Efficient Encoding Rules for ASN.1-Based Protocols

Nilotpal Mitra

Abstract syntax notation one (ASN.1) describes the data structures of protocol messages. In the open-systems interconnection (OSI) standards, the manner in which the data values are encoded is negotiated at the start of communications. In principle, there can be several different ways to encode the data values. Until recently, however, there was only one OSI-standardized way to do so. This standard was the basic encoding rules (BER) for ASN.1. Almost all OSI application-layer standards have specified their protocol messages in ASN.1, as have several integrated services digital network (ISDN) signaling standards based on OSI protocols. All have mandated use of the BER for their encoding. Many commercial products are also available that encode and decode protocol messages, as specified in ASN.1, to and from BER-encoded bit streams. Experience with the BER has led to criticism of their efficiency, particularly for real-time applications. Thus, there has been a move to standardize alternative encoding schemes that address these efficiency concerns. This paper introduces the alternative encoding schemes and compares them to the BER. The distinguished encoding rules (DER), canonical encoding rules (CER), packed encoding rules (PER), and lightweight encoding rules (LWER) are the alternative encoding schemes discussed.

Introduction

Abstract syntax notation one (ASN.1)[1], which describes the data structures of protocol messages, provides a standard way for application designers to specify the nature of the data exchanged without constraining how the data are encoded for transmission. In the open-systems interconnection (OSI) standards, the manner in which the data values are encoded is negotiated by the presentation-layer protocol at the start of communication. In principle, there can be several different ways to encode the data values. Until recently, however, there was only one OSI-standardized way to do so. This standard was the basic encoding rules (BER) for ASN.1.[2,3]

Almost all OSI application-layer standards have specified their protocol messages in ASN.1, as have several integrated services digital network (ISDN) signaling standards (for example, Signaling System 7, transaction capabilities, and Digital Subscriber Signaling One - Recommendation Q.932), based on open-systems interconnection (OSI) protocols. All have mandated use of the BER for their encoding. There are also many commercial products available that encode/decode protocol messages, as specified in ASN.1, to and from BER-encoded bit streams.

The BER have been standardized to ensure that there exists at least one universal encoding scheme that will permit interoperability. As with so many other design decisions made by OSI experts during early development stages, the main goal of the OSI standards was to ensure interoperability between as many vendors' products as possible.

Now that OSI has reached a mature state with many applications, the goal of interoperability appears to be within reach. So, the

Panel 1. Abbreviations, Acronyms, and Terms

ASN.1—abstract syntax notation one
BER—basic encoding rules
CER—canonical encoding rules
DER—distinguished encoding rules
ISDN—integrated services digital network
ISO—International Organization for Standardization
ITU-T—International Telecommunications Union - Telecommunication Standardization Sector
LWER—lightweight encoding rules
OSI—open-systems interconnection
PER—packed encoding rules

focus has shifted from standards development to OSI protocol efficiency. One area for efficiency improvement, which is being studied collaboratively by the International Telecommunications Union - Telecommunication Standardization Sector (ITU-T) and the International Organization for Standardization (ISO), is alternative encoding rules for data expressed in ASN.1. The BER have been criticized for being inefficient to encode and decode, and for consuming scarce bandwidth resources. Its encoding principles, while permitting great flexibility, can generate very long messages and consume much processing time to perform encoding and decoding.

The impetus to standardize other encoding schemes for OSI application-layer protocols has also arisen from many industry sectors. These include networks for air-traffic control and manufacturing process control, as both sectors have limited communication bandwidth and end-system computing resources. Another sector is the telephony-signaling networks, in which real-time efficiency concerns are paramount.

The fact that there has been, until recently, only one standardized encoding-rule set—the BER—has led to the unfortunate misunderstanding that ASN.1 and BER are inextricably linked. The supposed shortcomings of BER are mistakenly transferred to the fact that protocols are written using ASN.1. As mentioned earlier, these are two separate standards, with ASN.1 being the standard notation for specifying the type of values that are exchanged during communication. These values can be encoded in many different ways. The BER are, however, the only standard method defined at present.

In fact, the separation specified in the OSI reference model between the abstract syntax of protocol messages (the data types of the values exchanged) and the transfer syntax (the encoding of the values) permits the use of other encoding schemes. The OSI presentation-layer protocol permits the announcement or negotiation of the encoding scheme that is being used (or will be used) for data transfer between different systems.

As use of the BER is mandatory in many OSI standards, backward compatibility requires that the BER always be available as the default transfer syntax in every encoding-scheme negotiation. Thus, current implementations of BER-based encoders/decoders will continue to be required, but new encoders/decoders can be built to support new, more efficient encoding schemes.

In itself, the study of efficient encoding schemes is not new. There are many proprietary (or application-specific) encoding schemes that are geared toward meeting specific efficiency requirements. This is particularly true for such ISDN signaling protocols as the Q.931 protocol of the Digital Subscriber Signaling One protocol suite or the ISDN User Part protocol in Signaling System 7, both of which have complex bit patterns described for each message, and where bit sequences correspond to various information elements. When a Q.931 or ISDN User Part message is received, the received bits are compared against a bit mask and the information elements extracted from their positions. This procedure is very efficient, as the received pattern is always predictable.

From the protocol designer's point of view, however, defining a message by laying out its encoding is very constraining. If there is a need to add some new piece of information, empty "slots" (for example, unused bits) must be found or, failing that, the entire pattern of the message must be altered.

This situation reflects the classic dichotomy between efficiency and flexibility. Encodings may be made that are very compact and efficient, but the price paid is that the protocols are difficult to extend. Further, general purpose, application-independent tools cannot be built to automate the encoding and decoding processes.

The new work on alternative encoding standards has resulted in three encoding schemes: distinguished encoding rules (DER), canonical encoding rules (CER)[4], and packed encoding rules (PER)[5]. Each of these schemes solves a specific problem associated with use of the BER: the DER and CER eliminate the sender's options

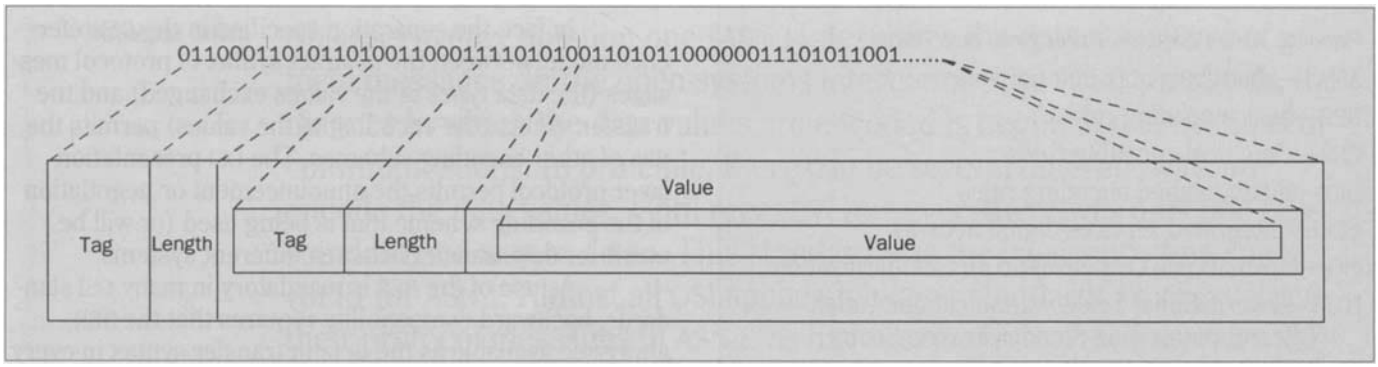


Figure 1. The Tag, Length, and Value form of a BER-based encoding.

