Relatório

Trabalho Prático 3

Fevereiro 2018

Universidade do Minho

Mestrado Integrado em Engenharia de Telecomunicações e Informática

Gestão de Redes

Raúl Filipe Cruz Antunes A75577

Índice

Introdução 2
SNMP 2
Unidade Protocolar de Dados 3
Codificação e Descodificação de PDUs 4
Documentação da API 4
Encoder 4
Decoder 8
Conclusão 9

Introdução

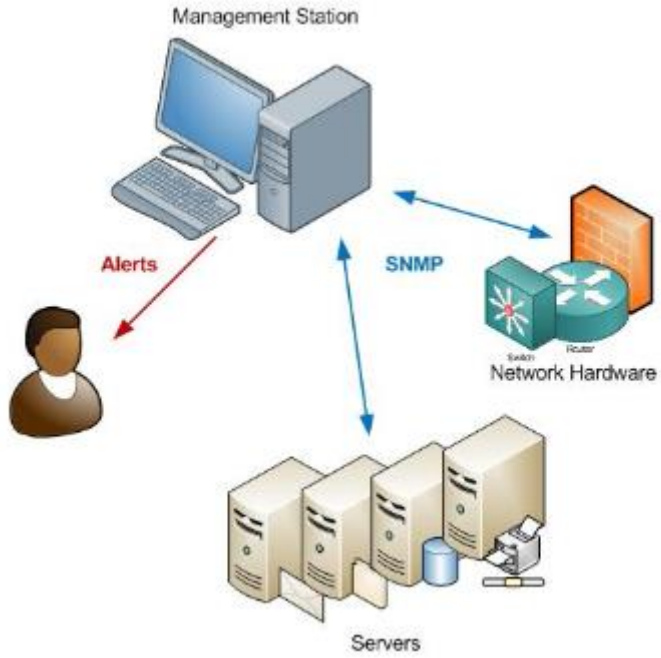
O presente projeto recai sobre a elaboração de uma API para codificar e descodificar PDUs SNMPv2c tendo em conta as regras definidas de codificação (ASN.1/BER). Estes módulos contém as classes que servem de base à interação com o protocolo SNMP. O objetivo encontra-se no envio de dados (IP, porta UDP, tag…) num PDU codificado através de um socket e/ou ficheiro e receber esse mesmo PDU no descodificador e obter a mesma mensagem enviada.

SNMP

O Simple Network Management Protocol é, como o nome indica, um protocolo standard que monitoriza hardware e software de inúmeros fabricantes e requer apenas alguns componentes básicos, como uma estação gestora e agentes. Os agentes são instalados nos equipamentos monitorizados, e enviam informação à estação gestora. Existem normalmente vários agentes a comunicar com a estação gestora, através de uma rede, de forma a manter uma MIB (Management Information Base) atualizada e com dados estatísticos e configurações importantes de toda a rede ou sistemas geridos por esse gestor. O SNMP pode também ser configurado para enviar alertas, por vários meios, em caso de erros ou apenas como avisos de possível mau funcionamento, alarmes previamente configurados, entre outros.

Este protocolo tem incontáveis aplicações no mundo tecnológico, entre as quais um chat com mensagens codificadas através de 2 meios que

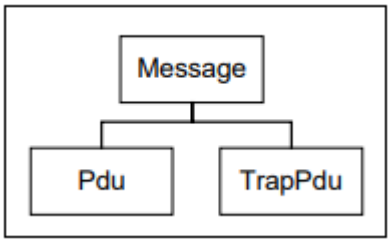
incorporam o SNMP e que estejam ligados a um sistema de gestão que os monitorize.



Unidade Protocolar de Dados

A comunicação entre gestor e agente no protocolo SNMP é feita recorrendo a 5 PDUs: get-request, getresponse get-next-request, set-request e trap.

O módulo de Unidades Protocolares contém classes que encapsulam estas PDUs. A figura que se segue mostra a estrutura deste módulo.



A classe Pdu encapsula as PDUs get-request, get-response get-next-request e set-request. A única diferença na estrutura destas é o tipo, de resto têm estrutura iguais. A classe TrapPdu encapsula a PDU Trap que possui uma estrutura diferente das restantes PDUs.

Codificação e Descodificação de PDUs

Antes de enviar ou receber uma PDU SNMP, é necessário proceder à sua codificação/descodificação. A codificação das PDUs é feita usando as Basic Encoding Rules (BER) para a Abstract Syntax Notation One (ASN.1).

Este módulo é composto por várias classes estáticas, sem uma estrutura hierárquica. A mecânica deste módulo consiste numa classe principal que recebe as PDUs e que invoca outras classes do módulo para realizar tarefas mais específicas, distribuindo o processo de codificação e descodificação.

Documentação da API

De forma a gerar o código responsável pela codificação e descodificação foi utilizado um tutorial disponibilizado pelo docente e acrescentado métodos de forma a completá-lo. Variáveis contidas no tutorial e sua posterior explicação não voltarão a ser explicadas neste relatório.

