Codificção e descodificação de PDUs SNMPv2c

Gestão de Redes, MIEI

Carlos Manuel Magalhães da Silva A75107

Pedro João Novais Cunha A73958

11 de Fevereiro, 2018

Contents

1 Introdução

Neste trabalho prático foi desenvolvida uma API que permite a codificação e descodificação de PDUs da norma SNMPv2c. A API foi desenvolvida de forma a que seja utilizada de forma fácil pelo utilizador, embora permita que este possa utilizar de forma a que esteja mais ciente dos processos necessários para a codificação e descodificação dos PDUs.

2 Estratégia Seguida

Através da ferramenta **asn1c** (disponibilizada em **https://github.com/vlm/asn1c**), em conjunto com as regras, definições de mensagens e PDUs presentes em ASN.1, foi gerada uma outra API que definia um conjunto de tipos e estruturas que são necessárias e importantes para o desenvolvimento do projecto.
Estava então definida a base necessária para suportar o desenvolvimento da API para codificação e descodificação de PDUs SNMPv2c. Foram seguidas duas abordagens face ao problema.
Por um lado, foram criadas várias funções que permitem, seguindo uma certa ordem, a codificação e descodificação de PDUs. Desta forma o utilizador da API está mais ciente dos passos necessários para atingir o fim.
Por outro lado, apesar de estarem disponíveis as várias funções para que se possa utilizar a API de uma forma mais detalhada, simplificou-se também o uso desta, reduzindo o nível de conhecimento sobre o seu funcionamento mais detalhado. Esta simplificação foi efetuada, disponibilizando-se uma série de funções (equivalente ao número de primitivas) que permitem, da mesma forma, a utilização de todas as primitivas necessárias.

3 Estrutura da API

A API foi desenvolvida dividindo-se as várias instâncias necessárias, por módulos, de forma a que fique mais organizada e estruturada e, ao mesmo tempo, mais compreensível o seu acesso.
Assim foram criados módulos para produzir as várias instâncias, como se explica a seguir.

Para a codificação:

- Para criar e atribuir valores às estruturas SimpleSyntax e ApplicationSyntax foi criado um módulo para cada uma destas. A funcionalidade destes é providenciar funções que inserem nas estruturas os diferentes tipos de valores

que estas podem albergam e os seus respectivos tipos. De igual forma, para a estrutura ObjectName, são disponibilizadas funções que auxiliam a criação da própria estrutura e atribuem valores a esta, fazendo também o *parse* de OID's. A mesma lógica segue para a estrutura VarBindList.

- Depois de criadas as estruturas SimpleSyntax e ApplicationSyntax, existe também um módulo que permite que seja criado um ObjectSyntax, onde é possível atribuir a este mesmo uma das estruturas referidas. É de igual forma possível criar um ObjectSyntax vazio, funcionalidade importante para primitivas como o GetRequest, GetNextRequest, entre outras.

- Para albergar este ObjectSyntax e nao só, são disponibilizadas uma série de funções que permitem que se crie e associe à estrutura VarBind novos valores e estruturas. Podemos tomar como exemplo de valores que se podem associar à estrutura VarBind, o ObjectSyntax já referido, um valor que expressa Unspecified, etc.

- Esta VarBind faz parte de uma VarBindList, que se assume como um conjunto de várias VarBinds. São providenciadas funções para criar a estrutura e também adicionar valores (valores estes do tipo da estrutura VarBind) a esta.

- Depois de estar bem definida a estrutura VarBindList, pode finalmente ser criado um PDU. Este PDU pode ser do tipo PDU ou um BulkPDU. Para qualquer um dos dois, existem funções responsáveis pela sua criação. A diferença entre estes dois tipos está em alguns campos da sua estrutura. Enquanto que um destes tem na sua estrutura campos com infomação sobre várias tipos de erros (PDU), o outro possui campos como *non repeaters e max repetitions* (BulkPDU).

- O PDU que é criado, dependendo do seu tipo (*request id*), é integrado numa outra estrutura, denominada por PDUs. A API permite que sejam adicionados a esta estrutura os vários tipos de PDU.

- Finalmente é possível codificar o PDU. Foram criadas funções para criar e preencher os devidos campos da Message, nomeadamente a versão, *community string* e o PDU codificado. Depois de tudo isto, a Message é também codificada.

Além de todos módulos referidos anteriormente, foi criado ainda um outro, que interage com os restantes. Este permite que os campos necessários a cada PDU, como a versão, oid, valor a alterar ou não, entre outros, sejam codificados para o buffer final e que toda as instruções relativas a uma primitiva sejam agregadas, não

sendo necessário o utilizador interagir com outros módulos.

Para a descodificação:

- É disponibilizada uma função que, através do valor hexadecimal recebido, transforma os bytes numa estrutura do tipo PDUs. Inicialmente faz a descodificação para uma estrutura do tipo Message e, só a partir desta, descodifica a estrutura PDUs criada no processo de encoding.
- Após o PDU descodificado, a API disponibiliza uma função de parse geral do PDU. O resultado desta função é expresso numa estrutura que possui todos os campos extraídos do que foi recebido. Após efetuada a função, o utilizador poderá aceder à estrutura criada e manipular os dados da forma que bem entender.