available in the BER, while the PER, in addition to removing sender options, reduces message size by packing information into the least possible number of bytes. Future efforts may include standardization of the so-called lightweight encoding rules (LWER)[6], whose intent is to reduce the processing time necessary to encode and decode protocol messages.

The paper's purpose is to introduce these alternative encoding schemes. No attempt is made to be all inclusive, but simply to provide a "flavor" for the issues and the manner in which they are approached. A brief introduction to the BER is first presented, followed by discussions of the DER and CER. Later sections describe the PER and LWER. Each section depicts the circumstances under which the corresponding encoding rule is used, as well as its advantages and disadvantages.

Introduction to the Basic Encoding Rules

The BER were defined so that a receiving entity could decode an incoming bit stream into its component values. In technical jargon, its encoding scheme is described as being self-identifying and self-delimiting, which means that each data value is encoded to permit identification and extraction. If the data value cannot be identified, it can be bypassed. To this end, the BER specify a basic pattern to follow when encoding a data value. Every encoding consists of a unique identifier (frequently called *tag*) for the type of the value being encoded, a length field that tells the decoder the length of the value encoding (in bytes), and then the value itself. The value can be atomic, meaning that the decoder probes no further. Or, it can be composite, meaning that

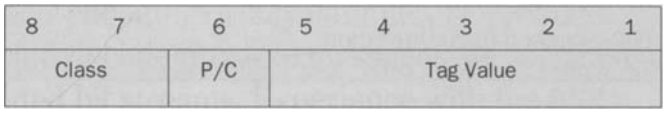


Figure 2. Encoding structure of a tag.

it is made up of other data values, each of which can be further extracted by its tag, length, and value. The tag contains an indication of whether the value is atomic (also known as *primitive*, in standard terminology) or composite (also known as *constructor*). This situation is illustrated in the following diagram, which shows a constructor value containing a primitive value. The three parts that make up an encoding—tag, length, and value—are illustrated in Figure 1 and described in the following subsections.

Tags. One feature of the ASN.1 notation is the concept of tagging. Each data structure is assigned a tag, either by the ASN.1 standard[1], or by the designer. The structure's value can then be uniquely identified when decoding a bit stream representing the structure's encoding. Thus, ASN.1, in addition to expressing the type of data structures that comprise protocol messages, also contains the key that provides a machine-independent mechanism for identifying the type.

The basic encoding structure of a tag, which is described in the BER standard[2,3], is defined in Figure 2. P/C stands for primitive/constructor and identifies, respectively, whether the value being encoded is atomic or composite (that is, made up of other tag, length, and value encodings). The tag takes values from 0 to 30. However, it is possible to extend tag values beyond this range.[2,3] The class field—bits 8 and 7 of the tag—specifies the domain within which the tag is unique. Tags for which these bits are coded 00 are termed UNIVERSAL,

```
Directory-messages ::=          CHOICE {
                                getNumber      [APPLICATION 0] IMPLICIT GetNumber,
                                number          [APPLICATION 1] IMPLICIT Number }

GetNumber ::=                   SEQUENCE {
                                  firstName      VisibleString,
                                  surname       VisibleString  }

Number ::=                      SEQUENCE SIZE (10) OF INTEGER (0..9)
```

Figure 3. The abstract syntax for the Directory look-up protocol example.

60_{16}	← APPLICATION 0 tag identifying GetNumber and replacing tag for SEQUENCE
$0C_{16}$	← Total length of the following value
$1A_{16}$	← UNIVERSAL tag for VisibleString and encoded as primitive
03_{16}	← Length of firstName
$6A_{16}$	← "j"
$6F_{16}$	← "o"
65_{16}	← "e"
$1A_{16}$	← UNIVERSAL tag for VisibleString and encoded as primitive
05_{16}	← Length of surname
$6A_{16}$	← "j"
$6F_{16}$	← "o"
$6E_{16}$	← "n"
65_{16}	← "e"
73_{16}	← "s"

Figure 4. A possible BER-based encoding for the `GetNumber` message

with different tag values identifying the data types INTEGER, REAL, OCTET STRING, and so on. These tags are defined in the BER standard so that, being unique, they should therefore be understood by all communicating systems. Other tags, unique within varying domains, are specified by the protocol designer. APPLICATION tags, where bits 8 and 7 are coded 01, are usually assigned by protocol designers in specific application-layer standards. These assigned bits can distinguish different outer-level data structures, such as protocol messages. PRIVATE tags, identified by the class field coded as 11, allow identification of proprietary extensions made to standards. CONTEXT-SPECIFIC tags (class field = 10) typically distinguish the different components within a set or a sequence of data values.

Lengths. The length field allows the decoder to find the end of an encoded value. BER can encode the length field in one of three ways: the *definite* form, comprised of the *short* form and the *long* form; and the *indefinite* form. The short form of indicating length consists of a single octet, with bit 8 set to 0. Bits 7 to 1 encode the number of bytes needed to represent the

