

policies in a Local Configuration Datastore (LCD). Portions of the LCD are themselves accessible as managed objects, defined in the View-Based Access Control Model Configuration MIB [RFC 3415], and thus can be managed and manipulated remotely via SNMP.

9.4 ASN.1

In this book, we have covered a number of interesting topics in computer networking. This section on ASN.1, however, may not make the top-ten list of interesting topics. Like vegetables, knowledge about ASN.1 and the broader issue of presentation services is something that is “good for you.” ASN.1 is an ISO-originated standard that is used in a number of Internet-related protocols, particularly in the area of network management. For example, we saw in Section 9.3 that MIB variables in SNMP were inextricably tied to ASN.1. So while the material on ASN.1 in this section may be rather dry, we hope the reader will take it on faith that the material *is* important.

In order to motivate our discussion here, consider the following thought experiment. Suppose one could reliably copy data from one computer’s memory directly into a remote computer’s memory. If one could do this, would the communication problem be “solved?” The answer to the question depends on one’s definition of “the communication problem.” Certainly, a perfect memory-to-memory copy would exactly communicate the bits and bytes from one machine to another. But does such an exact copy of the bits and bytes mean that when software running on the receiving computer accesses this data, it will see the same values that were stored into the sending computer’s memory? The answer to this question is “not necessarily!” The crux of the problem is that different computer architectures, different operating systems, and different compilers have different conventions for storing and representing data. If data is to be communicated and stored among multiple computers (as it is in every communication network), this problem of data representation must clearly be solved.

As an example of this problem, consider the simple C code fragment below. How might this structure be laid out in memory?

```
struct {
    char code;
    int x;
} test;
test.x = 259;
test.code = 'a';
```

The left side of Figure 9.6 shows a possible layout of this data on one hypothetical architecture: there is a single byte of memory containing the character **a**,

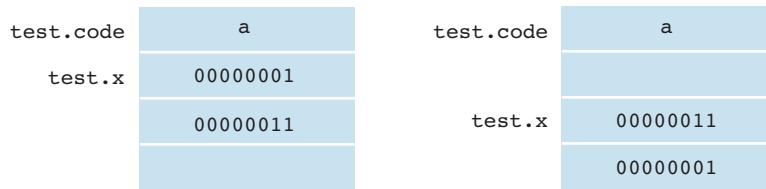


Figure 9.6 ♦ Two different data layouts on two different architectures

followed by a 16-bit word containing the integer value 259, stored with the most significant byte first. The layout in memory on another computer is shown in the right half of Figure 9.6. The character a is followed by the integer value stored with the least significant byte stored first and with the 16-bit integer aligned to start on a 16-bit word boundary. Certainly, if one were to perform a verbatim copy between these two computers’ memories and use the same structure definition to access the stored values, one would see very different results on the two computers!

The fact that different architectures have different internal data formats is a real and pervasive problem. The particular problem of integer storage in different formats is so common that it has a name. “Big-endian” order for storing integers has the most significant bytes of the integer stored first (at the lowest storage address). “Little-endian” order stores the least significant bytes first. Sun SPARC and Motorola processors are big-endian, while Intel processors are little-endian. As an aside, the terms “big-endian” and “little-endian” come from the book, *Gulliver’s Travels*, by Jonathan Swift, in which two groups of people dogmatically insist on doing a simple thing in two different ways (hopefully, the analogy to the computer architecture community is clear). One group in the land of Lilliput insists on breaking their eggs at the larger end (“the big-endians”), while the other insists on breaking them at the smaller end. The difference was the cause of great civil strife and rebellion.

Given that different computers store and represent data in different ways, how should networking protocols deal with this? For example, if an SNMP agent is about to send a Response message containing the integer count of the number of received UDP datagrams, how should it represent the integer value to be sent to the managing entity—in big-endian or little-endian order? One option would be for the agent to send the bytes of the integer in the same order in which they would be stored in the managing entity. Another option would be for the agent to send in its own storage order and have the receiving entity reorder the bytes, as needed. Either option would require the sender or receiver to learn the other’s format for integer representation.

