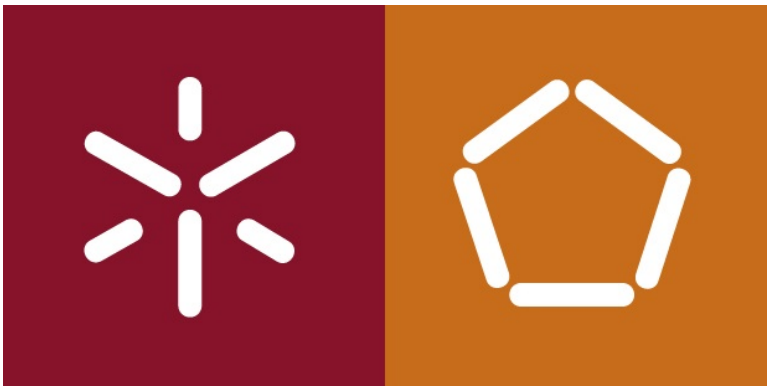

Trabalho Prático 3

# Gestão de Redes MIEI
# Codificção e descodificação de PDUs SNMPv2c


Bruno Machado - Paulo Guedes
A74941 - A74411

11 Fevereiro 2018

# 1 Introduction

O objetivo principal deste projeto é a criação de uma API para codificação e descodificação de PDUs da norma SNMPv2c tendo em consideração o par linguagem/regras de codificação em que foram definidos (ASN.1/BER).

Na parte de encode a partir dum grupo de dados/argumentos (endereço IP e porta UDP, tag, tipo de primitiva, OIDs, valores, community string, etc.) construir um PDU SNMPv2 seguindo os métodos de codificação descritos na norma e colocar o resultado num buffer binário (pedaço de memória contíguo). Como resultados adicionais deve disponibilizar-se informação sobre o sucesso ou insucesso do processo de codificação.

No que disse respeito ao código relativo ao decode será necessário a partir dum grupo de buffer binário tentar descodificar um PDU SNMPv2c seguindo o processo inverso dos métodos de codificação descritos na norma. O resultado deve incluir os dados/argumentos originais contidos no PDU (endereço IP e porta UDP, tag, tipo de primitiva, OIDs, valores, community string, etc.). Como resultados adicionais deve disponibilizar-se informação sobre o sucesso ou insucesso do processo de descodificação.

## 2 Encode

Para que fosse possível o encode de um PDU SNMPv2 foi criada uma API para a criação de todos os estrutura que constituiem um buffer final desde SimpleSyntax_t e ApplicationSyntax_t até à estrutura final Message_t.

**SimpleSyntax_t* createSimpleSyntax(int flag, void* value);**

Função que cria uma estrutura **SimpleSyntax_t***, alocando memória para esta e preenchendo com o valor "**value**", dependendo da **flag**.

Caso a flag seja **0**, assume que o campo "present"é um **SimpleSyntax_PR_integer_value** e preenche a "choice"**integer_value** com o value assumindo que este é um **long***.

Caso a flag seja **1**, assume que o campo "present"é um **SimpleSyntax_PR_string_value** e preenche a "choice"**string_value** com um **OCTET_STRING_t**, criado a partir do **char*** dado como value.

Caso a flag seja **2**, assume que o campo "present"é um **SimpleSyntax_PR_objectID_value** e preenche a "choice"**objectID_value** com um **OBJECT_IDENTIFIER_t**, criado a partir do **char*** dado como value.

**ApplicationSyntax_t* createApplicationSyntax(int flag, void* value);**

Função que cria uma estrutura **ApplicationSyntax_t***, alocando memória para esta e preenchendo com o valor "**value**", dependendo da **flag**.

Caso a flag seja **0**, assume que o campo "present"é um **ApplicationSyntax_PR_ipAddress_value** e preenche a "choice"**ipAddress_value** com um **OCTET_STRING_t**, criado a partir do **char*** dado como value.

Caso a flag seja **1**, assume que o campo "present"é um **ApplicationSyntax_PR_counter_value** e preenche a "choice"**counter_value** com o value assumindo que este é um **Counter32_t***.

Caso a flag seja **2**, assume que o campo "present"é um **ApplicationSyntax_PR_timeticks_value** e preenche a "choice"**timeticks_value** com o value assumindo que este é um **TimeTicks_t***.

Caso a flag seja **3**, assume que o campo "present"é um **ApplicationSyntax_PR_arbitrary_value** e preenche a "choice"**arbitrary_value** com um **OCTET_STRING_t**, criado a partir do **char*** dado como value.

Caso a flag seja **4**, assume que o campo "present"é um **ApplicationSyntax_PR_big_counter_value** e preenche a "choice"**big_counter_value** com o value assumindo que este é um **INTEGER_t***.