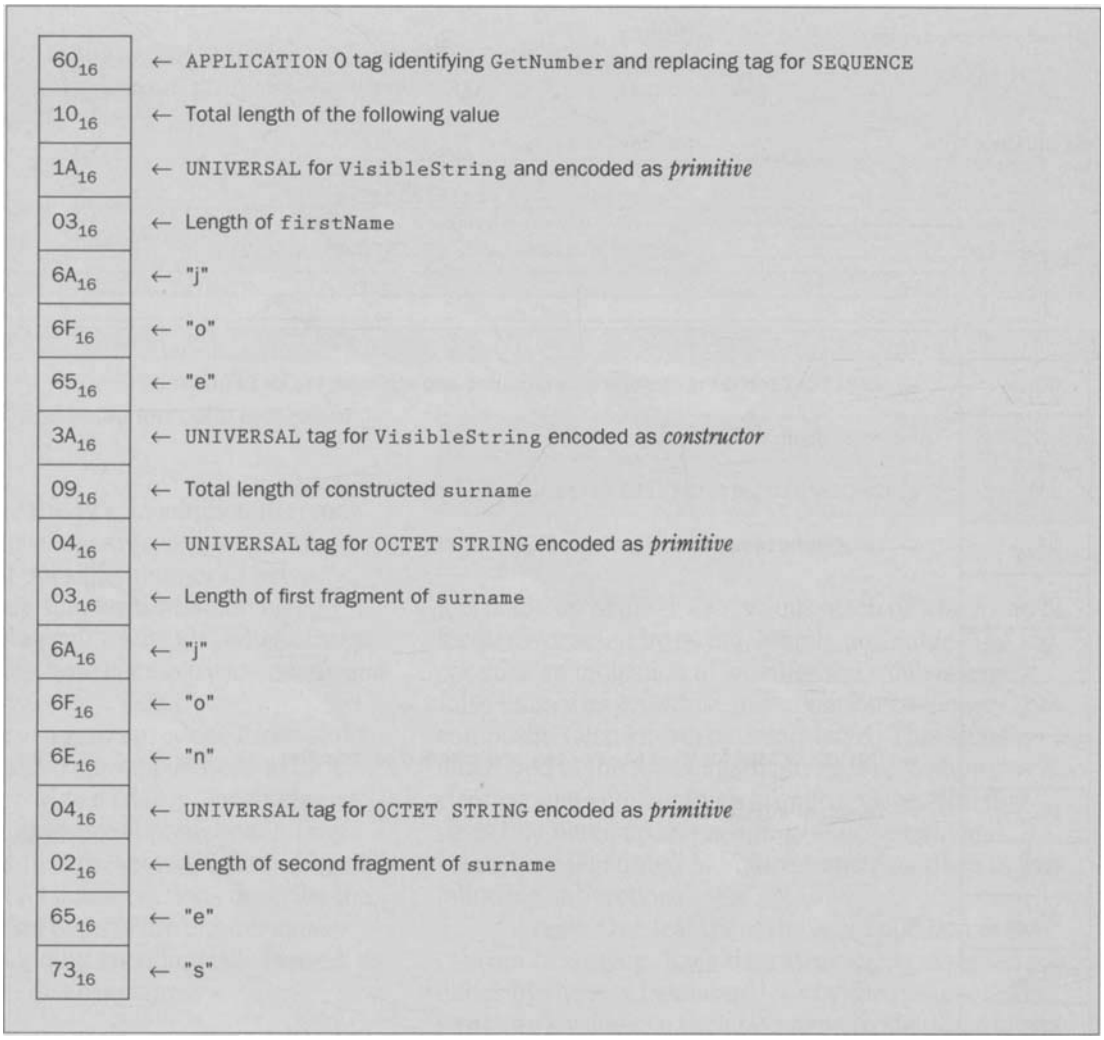


Figure 5. Another possible BER encoding for the same value of the `GetNumber` message.

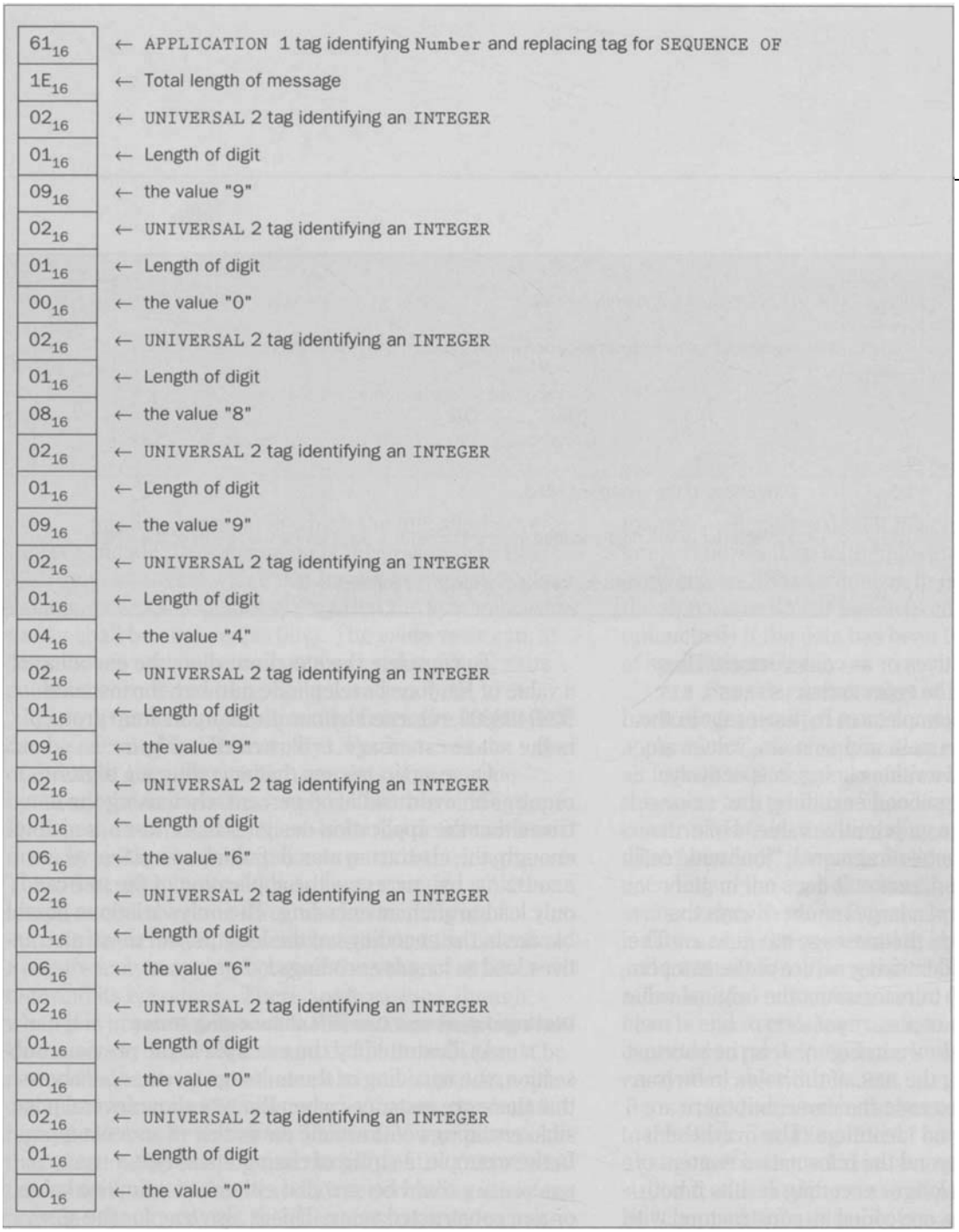
value. Clearly, values that only take up to 127 bytes can be encoded using the short form length. The long form of encoding lengths must be used for values that are longer. There is a specific method for coding the long form length.[2,3]

The indefinite form for encoding the length consists of a special pattern for indicating the "end of contents." Just in case this special pattern is present in an encoding of a value, the indefinite form cannot be used for encoding values that are primitive. It is the sender's option whether to use the definite or indefinite form when encoding the length of a constructed value, or whether to use the long or short form for a value of 127 bytes or fewer.