Encoder

```
printf("Insira o comando, no final adicione o valor inteiro, versao de SNMP, requestID\n");
char guarda[2048];
read(0,guarda,sizeof(guarda));
int contador = 0;
uint8_t buffer[2048];
size_t buffer_size = sizeof(buffer);
uint8_t buffer_final[2048];
size_t buffer_final_size = sizeof(buffer_final);
char *split = strtok(guarda," ");
char* snmp; char* com; char* ip; char* porta; uint8_t* name; int val; int versao; int reqID;
```

O projeto começa com a codificação da mensagem, daí que seja pedido ao utilizador que insira o comando que deseja ver após a mensagem ser descodificada. Também são inicializadas as variáveis.

```
while(split != NULL)
{
    switch(contador){
    case 0:
        snmp = split; printf("SNMP: %s\n",snmp);
        split=strtok(NULL," :"); contador ++;
        break;
    case 1:
```

O excerto de código é seguido do anterior e pretende guardar em variáveis as divisões do comando inserido pelo utilizador. Estas por sua vez serão colocadas nas estruturas explicadas posteriormente.

```
SimpleSyntax_t* simple;
simple = calloc(1, sizeof(SimpleSyntax_t));
simple->present = SimpleSyntax_PR_integer_value;
simple->choice.integer_value = val;         //Atribuo o valor inteiro

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

size_t name_size = sizeof(name);

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

A variável "val" e "name" contém respetivamente o valor inteiro que o utilizador escolheu e o OID. Note-se que ao executar "simple->choice.integer_value = val", além de se guardar o valor também será devolvido no descodificador pelo "choice" de uma var_bindList.

```
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;
```

Os dados são colocados numa estrutura "varBind_t*" que depois serão colocados numa lista.

```
VarBindList_t* varlist;
varlist = calloc(1, sizeof(VarBindList_t));
int r = ASN_SEQUENCE_ADD(&varlist->list, var_bind); //
int r1 = ASN_SEQUENCE_ADD(&varlist->list, var_bind2);
int r2 = ASN_SEQUENCE_ADD(&varlist->list, var_bind3);
int r3 = ASN_SEQUENCE_ADD(&varlist->list, var_bind4);
```

A cada vez que se repete o código mostrado anteriormente de forma a enviar todas as variáveis é adicionada á "varBindList_t*" todas as estruturas "varBind_t*".

```
if(snmp == "snmpset"){ //SNMPSET
SetRequest_PDU_t* setRequestPDU;
setRequestPDU = calloc(1, sizeof(GetRequest_PDU_t));
setRequestPDU->request_id = reqID; //Request ID
setRequestPDU->error_index = 0;
setRequestPDU->error_status = 0;
setRequestPDU->variable_bindings = *varlist;
```

É verificado com o comando utilizado de forma a utilizar a estrutura correta, seja esta getRequest, setRequest ou TrapNotification.

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

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

É finalmente inicializado o tipo de PDU e é colocado na variável “uint8_t buffer” o resultado da operação.

```
Message_t* message;
message = calloc(1, sizeof(Message_t));
message->version = versao; //versao snmp
message->community.buf = strdup(com);
message->community.size = strlen(com);
message->data = *data;
```

Nesta fase é construída a estrutura de dados que inclui a versão do snmp, a community string e os dados.

```
xer_fprint(stdout,&asn_DEF_Message,message); //verificar a estrutura
```

O “xer_fprintf” permite averiguar a situação da estrutura de dados no presente momento, segue-se a imagem com a situação onde podemos verificar o resultado em hexadecimal.

```
</PDUs>
<Message>
    <version>-1078501932</version>
    <community>70 75 62 6C 69 63</community>
    <data>
        A3 46 02 04 BF B7 5D D6 02 01 00 02 01 00 30 38
        30 0C 06 04 31 2E 32 2E 02 04 BF B7 5D D1 30 0C
        06 04 31 32 37 2E 02 04 BF B7 5D D1 30 0C 06 04
        73 6E 6D 70 02 04 BF B7 5D D1 30 0C 06 04 70 75
        62 6C 02 04 BF B7 5D D1
    </data>
</Message>
```

```
asn_enc_rval_t ret2 = asn_encode_to_buffer(0, ATS_BER, &asn_DEF_Message, message, buffer_final, buffer_final_size);
```

Por fim a mensagem é codificada em “BER” para um buffer final.

```
Insira o comando, no final adicione o valor inteiro, versao de SNMP, requestID
snmpget public 127.0.0.1:161 1.2.3.4 10 2 1
SNMP: snmpget
Community: public
IP: 127.0.0.1
Porta: 161
OID: 1.2.3.4
Valor: 10
Versao: 2
RequestID: 1
```

A inicialização do comando no terminal é realizada conforme a imagem acima e após os passos descritos acima é transformada em binário.