Caso a flag seja **5**, assume que o campo "present"é um **ApplicationSyntax_PR_unsigned_integer_value** e preenche a "choice"**unsigned_integer_value** com o value assumindo que este é um **Unsigned32_t***.

**ObjectSyntax_t* createObjectSyntax(int flag, void* value);**

Esta função cria uma estrutura **ObjectSyntax_t***, alocando memória para esta e preenchendo com o valor "**value**", dependendo da flag.

Caso a flag seja **0**, assume que o campo "present"toma como valor **ObjectSyntax_PR_simple** e preenche a "choice"**simple** com value assumindo que este é do tipo **SimpleSyntax_t***.

Caso a flag seja **1**, assume que o campo "present"toma como valor **ObjectSyntax_application_wide** e preenche a "choice"**application_wide** com value assumindo que este é do tipo **ApplicationSyntax_t***.

Caso a flag seja **2**, assume que o campo "present"toma como valor **ObjectSyntax_PR_NOTHING** e ignora o valor de value.

**ObjectName_t* getOID(char* oid)**

Dada uma string relativa a um **OID** (ex: 0.1.1.1.0), faz parse dos números e cria uma estrutura **ObjectName_t***, alocando memória e preenchendo o campo **buf** com um array **uint8_t** relativo ao OID, e preenche o campo **size** com o tamanho do **buf**.

**VarBind_t* createVarbind(ObjectSyntax_t* object_syntax, ObjectName_t* object_name);**

Cria uma estrutura VarBind_t* alocando memória para esta, e preenchendo a estrutura dependendo dos valores do **object_syntax** dado como argumento.

Caso o valor "present"do object_syntax seja **ObjectSyntax_PR_NOTHING**, preenche o campo "choice.present"da estrutura VarBind_t* com **choice_PR_unSpecified**, caso contrário preenche com **choice_PR_value**.

Preenche também os campos "name"e "choice.value"com object_name e object_syntax, respetivamente (caso o valor de present seja **choice_PR_unSpecified**, ele ignora o valor de object_syntax).

**PDU_t* createPDU(int requestID, long index, long status, VarBindList_t* varlist)**

Cria uma estrutura **PDU_t***, alocando memória para esta, os campos "request_id", "error_index", "error_status"e "variable_bindings"com os valores dados como argumentos requestID, index, status e varlist, respetivamente. O processo de **descodificação** assume os respetivos valores:

requestID = **0** -¿ primitiva **getRequest**

requestID = **1** -¿ primitiva **getNextRequest**

requestID = **2** -¿ primitiva **getBulkRequest**

requestID = **3** -¿ primitiva **response**

requestID = **4** -¿ primitiva **setRequest**

requestID = **5** -¿ primitiva **informRequest**

requestID = **6** -¿ primitiva **trap**

requestID = **7** -¿ primitiva **report**

Para cobrir o caso de ser uma primitiva getBulkRequest (que tem campos com nomes diferentes de todos os outros PDUs), foi criada uma nova função:

**BulkPDU_t* createBulk(long requestID, long non_repeaters, long max_repeaters, VarBindList_t* var)**

Realiza o mesmo que a função **createPDU**, apenas com a diferença que em vez de preencher os campos "error_index"e "error_status", preenche os campos "non_repeaters"e "max_repeaters"com os valores tomados como argumento **non_repeaters** e **max_repeaters**, respetivamente.

**PDUs_t* createPDUs(void* pdu, int escolha)**

Cria uma estrutura **PDUs_t***, alocando memória para esta, e preenchendo os respetivos campos com os valores dados como argumento, dependendo do valor da **flag**. Caso a **flag** seja:

**0** assume que o pdu é respetivo a uma primitiva **getRequest**, atribui o o valor de **PDUs_PR_get_request** ao campo "present"e preenche o campo "choice.get_request"com o value, assumindo que este é um **GetRequest_PDU_t***.

**1** assume que o pdu é respetivo a uma primitiva **getNextRequest**, atribui o o valor de **PDUs_PR_get_next_request** ao campo "present"e preenche o campo "choice.get_next_request"com o value, assumindo que este é um **GetNextRequest_PDU_t***.

**2** assume que o pdu é respetivo a uma primitiva **getBulkRequest**, atribui o o valor de **PDUs_PR_get_bulk_request** ao campo "present"e preenche o campo "choice.get_bulk_request"com o value, assumindo que este é um **GetBulkRequest_PDU_t***.

**3** assume que o pdu é respetivo a uma primitiva **response**, atribui o o valor de **PDUs_PR_response** ao campo "present"e preenche o campo "choice.response"com o value, assumindo que este é um **Response_PDU_t***.

**4** assume que o pdu é respetivo a uma primitiva **setRequest**, atribui o o valor de **PDUs_PR_set_request** ao campo "present"e preenche o campo "choice.set_request"com o value, assumindo que este é um **SetRequest_PDU_t***.