As discussed later, the newer encoding rules restrict the sender's flexibility in choosing how to encode the length fields.

An Example of a BER Encoding. As an example of how a value of an ASN.1 type is encoded using the BER, consider a simple request-reply protocol for a telephone-directory look-up. One message, `GetNumber`, provides the name of the person whose telephone number is being requested. The reply, `Number`, provides the telephone number and is written as illustrated in Figure 3.

The protocol designer differentiates between the two messages `GetNumber` and `Number` by assigning them different, application-specific tags (`[APPLICATION 0]` and `[APPLICATION 1]`, respectively). Also, for the



61_{16}	← APPLICATION 1 tag identifying Number and replacing tag for SEQUENCE OF
$1E_{16}$	← Total length of message
02_{16}	← UNIVERSAL 2 tag identifying an INTEGER
01_{16}	← Length of digit
09_{16}	← the value "9"
02_{16}	← UNIVERSAL 2 tag identifying an INTEGER
01_{16}	← Length of digit
00_{16}	← the value "0"
02_{16}	← UNIVERSAL 2 tag identifying an INTEGER
01_{16}	← Length of digit
08_{16}	← the value "8"
02_{16}	← UNIVERSAL 2 tag identifying an INTEGER
01_{16}	← Length of digit
09_{16}	← the value "9"
02_{16}	← UNIVERSAL 2 tag identifying an INTEGER
01_{16}	← Length of digit
04_{16}	← the value "4"
02_{16}	← UNIVERSAL 2 tag identifying an INTEGER
01_{16}	← Length of digit
09_{16}	← the value "9"
02_{16}	← UNIVERSAL 2 tag identifying an INTEGER
01_{16}	← Length of digit
06_{16}	← the value "6"
02_{16}	← UNIVERSAL 2 tag identifying an INTEGER
01_{16}	← Length of digit
06_{16}	← the value "6"
02_{16}	← UNIVERSAL 2 tag identifying an INTEGER
01_{16}	← Length of digit
00_{16}	← the value "0"
02_{16}	← UNIVERSAL 2 tag identifying an INTEGER
01_{16}	← Length of digit
00_{16}	← the value "0"

Figure 6. A BER encoding for the `Number` message.

sake of illustrating some points with respect to the various encoding rules, it is necessary to add some size constraints. The expression `SEQUENCE SIZE (10) OF INTEGER (0..9)` states that there are 10 integers in sequence, with each integer value being between 0 and 9, as is true for a North American telephone number.

The message `GetNumber` has two components in sequence, both of the ASN.1 type `VisibleString`. The first component carries the value of the first name of the person, and the second component carries the surname.

If this `GetNumber` message is used to request a certain Joe Jones's telephone number, and the data value exchanged is `{firstName "joe", surname "jones"}`, then the BER rules will automatically generate one of the encodings, shown in Figure 4, for this value (the values of tags, lengths, and integers are shown as two hexadecimal digits per byte, as is the value of the contents, one character per octet, which is coded according to ISO 646[7]). Another possible BER-based encoding for the same value is illustrated in Figure 5.

Different encodings of the same value are possible because the BER allow certain data types to be

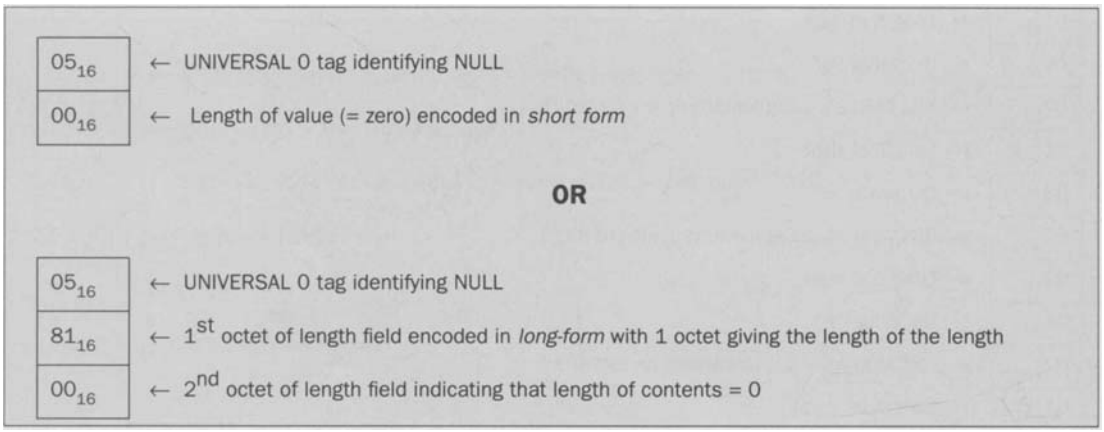


Figure 7. Two ways of encoding the value of the ASN.1 type `NULL` using BER.

encoded either as primitives or as constructors. These include, among others, the types `OCTET STRING`, `BIT STRING` and, as in the example, `VisibleString`. In the first encoding, the `firstName` and `surname` values are both encoded as primitive values being, respectively, "joe" and "jones." In the second encoding, the `firstName` is shown encoded as a primitive value, while "jones" is broken up into two fragments, "jon" and "es," each coded as an `OCTET STRING`. It does not matter which of these two (out of a large number) ways the sender chooses to encode the message `GetNumber`. The self-delimiting and self-identifying nature of the BER permits the receiving entity to reconstruct the original value `{firstName "joe", surname "jones"}`.

The first case, shown in Figure 4, is the shortest possible encoding, using the BER, of the value to be transmitted. It takes 8 bytes to code the name, but there are 6 extra bytes for lengths and identifiers. The overhead is, therefore, 42 percent beyond the information content of the value itself. An even longer encoding results if both `firstName` and `surname` are coded as constructors, with *each* letter of the names coded separately as an octet string. (This is the extreme case of the form shown in the second encoding example.) The total message length for this extreme case is 30 bytes, of which 22 bytes are overhead for identifiers and lengths. This represents an overhead of 73 percent beyond the information content of the value itself. As discussed in later sections, the overhead is necessary if flexibility at the level of specifications is desired. The PER-based encoding rules reduce this overhead, but they constrain the ability to add new components to existing messages at will, without making provisions for doing so beforehand.

To complete the BER discussion, the encoding of a value of Joe Jones's telephone number, for instance 9089496600, returned by our directory look-up protocol in the `Number` message, is illustrated in Figure 6.

Once again, we see that encoding ten digits requires an overhead of 66 percent when using the BER. Given that the application designer has chosen, sensibly enough, the abstract-syntax definition of `Number` as a `SEQUENCE OF INTEGER`, the application of the BER can only lead to such an encoding. The only variations possible are in the encodings of the lengths, but these alternatives lead to longer encodings.

