Codificação/Descodificação de PDUs SNMPv2c

Janeiro 2018
Fábio Gonçalves, Bruno Dias
Departamento de Informática, Universidade do Minho

O presente tutorial foi preparado numa distribuição Ubuntu do sistema operativo Linux para a linguagem de programação C. A ferramenta escolhida para compilar especificações ASN.1 para código C chama-se **asn1c** e pode ser encontrada em **https://github.com/vlm/asn1c**

1. Para instalar a ferramenta escolhida, primeiro fazer *clone* do código do **github**:

    a. Instalar dependências base:
        i. apt update
        ii. apt install build-essential
        iii. apt install autoconf
        iv. apt install libtool
        v. apt install git

    b. Fazer *clone* do projecto do **git**:
        i. git clone https://github.com/vlm/asn1c
        ii. Follow the instructions in INSTALL.md to install the tool:
            1. test -f configure || autoreconf -iv
            2. ./configure
            3. make
            4. make check
            5. make install

2. Copiar as definições ASN.1 das mensagens/PDUs SNMPv2c da norma RFC3416 para um ficheiro (por exemplo, snmpv2c.asn1) :

    a. Copiar SNMPv2-PDU DEFINITIONS
    (RFC3416, https://tools.ietf.org/html/rfc3416)

    b. Copiar COMMUNITY-BASED-SNMPv2 DEFINITIONS
    (RFC1901, https://tools.ietf.org/html/rfc1901)

3. Utilizar a ferramenta **asn1c** instalada para compilar o ficheiro criado anteriormente:

    a. asn1c snmpv2c.asn1

b. A compilação irá gerar todos os ficheiros necessários para efetuar a codificação e descodificação de mensagens SNMP, versão 2c. A compilação do ficheiro cria diversas estruturas de dados que necessitam depois de ser preenchidas por forma a codificar/descodificar cada uma das primitivas SNMP (get-request, set-request, etc.).

c. Uma aplicação exemplo é gerada com o nome de converter-example.c e um *makefile* com o nome converter-example.mk

d. Os ficheiros gerados poderão ser copiados para duas pastas diferentes (encode e decode, por exemplo) de forma a criar duas aplicações distintas, uma para codificar e outra para descodificar mensagens SNMPv2c.

e. Criar o ficheiro encoder.c numa pasta e decoder.c na outra pasta. Estes serão os ficheiros onde estará a *main()* de cada uma das aplicações. Copiar o ficheiro converter-example.mk para encoder.mk na pasta code e decoder.mk na pasta decode.

f. Alterar os *makefiles* de forma a compilarem os novos ficheiros e criarem os executáveis com os nomes encoder e decoder:

   i. Mudar a linha "ASN_PROGRAM ?= converter-example" para "ASN_PROGRAM ?= encoder" na pasta code (mudar para decoder no *makefile* da pasta decode);

   ii. Mudar converter-example.c para encoding.c (ou decoding.c) na linha que contem ASN_PROGRAM_SRCS;

   iii. Para compilar executar o comando "make -f encoding.mk" ou "make -f decoding.mk".

Codificador

Para codificar uma mensagem SNMP usando a biblioteca **asn1c** as diversas estruturas de dados deverão ser preenchidas. O exemplo aqui mostrado será para um set-request do valor inteiro 1 para o valor da instância 0 do objecto com o OID 1.2.3.4.

1. O primeiro passo será definir um tipo de estrutura que irá conter o valor e o tipo de variável a alterar. Isto pode ser feito usando a estutura SympleSyntax_t ou ApplicationSyntax_t, dependo do tipo de variável a alterar:

   a. SympleSyntax_t se for do tipo integer_value, string_value ou objectID_value

   b. ApplicationSyntax_t se for do tipo IpAddress_t, Counter32_t, TimeTicks_t, Opaque_t, Counter64_t, Unisnged32_t

Por exemplo, definir uma varável simple do tipo SympleSyntax_t usando um integer_value:

```
SimpleSyntax_t* simple;
simple = calloc(1, sizeof(SimpleSyntax_t));
simple->present = SimpleSyntax_PR_integer_value;
simple->choice.integer_value = integer_value;
```

O campo present indica qual dos tipos de variáveis será guardado nesta estrutura e o campo choice é onde o valor será guardado.