**5** assume que o pdu é respetivo a uma primitiva **informRequest**, atribui o o valor de **PDUs_PR_inform_request** ao campo "present"e preenche o campo "choice.inform_request"com o value, assumindo que este é um **InformRequest_PDU_t***.

**6** assume que o pdu é respetivo a uma primitiva **trap**, atribui o o valor de **PDUs_PR_snmpV2_trap** ao campo "present"e preenche o campo "choice.snmpV2_trap"com o value, assumindo que este é um **SNMPv2_Trap_PDU_t***.

**7** assume que o pdu é respetivo a uma primitiva **report**, atribui o o valor de **PDUs_PR_report** ao campo "present"e preenche o campo "choice.report"com o value, assumindo que este é um **Report_PDU_t***.

**void createANY(ANY_t* data, PDUs_t* pdu)**

Codifica a estrutura **PDUs_t** utilizando o esquema **ASN.1/BER**, colocando em data-¿buf o array de **uint8_t** resultante da operação e em data-¿size o tamanho do buffer. É necessário a estrutura **ANY_t** já vir com memória alocada.

**Message_t* createMessage(ANY_t* data, int version, char* community)**

Cria uma estrutura **Message_t*** alocando memória para esta, preenchendo o campo "version"com o int dado como argumento. Preenche também o campo "community"com uma estrutura **OCTET_STRING_t*** criado com as informações do char* dado como argumento, e por fim, preenchendo o campo "data"com a estrutura **ANY_t*** dada como argumento.

**uint8_t* auxPri(int typeObject, void* objectValue, int pduType, long index, long status, unsigned long version, char* community, char** oid)**

Função auxiliar que cria a primitiva **codificada** utilizando o esquema **ASN.1/BER** para um buffer.

Inicialmente, cria uma estrutura **ObjectSyntax_t***, utilizando a função auxiliar **createObjectSyntax**, dando como argumento o valor "typeObject"e "objectValue".

typeObject = **0** -¿ objectValue é um **SimpleSyntax**

typeObject = **1** -¿ objectValue é um **ApplicationSyntax**

typeObject = **2** -¿ objectValue é **descartado**

De seguida, para cada **OID** no array de **OIDs** "oid", recebido como argumento, cria uma estrutura **ObjectName_t***, através da função auxiliar **createObjectName**, recebendo como argumento um dos OIDs, criando a estrutura **VarBind_t*** com o ObjectName_t* e ObjectSyntax_t* criado anteriormente. De seguida, adiciona cada VarBind_t* criado a uma estrutura **VarBindList_t***.

Após todos os OIDs serem tratados, é criado uma variavel void* relativa ao PDU (devido a poder ser criada uma estrutura **PDU_t*** com a função auxiliar "**createPDU**", caso a pduType seja diferente de 2, ou criada uma estrutura **BulkPDU_t*** com a função auxiliar "**createBulk**"caso a pduType seja 2, devido ao nome dos campos serem diferentes).

Após criada a estrutura do PDU, é criada a estrutura que identifica para a descodificação, o tipo de PDU que foi codificado, através do campo present. A estrutura **PDUs_t*** é criada com o auxilio da função auxiliar **createPDUs**, que toma como argumento a variável que guarda o PDU, e o pduType, sendo que:

pduType = **0** -¿ primitiva **getRequest**

pduType = **1** -¿ primitiva **getNextRequest**

pduType = **2** -¿ primitiva **getBulkRequest**

pduType = **3** -¿ primitiva **response**

pduType = **4** -¿ primitiva **setRequest**

pduType = **5** -¿ primitiva **informRequest**

pduType = **6** -¿ primitiva **trap**

pduType = **7** -¿ primitiva **report**

Criado o PDU e identificado o tipo, é de seguida codificado utilizando o esquema **ASN.1/BER**, para uma estrutura **ANY_t***, através da função auxiliar **createANY**, que recebe a estrutura PDUs_t* e a estrutura ANY_t* que irá ser preenchida.

Finalmente é criada a última estrutura, **Message_t*** que guarda todos os dados criados anteriormente, assim como a **versão** do SNMP, e a **community string**, com o auxilio da função **createMessage**. Por fim, é codificada a estrutura Message_t* para um array de **uint8 ˜** que é retornado.

**uint8_t* getRequestPri(unsigned long version, char* community, char** oid)**

Retorna um buffer com a primitiva getRequest codificada, com auxilio da função auxPri. Executa a função auxPri(2, NULL, 0, 0, 0, version, community, oid).

**uint8_t* getNextRequestPri(unsigned long version, char* community, char** oid)**

Retorna um buffer com a primitiva getNextRequest codificada, com auxilio da função auxPri. Executa a função auxPri(2, NULL, 1, 0, 0, version, community, oid).