Distinguished and Canonical Encoding Rules

As illustrated by the example in the previous subsection, the encoding of the message `GetNumber` shows that there are instances when the BER allow several possible encodings of the same data value of an ASN.1 type. In the example, a string of characters of type `VisibleString` could be encoded either as a primitive value or as a constructed value. This is also true for the ASN.1 types `BIT STRING` and `OCTET STRING`. Given a series of bits (or bytes), the encoder can create a primitive encoding by putting the bits (or bytes) as the entire contents of the type `BIT STRING` (or `OCTET STRING`). Or, the encoder can chop up the bit-string value arbitrarily into smaller substrings and encode each piece, so derived, as a primitive `BIT STRING` (or `OCTET STRING`). Then, it envelopes the substrings in a constructed `BIT STRING` (or `OCTET STRING`) encoding. It follows that in the cases in which these types are encoded as constructors, the number of bytes required to encode the same value is considerably greater.

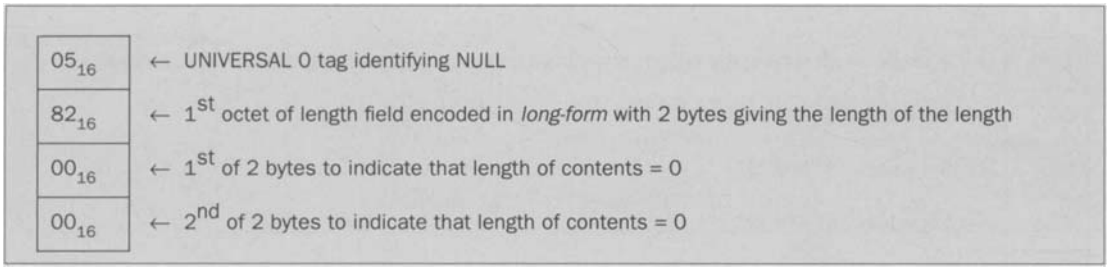


Figure 8. A third way of encoding a value of the ASN.1 type NULL using BER.

Another example in which the BER allow several ways to encode the same value is the encoding of the ASN.1 type BOOLEAN, which can have two symbolic values, TRUE and FALSE. The BER specify that the symbolic value FALSE shall be encoded as 00_{16}. The value TRUE can, at the sender's option, be any non-zero value. Thus, TRUE could be encoded as 01_{16}, FF_{16}, ...and so forth.

A third example shows that even the length field can be encoded in different ways. For instance, the value of the ASN.1 type NULL, which consists of zero value bytes, can be written two ways, as illustrated in Figure 7. Even in the second case, there is no restriction on how many bytes the sender can use to encode the length. Thus, the null value could have been encoded with the length encoded in long-form, as shown in Figure 8.

These examples demonstrate that the BER allow a one-to-many mapping between the value of an ASN.1 type and its encodings. There are occasions, though, when it is important to ensure that an encoding chosen by the sender is preserved, and that if the value must be re-encoded at an intermediate point, such as at an application relay in a store-and-forward situation, the encoding remain exactly as before. This is a new requirement that exceeds the existing functionality of the OSI presentation layer, which chooses whatever encoding it prefers, provided that it conveys the abstract syntax and semantics of the application-layer data values unchanged. An example[8] in which the encodings must be preserved is in the exchange of authentication information, relayed over a series of application associations, to ensure that the accompanying data are not manipulated or altered.

One mechanism for providing a data-integrity service is to take an encoding of the data to be transferred, compress it, encrypt it, and transmit this "digital signature" together with the original data. The receiving entity performs the same steps on the received data, and compares the newly generated signature with the received signature. If they match, the data have not been manipulated; if they do not match, the data have been tampered with. Use of the BER to encode the data, however, causes this mechanism to give false results (that is, the signatures do not match even if the data is untouched) if the data has been transmitted over a series of application relays.

Consider a data value of type OCTET STRING that has to be transmitted from an application A, to a peer application C, via an application relay at B. When in transit from A to B, the presentation layer at A might code this value as a primitive encoding. A digital signature is constructed from this encoding and accompanies the data. At node B, the presentation layer converts the encoding of the user data into the appropriate abstract-syntax value. Because B is an application relay, the value is returned to the presentation layer at B for encoding and transmission. The received digital signature is retransmitted untouched, as the data-integrity mechanism is end to end. Intermediate relay nodes are not expected to participate in, or have "knowledge" of, the encryption algorithm in use.

As discussed in the section titled "Introduction to the Basic Encoding Rules," a value of type OCTET STRING can be encoded in two ways, primitive or constructor, with the latter case allowing the octet string to be "chopped up" arbitrarily into substrings. If one of the latter alternatives is chosen, the encoding of the same value on transmission from B to C is different from that between A and B. If C computes the digital signature, based on the new encoding of the same value, the digital signatures will not match, and C is mistakenly "led to believe" that the data have been tampered with.

The DER and CER have been designed to prevent such mistaken conclusions by removing all of the sender's options allowed by the BER. In essence, they define completely "predictable" BER. DER and CER are defined as a subset of the BER by placing a series of constraints on the BER encoding of various ASN.1 types. In

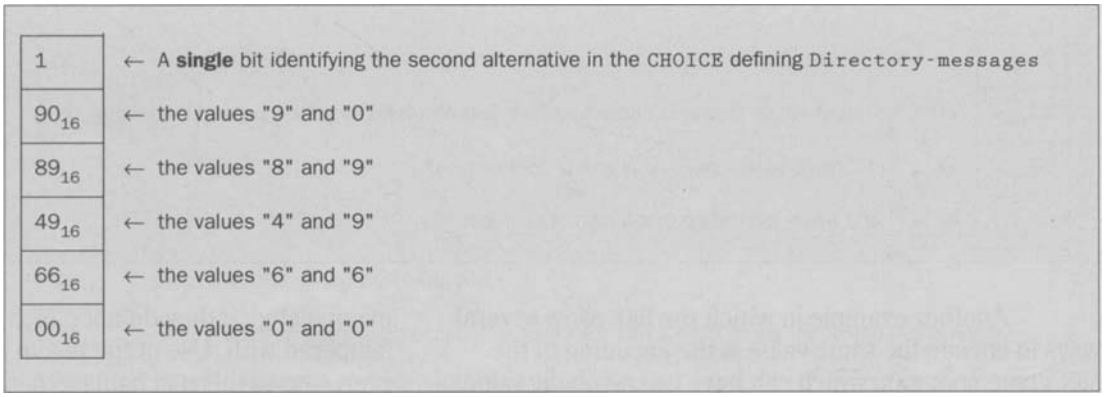


Figure 9. A PER-based encoding of a value of the `Number` message.

what follows, a few of these constraints are included to provide a "feel" for the DER and CER. The primary difference between the DER and CER is that the former uses the definite form (short or long form) for encoding lengths, while the latter uses the indefinite form for constructed values. CER are more suitable than the DER if there is a need to encode a value so large that it does not fit into the available memory, or if there is a need to encode and transmit a part of a value before the entire value is available.