A third option is to have a machine-independent, OS-independent, language-independent method for describing integers and other data types (that is, a data-definition language) and rules that state the manner in which each of the data types is to be transmitted over the network. When data of a given type is received, it is received in a known format and can then be stored in whatever machine-specific format is required. Both the SMI that we studied in Section 9.3 and ASN.1 adopt this third option. In ISO parlance, these two standards describe a **presentation service**—the service of transmitting and translating information from one machine-specific format to another. Figure 9.7 illustrates a real-world presentation problem; neither receiver understands the essential idea being communicated—that the speaker likes something. As shown in Figure 9.8, a presentation service can solve this problem by translating the idea into a commonly understood (by the presentation service), person-independent language, sending that information to the receiver, and then translating into a language understood by the receiver.

Table 9.5 shows a few of the ASN.1-defined data types. Recall that we encountered the INTEGER, OCTET STRING, and OBJECT IDENTIFIER data types in our earlier study of the SMI. Since our goal here is (mercifully) not to provide a complete introduction to ASN.1, we refer the reader to the standards or to the printed and online book [Larmouth 1996] for a description of ASN.1 types and constructors, such as SEQUENCE and SET, that allow for the definition of structured data types.

In addition to providing a data definition language, ASN.1 also provides **Basic Encoding Rules (BER)** that specify how instances of objects that have been defined using the ASN.1 data definition language are to be sent over the network. The BER adopts a so-called **TLV (Type, Length, Value) approach** to encoding data for transmission. For each data item to be sent, the data type, the length of the data item,

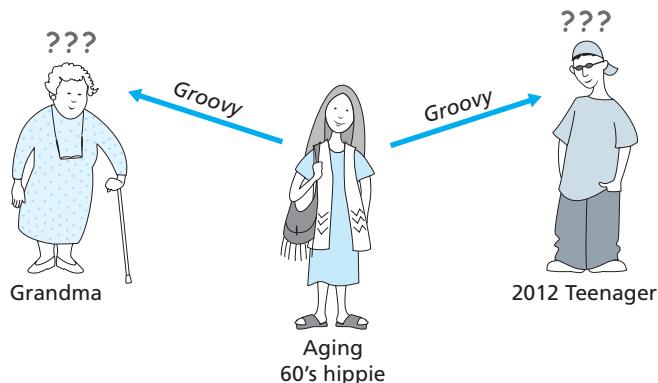


Figure 9.7 ♦ The presentation problem

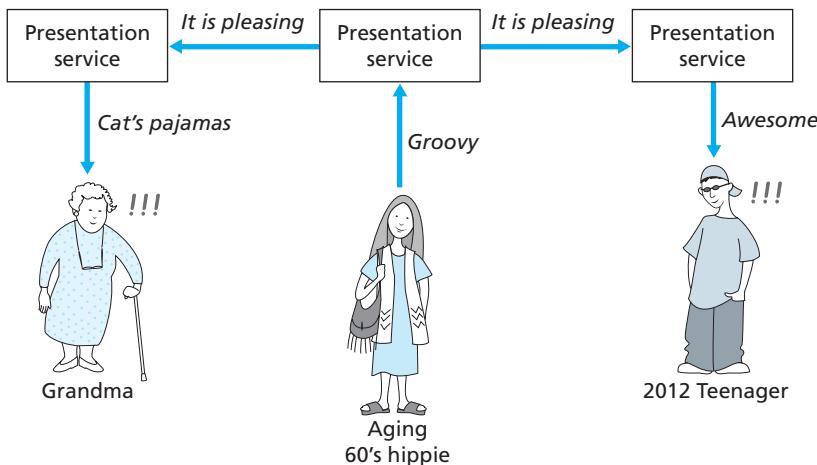


Figure 9.8 ♦ The presentation problem solved

and then the actual value of the data item are sent, in that order. With this simple convention, the received data is essentially self-identifying.

