	0.103	0.114	228
LWER(*32bit Big-Endian*)	0.150	0.037	1,424	0.132	0.028	1,124	0.048	0.016	356
LWER(*32bit Little-Endian*)	0.351	0.167	1,424	0.267	0.112	1,124	0.100	0.052	356
LWER (*64bit Big-Endian*)	0.342	0.262	2,448	0.258	0.243	1,896	0.102	0.099	664
LWER (*64bit Little -Endian*)	0.561	0.556	2,448	0.429	0.496	1,896	0.167	0.186	664

Note) - The GetResult PDU includes 50 attribute values.
- The IM-UAPDU PDU have a body-part which includes a 200 octets character string.

5.2.3 Extended Encoding Rules for Utilizing Sub-type

An abstract syntax definition without sub-type definition, Example-3, and an abstract syntax definition with sub-type definition, Example-4, in Figure 8 are used. The encoding/decoding time and the data length are shown in Table 6.

```
Example-3 ::=  SEQUENCE OF SEQUENCE { first   INTEGER,   second IA5STRING, third INTEGER }
Example-4 ::= SEQUENCE SIZE (20) OF SEQUENCE { first   INTEGER(0..128),
                second   IA5STRING SIZE (16),   third INTEGER (32) }
```

Figure 8 Abstract syntax definition.

Table 6 Evaluation results in extended encoding rules for sub-type definition

Abstract Syntax	Enc. (ms)	Dec. (ms)	Len. (octet)
Example-3 (without sub-type)	0.059	0.043	421
Example-4 (with sub-type)	0.030	0.017	340

6. DISCUSSION

6.1 Encoding/Decoding Time and Length in EPER

According to the evaluation results in Table 4 and Table 5, the encoding time in EPER is 3.0 to 7.6 times faster than that in BER, decoding time in EPER is 3.3 to 11.0 times faster than that in BER, and the encoded data length in EPER is shortened to 15 to 77% of that in BER, with a

substantial improvement of performance.

Furthermore, the encoding time in EPER is 1.3 to 1.7 times and 1.8 to 4.2 times faster than that in *Aligned* and *Unaligned* transfer syntaxes in PER, respectively. The decoding time in EPER is 1.3 to 2.1 times and 1.6 to 5.4 times faster than that in *Aligned* and *Unaligned* transfer syntaxes in PER, respectively. The length of encoded data in EPER is shortened to 41% to 96% of that in *Aligned* transfer syntax in PER.

In general, there is the case where encoding/decoding time and data length in EPER are equal to those in PER. The case is encountered only when bit data and Integer type do not exist in encoding data. However, bit data and Integer type frequently exist in most OSI application protocol data units. Therefore, it is concluded that EPER is more effective than PER in most cases like the above.

6.2 Comparison between EPER and LWER

The encoding time and decoding time in EPER are 1.6 to 4.1 and 1.2 to 6.7 times, respectively, faster than those in transfer syntaxes in LWER, except for the decoding time in *32bit Big-Endian* transfer syntax. Moreover, the length of encoded data in EPER is shortened to 13% to 80% of that in LWER.

The decoding time in EPER is 2.0 to 3.7 times slower than that in *32bit Big-Endian* transfer syntax, where both end systems have the same byte-order and word-length, and it may be felt that the difference between decoding time in EPER and that in *32bit Big-Endian* transfer syntax is large. However, in considering the sum of encoding time in an end system and decoding time in another end system as total performance in a real environment, the sum in EPER is 1.2 to 1.4 times slower that in LWER and the difference of the sum is not so large.

Accordingly, EPER is more efficient than LWER due to considerable reduction of data length, if a low speed network where total performance depends on transmission time is used. In a high speed network, although EPER is slightly slower than LWER in the sole case where both end systems have the same byte-order or word-length, EPER is more efficient than LWER in other cases. Moreover, single transfer syntax in EPER eliminates need for knowledge of byte-order and word-length of the communication partners, complexity of selecting transfer syntax and increase of program size due to supporting multiple transfer syntax in LWER, as well as in PER.

6.3 Extension of Encoding Rules for Utilizing Subtype Definition in EPER

The effectiveness of utilizing sub-type definition depends on the kinds of constraints. The effect of the extension is significant especially when values and/or data length are constrained to fixed values, and when the range of values is constrained to be small or the absolute value of the lower limit is large.

As seen from the evaluation results in Table 6, the encoding time and the decoding time of Example-4 with sub-type definition are 1.7 times and 3.0 times, respectively, faster than that of Example-3 without sub-type definition. Further, the encoded data length is shortened to 80% of that without sub-type definition.

The extension utilizing sub-type definition is effective in many cases, except for a special

case when the range of values is large, the absolute value of the lower limit is small and all values exist around the lower limit. For example, when the range of Integer type is from 0 to 32 K and most of the actual values are from 0 to 31, the length of encoded data without the extension is one octet and that with the extension is 2 octets. This problem can be avoided by eliminating such a sub-type definition from abstract syntax definition.

6.4 Encoding/Decoding Procedures in EPER

Encoding functions in PER sequentially encode values from the beginning of encoded data, and those in EPER separately encode bit data in BIF and octet data in OCF. Although there is such a difference, the complexity of encoding/decoding functions is ensured to be almost the same through the experience of implementing a compiler for EPER and PER. For example, the program size of encoding/decoding functions in PER and EPER generated for CMIP abstract syntax are both about 6.1 Ksteps.