Some examples of constraints on the BER to produce the DER or CER are:

- If the encoding is that of the `BOOLEAN` value `TRUE`, it shall be coded as one octet, with all bits set to one.
- For the CER only, values of type `BIT STRING` or `OCTET STRING` shall be encoded as primitive encodings if they require no more than 1,000 bytes to form the contents, and as a constructed encoding otherwise. The string fragments in the constructed encoding shall be encoded as primitive encodings. The encoding of each fragment, except possibly the last, shall be exactly 1,000 bytes.
- When encoding a `SET` of ASN.1 types, the component's order shall be determined by their tags as follows:
  - — Those with tags of class `UNIVERSAL` shall appear first, followed by those with tags of class `APPLICATION`, then those with `CONTEXT SPECIFIC`, and finally those with `PRIVATE` tags.
  - — Within each class, the components shall appear in the ascending order of their tag values.
- The value of `NULL` shall be encoded as $\{05_{16},00_{16}\}$.

The DER or CER are not directly "concerned" with efficiency. By suppressing the sender's options, a simpler decoder design is permitted. Of course, in reality, each implementation of a BER encoder is simplified as one chooses from the many options available in the BER. The DER or CER, however, enforce one method for all implementations. Naturally, to ensure interoperability, any *de*coder that supports the BER must be able to decode all options available in the BER. So, there is no savings on the decoding front. A major inefficiency of these encoding rules arises for encoding the `SET OF` and `SET` types, owing to the need to perform an ordering of the components.

It is interesting to note that some users of the BER (for instance, the designers of ISDN signaling standards, such as transaction capabilities) chose to place very similar constraints on the BER, which are now standardized by the DER and CER.

Packed Encoding Rules

As discussed in the section titled "Introduction to the Basic Encoding Rules," the self-identifying and self-delimiting nature of the BER allows great flexibility in encoding and decoding the values of arbitrary ASN.1 types. The subsection titled "An Example of a BER Encoding," however, showed that the size of such encodings can become quite large, often needlessly so, as in the case in which a repetitive sequence of values of a known ASN.1 type is transmitted.

The motivation behind the PER is, simply, to "pack where possible." In many cases, where the sending and receiving entities are dealing with a perfectly predictable message structure, the greatest overhead of the BER comes from including the tags and length fields of the data values. Going back to the example of the message `Number` from a previous subsection ("An Example of a BER Encoding"), when the user expects a value of

```
ExtendedGetNumber ::=       SEQUENCE {
                            firstName       VisibleString,
                            middleName      [0] IMPLICIT VisibleString OPTIONAL,
                            surname         VisibleString  }

Number ::=                  SEQUENCE SIZE (10) OF INTEGER (0..9)
```

Figure 10. An extension of the Directory look-up protocol.

type SEQUENCE SIZE (10) OF INTEGER (0..9) giving a ten-digit telephone number, it is unnecessary to precede each digit with two bytes stating that what follows is a digit of type INTEGER, whose value is expressed as a binary value in one octet. (Even though there is a size constraint limiting the integer values to the digits 0 through 9, each of which could be coded most economically in 4 bits, the BER ignore this size constraint, as they do all other constraints.) Similarly, one can drop the total length of the contents, because the type being exchanged is known, exactly, beforehand. The decoder can deduce, therefore, precisely where the end of the contents will be; namely, after the tenth integer number. The PER-based encoding of this value drops these redundant bytes, and the resulting encoding is illustrated in Figure 9. The length of the message Number has dropped from 32 bytes to five bytes and one bit.

The PER come in two versions. The most compact version of a PER-based encoding, called the "octet unaligned" version, is one into which no padding bits are inserted to restore octet alignment between certain values. In the (octet) "aligned" version of the PER, the resulting encoding can be somewhat longer, as padding bits may be inserted at certain points. (The drawback of the unaligned variant, despite the gain in further reducing bandwidth, is that the encoder/decoder has to perform, in general, many CPU-intensive manipulations to encode/extract the values.) Note, however, that the decoder and the reader are able to "understand" the compressed encoding and extract the values, only because they "know" exactly the abstract syntax, or the expected type definition. The PER, unlike the BER, do not allow the decoder to skip over an unknown field and extract the value of some later field. This is because it is not possible—owing to the absence of tags and, often, lengths—to determine the end of the encoding of some value without knowing what *type* of value it is.

As before, only an overview is provided of the rules that, when applied, constitute the PER-based encoding of an ASN.1 value. Additional details of the procedure are available.[5] Examples of the PER are:

- *Rules for when tag fields should be removed.* No tags are encoded. For cases such as the ASN.1 type CHOICE, in which distinct tags are required to identify the alternatives, a choice-index field, preceding the encoding, identifies the chosen alternative. (To ensure that both parties agree on which alternative is chosen, the choice alternatives must be ordered, based on their tags. Alternatives with UNIVERSAL tags are placed first, followed by those of class APPLICATION, and so forth, and, within each class, in ascending order of tag values. This ordering would not have been necessary with the BER, as the tag identifying the chosen alternative is always explicitly present.) After the ordering, an index is assigned to each alternative. It is this *choice index* that is encoded, as a single bit, to indicate the chosen alternative.
- *Rules for when length fields should be removed.* The size and length constraints of the ASN.1 notation are used to decide on the encoding, and the length fields are not included when length can be determined from the notation. For example, if there is a finite (known) sequence of BOOLEAN values being encoded, it is enough to use one bit to represent a BOOLEAN value. If this assumption is understood, it is not necessary either to precede each value with its length count or to provide the total length.
- *How to represent* INTEGER, BOOLEAN, *and so forth.* If an integer value is constrained [say, by being of type INTEGER (0..9)] as in the example, then it can be depicted as a bit string representing the offset of this value from the lower bound. In the example, any integer value in this range is represented by four bits. If the value is unconstrained, then it has to be preceded by a length field. A BOOLEAN value is exactly one bit, with 1 representing TRUE.
- *How to encode* SET *or* SEQUENCE. The encoding of a SEQUENCE is preceded by a bit map, which describes if

Figure 11. A PER-based encoding of a value of the `GetNumber` message.

0	← A **single bit** identifying the first alternative in the `CHOICE` defining `Directory-messages`
03_{16}	← Length of `firstName`
0110 1010	← the binary representation of "j" with the **leading bit deleted**
0110 1111	← the binary representation of "o" with the **leading bit deleted**
0110 0101	← the binary representation of "e" with the **leading bit deleted**
05_{16}	← Length of `surname`
0110 1010	← the binary representation of "j" with the **leading bit deleted**
0110 1111	← the binary representation of "o" with the **leading bit deleted**
0110 1110	← the binary representation of "n" with the **leading bit deleted**
0110 0101	← the binary representation of "e" with the **leading bit deleted**
0111 0011	← the binary representation of "s" with the **leading bit deleted**

any component marked `OPTIONAL` or `DEFAULT` is included in the encoding. For a `SET`, the components shall be ordered in canonical order based on their tag values (see the first dashed item on tag ordering rules), and then treated exactly as if it were a `SEQUENCE`.