Figure 9.9 shows how the two data items in a simple example would be sent. In this example, the sender wants to send the character string “smith” followed by the value 259 decimal (which equals 00000001 00000011 in binary, or a byte value of 1 followed by a byte value of 3), assuming big-endian order. The first byte in the

Tag	Type	Description
1	BOOLEAN	value is “true” or “false”
2	INTEGER	can be arbitrarily large
3	BITSTRING	list of one or more bits
4	OCTET STRING	list of one or more bytes
5	NULL	no value
6	OBJECT IDENTIFIER	name, in the ASN.1 standard naming tree; see Section 9.2.2
9	REAL	floating point

Table 9.5 ♦ Selected ASN.1 data types

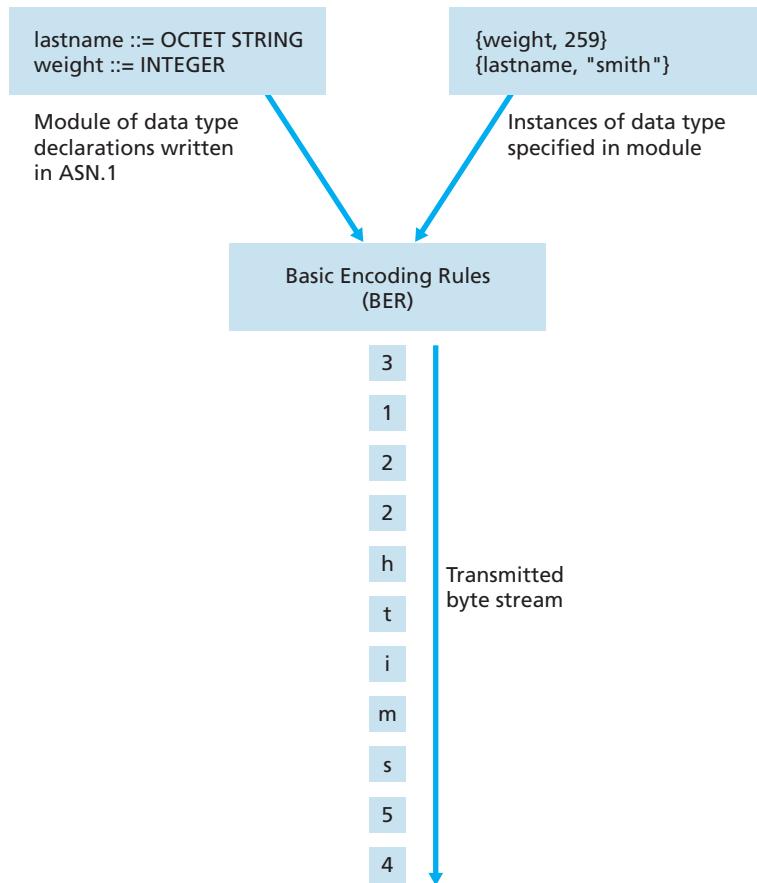


Figure 9.9 ♦ BER encoding example

transmitted stream has the value 4, indicating that the type of the following data item is an OCTET STRING; this is the “T” in the TLV encoding. The second byte in the stream contains the length of the OCTET STRING, in this case 5. The third byte in the transmitted stream begins the OCTET STRING of length 5; it contains the ASCII representation of the letter *s*. The T, L, and V values of the next data item are 2 (the INTEGER type tag value), 2 (that is, an integer of length 2 bytes), and the 2-byte big-endian representation of the value 259 decimal.

In our previous discussion, we have only touched on a small and simple subset of ASN.1. Resources for learning more about ASN.1 include the ASN.1 standards document [ISO X.680 2002], the online OSI-related book [Larmouth 2012], and the ASN.1-related Web sites, [OSS 2012] and [OID Repository 2012].