2. O próximo passo é declarar uma variável do tipo `ObjectSyntax_t` que irá conter a variável simple:

```
ObjectSyntax_t* object_syntax;
object_syntax = calloc(1, sizeof(ObjectSyntax_t));
object_syntax->present = ObjectSyntax_PR_simple;
object_syntax->choice.simple = *simple;
```

3. Declarar uma variável do tipo `ObjectName_t` que irá conter o nome/OID do objeto:

```
ObjectName_t* object_name;
object_name = calloc(1, sizeof(ObjectName_t));
object_name->buf = name;
object_name->size = name_size;
```

name: é do tipo uint8_t* e contem o OID do objeto
name_size: é do tipo size_t e contem o tamanho o OID do objeto

4. Declarar uma variável do tipo VarBind_t para conter as variáveis object_name e object_syntax:

```
VarBind_t* var_bind;
var_bind = calloc(1, sizeof(VarBind_t));
var_bind->name = *object_name;
var_bind->choice.present = choice_PR_value;
var_bind->choice.choice.value = *object_syntax;
```

5. As variáveis do tipo VarBind_t têm que ser depois adicionadas a uma variável do tipo VarBindList_t:

```
VarBindList_t* varlist;
varlist = calloc(1, sizeof(VarBindList_t));
int r = ASN_SEQUENCE_ADD(&varlist->list, var_bind);
```

6. A lista de variáveis, juntamente com o request_id, deve depois ser inserida numa variáveldo tipo SetRequest_PDU_t:

```
SetRequest_PDU_t* setRequestPDU;
setRequestPDU = calloc(1, sizeof(GetRequest_PDU_t));
setRequestPDU->request_id = requestID;
setRequestPDU->error_index = 0;
setRequestPDU->error_status = 0;
setRequestPDU->variable_bindings = *varlist;
```

   requestID: valor duma *tag* que identificará o comando de forma a que o gestor SNMP consiga identificar a resposta correspondente do agente; os campos error_index e error_status são inicializados a 0 nas primitivas/comandos enviadas do gestor para o agente

7. A estrutura `setRequestPDU` é depois inserida numa estrutura do tipo PDUs_t :

```
PDUs_t *pdu;
pdus = calloc(1, sizeof(PDUs_t));
pdus->present = PDUs_PR_set_request;
pdus->choice.set_request = *setRequestPDU;
```

   presente: indica qual o tipo do PDU

8. Em seguida, o PDU é codificado utilizando o esquema ASN.1/BER.

```
asn_enc_rval_t ret = asn_encode_to_buffer(0, ATS_BER,
&asn_DEF_PDUs, pdu, buffer, buffer_size);
```

   ATS_BER: Tipo de encoding a ser utilizado
   asn_DEF_PDUs: tipo de mensagem a ser codificado
   pdus: estrutura a ser codificado
   buffer: variável do tipo uint8_t* onde será guardado o resultado da operação
   buffer_size: variável do tipo size_t com o tamanho do buffer (que já deve estar determinado de antemão, i.e., o espaço para o buffer deve já estar alocado em memória)
   ret: variável para guardar o estado da operação; contem os campos encoded (número de bytes resultantes, -1 em caso de erro) e failed_type (que tipos não foram codificados)

9. As variáveis buffer e ret vão ser utilizadas para construir uma variável do tipo ANY_t.

```
ANY_t* data;
data = calloc(1, sizeof(ANY_t));
data->buf = buffer;
data->size = ret.encoded;
```

10. Finalmente deverá ser construída a estrutura de dados que inclua a versão do protocolo, a *community string* e os dados (buffer codificado em BER).

```
Message_t* message;
message = calloc(1, sizeof(Message_t));
message->version = version;
message->community = community;
message->data = *data;
```

version: a versão do protocolo, do tipo long
community: a *communty string* do tipo OCTECT_STRING_t
data: o pdu codificado em asn1 do tipo ANY_t

11. Finalmente, tal como aconteceu com o PDU, esta deverá ser codificada em BER para um buffer final.

```
asn_enc_rval_t ret = asn_encode_to_buffer(0, ATS_BER,
&asn_DEF_Message, message, buffer_final, buffer_final_size);
```