Extensibility and the PER. As always, the benefit of having shorter encodings for ASN.1 types, for use in restricted bandwidth environments, comes at a price. In this case, the price is the inability to make extensions, *arbitrarily*, to an abstract syntax. A description of this handicap follows.

Often, minor changes must be made to a protocol after it has been defined, without creating either a new abstract-syntax name, or a new version number. As long as extensions are not used when communicating with a node that supports the unextended version, interworking is possible, provided that the encoding rules for the abstract syntax support the extensibility of the syntax. The following example illustrates this point.

Going back to the discussion of the protocol `Directory-messages` for the telephone-number lookup, the protocol is extended by allowing the user to add the middle name—a common enough requirement, as there are usually many J. Joneses in the telephone directory. The extended version of the protocol (with the addition in bold), is shown in Figure 10. Note that the value `middleName` must be made optional, as it should not be sent when communicating with an earlier version of the directory not expecting to receive it. Also, the designer has inserted this new, optional value in the middle, which would seem like the logical place to add this field.

As long as the `middleName` value is not encoded, the BER encoding is obviously identical to the one shown in the subsection titled "An Example of a BER Encoding." The receiver is not aware that the syntax has been changed at the sender's end. This is an example of the great flexibility provided by the BER. New components can always be added anywhere within a `SET` or `SEQUENCE`. Furthermore, as long as values of the new component are not generated, the receiver does not "know" that the abstract-syntax definition has changed.

Application of the PER, however, leads to a different situation. First, consider the application of the unaligned version of the PER to the abstract syntax of the message `GetNumber`, which was defined in the previously mentioned subsection. It leads to the encoding for the value `{firstName "joe", surname "jones"}` shown in Figure 11.

The packing philosophy of the PER can be used to understand the preceding encoding. A single bit, known as the *choice index* (rather than a one-byte tag), distinguishes this message from the other message that, together, make up the message set for this directory example. If the receiver expects values for two `VisibleString` items in sequence, the redundant tags for these values can be discarded, as well as the total message length. As the number of characters in each string is a variable, however, string lengths have to be explicitly

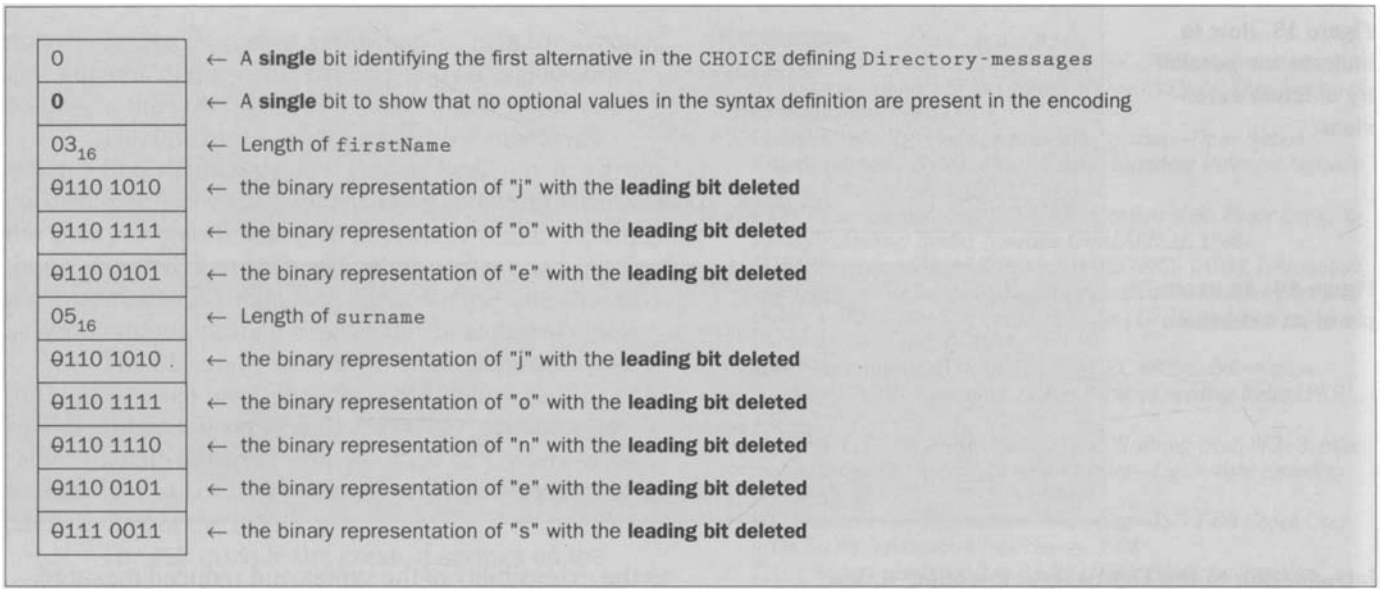


Figure 12. PER-based encoding of the identical value using the `ExtendedGetNumber` message.

included. Furthermore, in the unaligned version of this PER encoding, the leading bit in the encoding of each character value is deleted, as seven bits are sufficient to encode all characters of the type `VisibleString`.

It can be verified, by concatenating the bit fields shown, that the total encoding is nine bytes and one bit. This may be compared with the 14 bytes comprising the smallest BER encoding of the same value, as illustrated in the subsection titled "An Example of a BER Encoding." Note also that it is not possible—when using the PER—to sight-read what is being encoded, from analyzing the generated bit pattern, without knowing the complete abstract-syntax definition.

If, however, the abstract syntax of `GetNumber` is extended, as shown in this section, by the inclusion of the middle name—but the value for the optional component `middleName` is not transmitted—the encoding of the same value `{firstName "joe", surname "jones"}`, using the (unaligned variant of the) PER, is shown in Figure 12.

Note that application of the PER requires the presence of an extra bit (the second bit in bold) coded zero, which is the indication that there is one optional component in the `SEQUENCE`, but the value is not encoded. Thus, even though the optional component's value is not transmitted, the encoding has changed because the abstract-syntax definition has changed. The receiver (which only understands the earlier syntax) is unable to decode this bit stream. This is due to the lack of flexibility in using the PER, as opposed to the BER. A naive application of the PER does not preserve the transfer syntax of existing values when a change is made to an abstract syntax, even when the new additions are not included in a transmitted value.