7. CONCLUSION

This paper proposed new encoding rules for ASN.1, EPER (Efficient Packed Encoding Rules), to improve existing PER (Packed Encoding Rules). EPER have the following features: (1) they encode bit-aligned data and octet-aligned data in separate consecutive fields, to avoid the increase of encoding/decoding time due to many bit-shift operations and the increase of data length due to padding for octet-alignment in PER, (2) they encode frequently used integer type so as to further eliminate redundant length octets in PER, (3) they have single transfer syntax, unlike PER which have multiple transfer syntax to be selected depending on communication partners. Furthermore, EPER optionally provide extended encoding rules which enable more efficient encoding when a value of a particular type is constrained by an ASN.1 sub-type definition.

The encoding time and decoding time in EPER are 1.3 to 4.2 and 1.3 to 5.4 times, respectively, faster than those in PER, and the length of encoded data is shortened to 41% to 96% of that in PER. Even in comparison with LWER, encoding and decoding time in EPER are faster than or nearly equal to those in LWER in most cases. The data length in EPER is shortened to 13% to 80% of LWER. It is concluded that the proposed EPER are highly efficient encoding rules applicable to wide range of network speeds and applications with single transfer syntax.

ACKNOWLEDGMENT

The authors wish to express sincere thanks to Dr. Hitomi MURAKAMI, Director of KDD R&D Laboratories, and Prof. Yoshiyori URANO of Waseda University for continuous guidance and kind suggestions.

8. REFERENCES

[Bilgic] M. Bilgic, B. Sarikaya : Performance Comparison of ASN.1 encoder/decoders using

FTAM, Computer Communications, vol. 16, No. 4, pp. 229-240 (1993).
[Cardoso] A. Cardoso, E. Tovar : Defining More Efficient Transfer Syntax for Application Layer PDUs in Field Bus Applications, Sigcomm Computer Communication Review, Vol. 22, No. 3, pp. 98-105 (1992).
[Hasegawa] T. Hasegawa, S. Nomura, T. Kato : Implementation and Evaluation of ASN.1 Compiler, Information Processing in Japan, Vol.15, No.2, pp. 157-167 (1992).
[Huitema] C. Huitema, A. Doghri : Defining Faster Transfer Syntaxes for the OSI Presentation Protocol, ACM Sigcomm Computer Communication Review, Vol. 19, No. 5, pp. 44-55 (1989).
[ISO8824] ISO/IEC 8824 : Specification of Abstract Syntax Notation One (ASN.1) (1989).
[ISO8825] ISO/IEC 8825 : Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1) (1989).
[ISO8825-2] ISO/IEC 8825-2 : Specification of Basic Encoding Rules for ASN.1 - Part 2: Packed Encoding Rules (1994).
[ISON6131] ISO/IEC JTC1/SC21 N6131 : Light Weight Encoding Rules (1991).
[Rose] M.T. Rose : The Open Book: A Practical Perspective on OSI, Prentice-Hall, Inc. (1990).
[Sample] M. Sample, G. Neufield : Implementing Efficient Encoders and Decoders for Network Data Representations, Proceeding of Infocom'93, pp. 1144-1153. (1993)

9. BIOGRAPHY

Hiroki Horiuchi is a research engineer of Network Management System Laboratory, KDD R&D Laboratories. Since joining the Labs. in 1985, he has been engaged in the researches on formal description techniques for communication protocol, implementation of OSI protocol and network management systems. He received the B.E. and M.E. from Nagoya University in 1983 and 1985 respectively. He received the Young Engineers Award from IEICE (The Institute of Electronics, Information and Communication Engineers), Japan in 1992.

Tetsuya Kuroki is a staff engineer of Network Management System Laboratory, KDD R&D Laboratories. He graduated from Miyakonojyo Technical High School, Miyazaki, Japan, in 1983. He joined KDD in 1983. While with the software center of the KDD headquarters, he developed database system for intelligent telephone services, such as Virtual Private Network. Since 1995 he has been engaged in the researches on Telecommunications Management Network (TMN) with the KDD R&D Laboratories.

Dr. Sadao Obana is the senior manager of Network Management System Laboratory, KDD R&D Laboratories. Since joining the Labs. in 1978, he has been engaged in the researches in the field of computer communication, database and network management systems. He received the B.E., M.E. and Dr. of Eng. Degree of electrical engineering from Keio University in 1976, 1978 and 1993 respectively.

Dr. Kenji Suzuki is a senior project manager of R&D planning group in KDD R&D Laboratories. Since joining the Labs in 1976, he worked in the field of computer communication. He received the B.S., M.E. and Dr. Eng. Degree of electrical engineering from Waseda University, Tokyo, Japan, in 1969, 1972 and 1976 respectively. From 1969 to 1970, he was with Philips International Inst. of Technological Studies, Eindhoven, The Netherlands as an invited student. He received Maejima Award from Communications Association of Japan in 1988, Achievement Award from the Institute of Electronics, Information and Communication Engineers in 1993, and Commendation by the Minister of State for Science and Technology (Persons of scientific and technological research merit) in 1995. Since 1993, he has been a Guest Professor of Graduate School of Information Systems, in the University of Electro-Communications. He is a member of IEEE.