```
FILE *fp = fopen("EncoderResult.txt", "w");
if (fp == NULL)
{
    printf("Error opening file!\n");
    exit(1);
}
int h; int binary;
for(h = 0; h < buffer_final_size ; h++){
    binary = convertDecimalToBinary(buffer_final[h]);
    fprintf(fp, "%d" , binary);
}

fclose(fp);
```

Ao abrir o ficheiro e escrever o resultado da operação, é primeiro enviado o buffer para o conversor para binário e só depois escrito no ficheiro.

encoder.c × EncoderResult.txt ×

```
1100001010110101001011111110110111101
```

```
struct sockaddr_in addr;
addr.sin_family = AF_INET; //AF_INET: família de endereços a utilizar
addr.sin_port = htons(8080); //161: porta para escrever get,set,get-bulk
addr.sin_addr.s_addr =  inet_addr("127.0.0.1"); //Ip da porta
int sock = socket(AF_INET, SOCK_DGRAM, 0); //sock: socket  que permite ouvir os pacotes recebidos na porta 9999
socklen_t udp_socket_size = sizeof(addr);

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

É ainda aberto o socket caso a comunicação seja efetuada assim. Sendo possível comunicar com o descodificador por sockets ou ficheiros.

Decoder

```
int ee = 0;
uint8_t buffer_final[2048];
size_t buffer_final_size = sizeof(buffer_final);
```

É inicializado com o "buffer_final" de forma a receber o PDU, e com a variável "ee" que permite escolher caso queira se descodificar a partir de uma mensagem no ficheiro ou caso "ee = 1" seja feita a comunicação por sockets.

```
FILE *fp = fopen("/home/raul/Desktop/encode/EncoderResult.txt", "r");
if (fp == NULL)
{
    printf("Error opening file!\n");
    exit(1);
}

char buf[BUFSIZ] = {'\0'};

while (fgets(buf,BUFSIZ, fp)!=NULL)
        fputs(buf, stdout);

buffer_final[BUFSIZ]; int h;

for(h=0;h<BUFSIZ;h++)
    buffer_final[h] = buf[h];

printf("Valores no uint8_t: %s",buffer_final);
size_t buffer_final_size = sizeof(buffer_final);
```

No primeiro caso, onde "ee = 0" é aberto o ficheiro que se encontra na pasta "encode" e é colocado no "buffer_final" o valor binário contido nele.

```
struct sockaddr_in addr;
addr.sin_family = AF_INET; //AF_INET: família de endereços a utilizar
Message_t *message2 = 0;
asn_dec_rval_t rval = asn_decode(0, ATS_BER, &asn_DEF_Message, (void **)&message2, buffer_final, buffer_final_size);
bind(sock, (struct sockaddr *)&addr, udp_socket_size); //bind: associa o socket a um endereço IP e porta específicos, indicados na estrutura sockaddr_in
int recv = recvfrom(sock, buffer_final, buffer_final_size, 0, (struct sockaddr *)&addr, &udp_socket_size); }//Acaba o ELSE
```

Caso seja por socket, a função é similar à do encoder e o resultado recebido é guardado diretamente no buffer_final.

Começa-se a proceder à descodificação dos dados guardados em "buffer_final" que estão codificados.

```
PDUs_t* pdu2 = 0;
asn_dec_rval_t rval2 = asn_decode(0, ATS_BER, &asn_DEF_PDUs, (void **)&pdu2, message2->data.buf, message2->data.size);
```

A mensagem contém o campo data que é descodificado para uma estrutura do tipo PDUs_t

```
PDUs_t* pdu3 = pdu2;
VarBindList_t var_bindings = pdu3->choice.set_request.variable_bindings;
int var_list_size = var_bindings.list.count;
VarBind_t* var_bind = var_bindings.list.array[0];
```

Através da variável “var_bind” podemos selecionar a posição do array que queremos imprimir, tendo elas as variáveis enviadas no codificador.

```
 printf("A mensagem: \n);
 int i;
for(i=0; i<var_list_size; i++){
 var_bind = var_bindings.list.array[i];
printf("%s ", var_bind);}
```

Por fim é imprimida a mensagem final através da leitura das posições do array.

Conclusão

Sendo este o último dos 3 projetos da cadeira, e sem dúvida o que impõe uma maior dificuldade, ao contrário dos outros dois em que a sua resolução foi exatamente dentro do pedido, este projeto não chegou ás expetativas impostas por mim.

Apesar do encoder funcionar e realmente mandar os dados para o ficheiro ou socket e o outro lado receber, no que toca ao decoder e na sua execução segue-se um “core dumped” mesmo seguindo o tutorial dado pelo professor. Apesar de ter consultado 2 vezes os autores do tutorial, e eu mesmo ter alterado o programa de forma a possibilitar a descodificação correta, não foi possível e o programa “crasha” sempre.

De resto todos os outros requisitos foram adicionados.