A new amendment to the ASN.1 notation[9] can be implemented, which allows use of the PER and other encoding rules when abstract syntaxes must be extended. The amendment permits a protocol designer to indicate that a particular abstract syntax may be extended, in the future, by adding optional components *at the end* of a `SET`, `SEQUENCE`, `CHOICE`, or `ENUMERATED` type. This is done by placing ellipses "...," called the *extensibility marker*, as shown in Figure 13. So, when an extension has to be made, the abstract syntax changes as illustrated in Figure 14.

The PER have been designed to be extensible only where the abstract-syntax definition includes this explicit indication of extensibility. When such an extensibility marker is present, the PER encoding of the `SEQUENCE` is preceded by a single bit, which tells whether the values of any extensions are present. All application designers are encouraged to use the extensibility marker to ensure that the PER can be used.

Figure 13. How to indicate the possibility of future extensions.

```
GetNumber ::=       SEQUENCE {
                      firstName      VisibleString,
                      surname        VisibleString,
                       ...   }

Number ::=          SEQUENCE SIZE (10) OF INTEGER (0..9)
```

Figure 14. An example of an extension.

```
GetNumber ::=       SEQUENCE {
                      firstName      VisibleString,
                      surname        VisibleString,
                       ... ,
                      middleName     [0] IMPLICIT VisibleString OPTIONAL  }

Number ::=          SEQUENCE SIZE (10) OF INTEGER (0..9)
```

Introduction to the Lightweight Encoding Rules

With implementations of application-layer protocols using ASN.1 and BER, it is common that the encoding and decoding portion of the processing consumes the most resources. It is the bottleneck in meeting efficiency requirements in high-speed, high-bandwidth data communications. A few of the reasons for this are:

- The efficiency of processors, whose natural processing size is larger than a byte (those with 16-bit, 32-bit, or 64-bit word sizes), is reduced when dealing with the BER. For instance, the BER require that integer values be coded in the least number of bytes. Encoding or decoding an integer value, using the BER, are slowed because simple memory-copying techniques cannot be used. Rather, a series of steps has to be taken to compute integer-value length.
- Many BER-encoded values, such as tags or `OBJECT IDENTIFIER` values, are not a fixed number of bytes. The work expended to isolate these values has been found by some to be computationally expensive. Also, the byte ordering in BER is predetermined and may not match the architecture of the encoder/decoder.
- The decoding of an ASN.1 `SET` is as time consuming as the order of the received elements is unpredictable. So, additional checks must be made to verify that all the expected components have been received.

The LWER comprise a set of rules in the very early stages of design. They are meant for use when it is important to increase the efficiency with which data are encoded and decoded, and in which other aspects, such as the extensibility of the syntax and reduced message size, are not crucial. To accomplish this, the variants of LWER, each of which is based on a particular machine architecture (word size and storage representation), attempt to maximize the similarity of their machine architectures to decrease encoding and decoding times. As usual, this comes at a price. The encodings are sometimes considerably larger than those generated by the BER, because entire words are transmitted. Therefore, the LWER will be most effective in high-speed, high-bandwidth environments, where end-system processing delays are the limiting factor.

It must be mentioned that work on the LWER is in its infancy in the standards process, and there is an understandable reluctance to standardize too many encoding rules, particularly those that appear to be extremely machine dependent. Individuals not sympathetic to this activity feel that increasing the number of encoding schemes will prevent interoperability, apart from the greater development cost of implementing and supporting several types of encoders and decoders. Supporters feel that the current bandwidth restrictions will disappear with high-speed communications, but as end-system processing powers may not increase quite so quickly, the bottleneck there must be removed.

Conclusions

If the three aspects of good design—flexibility, extensibility, and compactness—are considered when implementing protocol specifications, each of the

encoding rules discussed in this paper offers the greatest gains on one of these dimensions, and neutral to poor results on the others.

The BER were designed to be *the* encoding scheme that would provide complete flexibility to a protocol designer. They allow extensibility, and offer decoders the ability to ignore unknown information. The encodings thus generated, however, are rather verbose and much of the information is redundant, particularly in the case of systems communicating repeatedly about one matter.

The DER and CER remove the options available to the sender when using the BER. This, in turn, makes possible their application in situations where *encodings*—rather than the abstract values—must be preserved as, for example, in security exchanges. Their efficiency is on par with that of the BER.

The PER provide the greatest savings on the compactness, or bandwidth, dimension. This dramatic improvement is more than compensated for by the need to ensure, from the beginning, that the abstract-syntax definition of a protocol can be extended. For perfectly predictable message structures exchanged, and where there are restrictions on the bandwidth of the transmission "pipe," the PER prove truly beneficial.

The LWER overcome the efficiency concerns of actual end-system processing, and are suitable if there is communication between machines of like internal architecture in high-speed, high-bandwidth environments. It is unclear if variants of the LWER can accommodate all possible architectures, so it may be that their efficiency will be beneficial only to islands of users.

Acknowledgments

The author wishes to thank Bancroft Scott of Open Systems Solutions, Inc., Princeton, New Jersey, who is also the editor of the standards on ASN.1 and its encoding rules, for many helpful discussions on these matters and a careful reading of this manuscript.

References

1. ITU-T Recommendation X.680 | ISO/IEC 8824-1, *Information technology—Abstract Syntax Notation One (ASN.1): Specification of basic notation*, 1994.
2. ISO/IEC 8825, *Information processing systems—Open Systems Interconnection—Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1)*, 1988.
3. CCITT Recommendation X.209, *Specification of the Basic Encoding Rules for Abstract Syntax Notation One (ASN.1)*, 1988.
4. ITU-T Recommendation X.690 | ISO/IEC 8825-1:1994, *Information technology—ASN.1 encoding rules—Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER), and Distinguished Encoding Rules (DER)*, 1994.
5. ITU-T Recommendation X.691 | ISO/IEC 8825-2, *Information technology—ASN.1 encoding rules—Packed Encoding Rules (PER)*, 1995.
6. ISO/IEC JTC1/SC21 N6131, ISO/IEC Working Draft 8824-3, *Information technology—ASN.1 encoding rules—Lightweight Encoding Rules (LWER)*.
7. ISO Standard 646, *Information Processing—ISO 7-Bit Coded Character Set for Information Interchange*, 1993.
8. ITU-T Recommendation X.509 | ISO/IEC 9594-8, *Information technology—Open Systems Interconnection—The Directory: Authentication Framework*, 1993.
9. Draft Amendment to ITU-T Recommendation X.680 | ISO/IEC 8824-1, *Information technology—Abstract Syntax Notation One (ASN.1)—Rules of Extensibility*, 1994.

(Manuscript approved April 1994)

Nilotpal Mitra *is a member of the technical staff in the Standards Planning Department at AT&T Bell Laboratories in Holmdel, New Jersey. His work involves developing architectures and standards on ISDN signaling for broadband and intelligent networks, as well as research on OSI-based data communications. He also represents AT&T in these areas in the ITU-T and ISO/IEC standards bodies. Mr. Mitra received a Ph.D. in theoretical physics from Columbia University in New York City. He joined AT&T in 1983.*