4 Exemplos de Utilização e Aplicações de teste

Nesta secção irão ser demonstrados exemplos de construção (codificação e descodificação) de alguns tipos de PDUs, dando a conhecer a forma de interação com a API que foi desenvolvida.
Por razões de teste, foram também criadas duas aplicações que colocam em prática a API, de forma a que seja possível numa destas, fazer a codificação de todos os tipos de PDUs disponíveis e, noutra, fazer exatamente a descodificação do que foi gerado pela aplicação de codificação.
A aplicação de codificação foi desenhada de forma a que leia o *input* de um ficheiro (seguindo uma certa sintaxe que será demonstrada a seguir), codifique o PDU envolvido num *message* e, em seguida, envie por UDP o resultado em bytes da codificação.
A aplicação de descodificação recebe por UDP uma *message*, que contém o PDU, e faz todo o *parse* deste. Depois de extraídos todos os campos relevantes do PDU(s),é escrito para um ficheiro o resultado da descodificação (formato este que será igualmente demonstrado a seguir).

4.1 Exemplos para contrução de ficheiro de *input*

Para construir o ficheiro de input é necessário ter informação sobre qual a identificação dos tipos de PDUs, tipos de valores e tipos de respostas. Assim a informação necessária é:

```
IDs de PDUs
```

```
1 = get
2 = getNext
3 = getBulk
4 = response
5 = set
6 = informe
7 = trap

IDs de tipos de valores
1. Long
2. String
3. OID
4. IpAddress
5. CounteValue(32bit)
6. TimeTicks
7. Arbitrary_value
8. BigCounterValue(64bit)
9. Unsigned_integer_value

IDs Tipos de Resposta
1.Unspecified
2.No such object
3.No such instance
4.End of Mib view
5.Success
```

O ficheiro de input a passar como argumento à aplicação de codificação só poderá conter um tipo de PDU e terá a seguinte estrutura:

- Se o tipo de PDU for get:

  ```
  1 2 private 1 #Numero de PDUs presentes(p.e get, getNext,...), versão, community
  1.2.2.2 #oid
  ```

- Se o tipo de PDU for getNext

  ```
  1 2 private 2 #Numero de PDUs presentes(p.e get, getNext,...), versão, community
  1.2.2.2 #oid
  ```

- Se o tipo de PDU for getBulk

```
1 2 public 3 #Numero de PDUs presentes(p.e get, getNext,...), versão, community
1 10 #non-repeaters, max_repetitions
1.2.2.21 #oid
```

- Se o tipo de PDU for response

```
2 2 public 4 #Numero de PDUs presentes(p.e get, getNext,...), versão, community
0 0 #error_status, error_index
1.2.3.2 5:2:response #oid, [tipo de resposta]:[tipo do valor]:[valor]
1.2.2.2 1 #oid, tipo de resposta (se nao for bem sucedida)
```

- Se o tipo de PDU for set

```
2 2 public 5 #Numero de PDUs presentes(p.e get, getNext,...), versão, community
1.2.3.4 2:ola #oid, [tipo do valor]:[valor a mudar]
1.2.3.4.5 4:192.168.1.1
```

- Se o tipo de PDU for informe

```
1 2 public 6 #Numero de PDUs presentes(p.e get, getNext,...), versão, community
1.1.1.1.1.1 #oid
```

- Se o tipo de PDU for trap

```
1 2 public 7 #Numero de PDUs presentes(p.e get, getNext,...), versão, community
1.2.3.2.1 #oid
```

4.2 Execução das aplicações de teste

Antes de ser executado o comando que codifica o PDU e envia para a aplicação de descodificação, é necessário executar primeiro a aplicação de descodificação, já que esta fica à espera que lhe chegue os bytes vindos da codificação. Para executar então esta aplicação é necessário fornecer a porta em que vai estar à escuta e um nome para o ficheiro onde vai ser executado o *output*.

```
./NOME_EXECUTÁVEL [PORTA_ESCUTA] [FICHEIRO_OUTPUT]
```

Após ter sido criado um ficheiro com a informação necessária para a codificação, resta executar a aplicação que implementa a API desenvolvida, para isso:

```
./NOME_EXCUTÁVEL [IP_Descodificador] [PORTA__Descodificador] [FICHEIRO_INPUT]
```

4.3 Resultado final gerado

Tendo em consideração este *input*

```
1 2 public 5
1.2.3.4.5 4:192.168.1.1
```

que é passado como argumento à aplicação de codificação, após ser codificado e enviado para a aplicação de descodificação, o resultado final é

```
Version : 2  Community String : public
error status:0 error index:0
set 1.2.3.4.5 192.168.1.1
```

Outro exemplo de um possível *input* seria, desta ve demonstrando um getBulk,

```
1 2 public 3
1 10
1.2.2.21
```

com o resultado final do lado do descodificador após codificação de

```
Version : 2  Community String : public
getbulk non repeaters:1 max repetitions:10 1.2.2.21  UnSpecified
```

References

[1] Fábio Gonçalves, Bruno Dias - Tutorial ”Codificação/Descodificação de PDUs SNMPv2c”

[2] https://github.com/vlm/asn1c/blob/master/doc/asn1c-usage.pdf

[3] http://asn1-playground.oss.com/