Descodificador

1. Para fazer a descodificação duma mensagem SNMP que está num buffer_final:

```
Message_t *message = 0;
asn_dec_rval_t rval = asn_decode(0, ATS_BER, &asn_DEF_Message,
(void **)&message, buffer_final, buffer_final_size);
```

buffer_final: variável do tipo uint8_t que contem os dados codificados
buffer_final_size: variável do tipo size_t com o tamanho do buffer_final
rval: retorno da função asn_decode; contem os campos consumed (bytes descodificados, -1 se erro) e code (código de erro)

2. A mensagem irá conter o campo data que poderá ser descodificado para uma estrutura do tipo PDUs_t. Este campo (data) pode ser acedido diretamente a partir do objeto message, fazendo apenas message->data. O campo data, é do tipo ANY_t:

```
PDUs_t* pdu = 0;
asn_dec_rval_t rval = asn_decode(0, ATS_BER, &asn_DEF_PDUs,
(void **)&pdu, message->data.buf, message->data.size);
```

Os restantes campos, à exceção dos valores contidos numa estrutura VarBindList_t, poderão ser obtidos diretamente. Todas as estruturas que podem ter diferentes tipos de dados (que possuem *unions*), têm o campo “present” que indica que tipo de dados possuem.

3. Os dados contidos numa *sequence of* de uma VarBindList_t poderão ser obtidos da seguinte forma:

```
PDUs_t* pdu = decodePDUS();
VarBindList_t var_bindings = pdu->choice.set_request.variable_bindings;
int var_list_size = var_bindings.list.count;
VarBind_t* var_bind = var_bindings.list.array[0];
```

var_list_size: o número de elementos VarBind_t contidos na lista VarBindList_t
var_bind: o primeiro elemento dessa lista

Debug

1. A biblioteca possui uma função que permite imprimir os dados em XER (*XML Encoding Rules*). Este formato permite que seja facilmente verificado o conteúdo de cada objeto.

```
xer_fprint(file_output_descriptor, &asn_DEF_Message, message);
```

file_output_descriptor: apontador para o ficheiro de *output*.; normalmente stdout
asn_DEF_Message: define o formato do objeto a codificar
message: objeto a codificar, neste caso do tipo Message_t

Sockets

1. Para criar um *socket* UDP que esteja à escuta numa porta designada:

```
struct sockaddr_in addr;
addr.sin_family = AF_INET;
addr.sin_port = htons(9999);
addr.sin_addr.s_addr = htonl(INADDR_ANY);
int sock = socket(AF_INET, SOCK_DGRAM, 0);
socklen_t udp_socket_size = sizeof(addr);
bind(sock, (struct sockaddr *)&addr, udp_socket_size);
```

AF_INET: família de endereços a utilizar
9999: porta a onde o *socket* irá estar à escuta
INADDR_ANY: IPs a que o *socket* estará ligado, neste caso todos os da máquina
sock: *socket* que permite ouvir os pacotes recebidos na porta 9999
bind: associa o *socket* a um endereço IP e porta específicos, indicados na estrutura sockaddr_in

2. Ler dados do *socket*:

```
int recv = recvfrom(sock, buffer, buffer_size, 0, (struct sockaddr *)&addr,
&udp_socket_size);
```

sock: *socket* de onde os dados vão ser lidos
buffer: variável do tipo uint8_t a onde vão ser guardados os dados
buffer_size: tamanho do buffer em bytes (pelo menos 1024 bytes, mas lembrar que uma mensagem SNMP pode ocupar até um datagrama UDP completo, quase 64Kbytes)
0: flags
addr: estrutura que contem os parâmetros do *socket*
udp_socket_size: tamanho da estrutura addr

3. Para escrever dados para o *socket*, este tem que ser criado como indicado no ponto dois, com algumas diferenças (neste caso não é necessário fazer o *bind* do *socket*).

```
struct sockaddr_in addr;
addr.sin_family = AF_INET;
addr.sin_port = htons(port);
addr.sin_addr.s_addr = inet_addr(ip);
int sock = socket(AF_INET, SOCK_DGRAM, 0);
socklen_t udp_socket_size = sizeof(addr);
```

4. Para enviar os dados:

```
int sent = sendto(sock, buffer, buffer_size, 0, (struct sockaddr *)&addr,
udp_socket_size);
```