**uint8_t* getBulkRequestPri(long non_r, long max_r, unsigned long version, char* community, char** oid)**

Retorna um buffer com a primitiva getBulkRequest codificada, com auxilio da função auxPri. Executa a função auxPri(2, NULL, 2, non_r, max_r, version, community, oid).

**uint8_t* setRequestPri(int flag, void* setValue, unsigned long version, char* community, char** oid)**

Dependendo do valor da **flag**, cria diferentes estruturas: Flag ¿=0 e ¡ 3, cria uma **SimpleSyntax_t***, com auxilio da função **createSimpleSyntax**, dando como argumento a flag e o objectValue. De seguida retorna o buffer criado pela função auxiliar auxPri(0, simple, 4, 0, 0, version, community, oid), sendo simple a estrutura criada no passo anterior. Flag ¿=4, cria uma **ApplicationSyntax_t***, com auxilio da função **createApplicationSyntax**, dando como argumento a flag e o objectValue. De seguida retorna o buffer criado pela função auxiliar auxPri(0, application, 4, 0, 0, version, community, oid), sendo application a estrutura criada no passo anterior.

**uint8_t* informRequestPri(long index, long status, unsigned long version, char* community, char** oid)**

Retorna a função auxPri(2, NULL, 5, index, status, version, community, oid).

**uint8_t* trapPri(long index, long status, unsigned long version, char* community, char** oid)**

Retorna a função auxPri(2, NULL, 6, index, status, version, community, oid).

**uint8_t* reportPri(long index, long status, char* mensagem, unsigned long version, char* community, char** oid)**

A primitiva report definiu-se que teria os valores de **error_index** e **error_status** da estrutura PDU_t preenchidos, logo recebe os argumentos index e status. Ainda mais, recebe uma mensagem a explicar o report, que será guardado numa estrutura **SimpleSyntax_t***, preenchendo como se toma-se um string. A estrutura SumpleSyntax_t* é criada com auxilio da função auxiliar createSimpleSyntax(1, mensagem).

É de seguida retornada a função auxPri(0, simple, 7, index, status, version, community, oid).

**void escreveUDP(int port, char* ip, uint8_t* buffer)**

Envia o que está no buffer, pela porta "port"para o ip "ip".

**void escreveFicheiro(char* file, uint8_t* buffer)**

Escreve o que está no buffer para um ficheiro com o nome de "file".txt. Separa cada elemento do buffer com um espaço.

# 3 Decode

Na realização do decoder foi necessário fazer processo contrário ao feito no encoder. Uma vez que muitas operações são semelhantes, esta parte será explicada de uma forma mais resumida.

**uint8_t* leUDP(int port, uint8_t* buffer)**

Função que recebe com argumento uma porta UDP e um uint8_t* que irá conter o resultado do encode de um PDU SNMPv2. Esta função espera pela chegada de um buffer uint8_t por parte do programa encoder que irá preencher a variável buffer.

**int buffer2pdus (uint8_t* buffer_final,size_t size,PDUs_t* pdus)**

A função buffer2pdus recebe como argumentos um buffer binário, o seu respetivo tamanho e um apontador para um estrutura PDUs_t vazia. Nesta função são é preenchida a estrutura PDU_t através do decode de um buffer binário e é feito o print da versão e da community string do respetivo PDU SNMP.

**int Pdus2Pdu (PDUs_t* pdu,void* pdu_t)**

A função Pdus2Pdu recebe como argumento um apontador para um estrutura PDUs_t previamente preenchida e um apontador para void*. Depois de visitado o campo present da estrutura PDU_t é feito o cast para PDU_t* ou no caso do pdu ser do tipo Get_bulk_request para BulkPDU_t * preenchendo-se os respetivos campos de cada estrutura. São também imprimidos o request_id,Error_status,Error_index e no caso do pdu Get_bulk_request são imprimidos os campos non_repeaters,max_repetitions.

**int varBindList2VarBind(VarBindList_t *vbl,VarBind_t** vn)**

A função varBindList2Varbind recebe VarbindList_t * previamente preenchido assim Varbind_t**. Nesta função é preenchido o array de VarBind_t* com as informações presentes na VarBindList* consoante o tipo de variáveis contida no campo present. No decorrer da função são também imprimidos todos os OID's presentes na VarBindList_t*.

**int ParseObjSyntax(ObjectSyntax_t* obj)**

A função ParseObjectSyntax recebe como argumento um apontador para um estrutura ObjectSyntax_t previamente preenchida. Nesta função é feito o decode de uma estrutura ObjectSyntx_t imprimindo-se os possíveis tipos e dados presentes nas subestruturas SimpleSyntax_t e ApplicationSyntax_t através da verificação do conteúdo dos campos present;