

Trabalho 5 - VLAN e SNMP

Miguel Ferreira
miguelferreira108@gmail.com
Vanessa Silva
up201305731@fc.up.pt

*Administração de Redes,
Departamento de Ciências de Computadores,
Faculdade de Ciências da Universidade do Porto*

6 de Junho de 2016

Introdução

No âmbito da unidade curricular de Administração de Redes, implementamos a rede da figura abaixo, em que R1 é um router Cisco 2691 com um módulo de comutação de 16 portas.

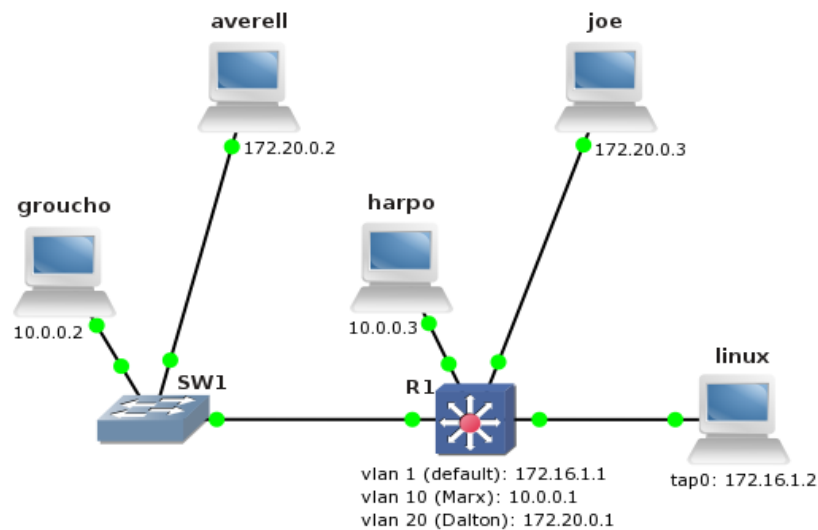


Figura 1: Rede implementada na aula.

Questões

1.

- a. Captura do primeiro ICMP *Echo Request* enviado em ambas as interfaces:

The image shows a Wireshark capture on interface f0/0. The filter is set to 'icmp'. The packet list shows several ICMP Echo (ping) requests and replies. The packet details pane for packet 91 is expanded, showing the Ethernet II, Internet Protocol Version 4, and Internet Control Message Protocol (Type 8 Echo (ping) request) fields. The data field is highlighted in red.

No.	Time	Source	Destination	Protocol	Length	Info
91	42.2677580	10.0.0.2	10.0.0.1	ICMP	118	Echo (ping) request id=0x0001, seq=0/0, ttl=255 (reply in 98)
92	42.2678140	10.0.0.2	10.0.0.1	ICMP	118	Echo (ping) request id=0x0001, seq=1/256, ttl=255 (reply in 99)
97	43.2485160	10.0.0.2	10.0.0.1	ICMP	118	Echo (ping) request id=0x0001, seq=2/512, ttl=255 (reply in 100)
98	43.2844860	10.0.0.1	10.0.0.2	ICMP	118	Echo (ping) reply id=0x0001, seq=0/0, ttl=255 (request in 91)
99	43.2946380	10.0.0.1	10.0.0.2	ICMP	118	Echo (ping) reply id=0x0001, seq=1/256, ttl=255 (request in 92)
100	43.3048310	10.0.0.1	10.0.0.2	ICMP	118	Echo (ping) reply id=0x0001, seq=2/512, ttl=255 (request in 97)
101	43.3102870	10.0.0.2	10.0.0.1	ICMP	118	Echo (ping) request id=0x0001, seq=3/768, ttl=255 (reply in 102)
102	43.3250270	10.0.0.1	10.0.0.2	ICMP	118	Echo (ping) reply id=0x0001, seq=3/768, ttl=255 (request in 101)
103	43.3304900	10.0.0.2	10.0.0.1	ICMP	118	Echo (ping) request id=0x0001, seq=4/1024, ttl=255 (reply in 104)
104	43.3351760	10.0.0.1	10.0.0.2	ICMP	118	Echo (ping) reply id=0x0001, seq=4/1024, ttl=255 (request in 103)

Frame 91: 118 bytes on wire (944 bits), 118 bytes captured (944 bits) on interface 0
Ethernet II, Src: ca:04:19:6c:00:00 (ca:04:19:6c:00:00), Dst: c0:02:1b:14:00:00 (c0:02:1b:14:00:00)
802.1Q Virtual LAN, PRI: 0, CFI: 0, ID: 10
Internet Protocol Version 4, Src: 10.0.0.2 (10.0.0.2), Dst: 10.0.0.1 (10.0.0.1)
Internet Control Message Protocol
Type: 8 (Echo (ping) request)
Code: 0
Checksum: 0x71eb [correct]
Identifier (BE): 1 (0x0001)
Identifier (LE): 256 (0x0100)
Sequence number (BE): 0 (0x0000)
Sequence number (LE): 0 (0x0000)
[Response frame: 98]
Data (72 bytes)
Data: 00000000000a0c54abcdabcbcdabcbcdabcbcdabcbcd...
[Length: 72]

Figura 2: Captura *wireshark* na interface f0/0 de groucho.

The image shows a Wireshark capture on interface f1/0. The filter is set to 'icmp'. The packet list shows several ICMP Echo (ping) requests and replies. The packet details pane for packet 44 is expanded, showing the Ethernet II, Internet Protocol Version 4, and Internet Control Message Protocol (Type 8 Echo (ping) request) fields. The data field is highlighted in red.

No.	Time	Source	Destination	Protocol	Length	Info
44	64.2285840	10.0.0.2	10.0.0.1	ICMP	114	Echo (ping) request id=0x0001, seq=0/0, ttl=255 (reply in 49)
45	64.2286800	10.0.0.2	10.0.0.1	ICMP	114	Echo (ping) request id=0x0001, seq=1/256, ttl=255 (reply in 50)
48	65.2094410	10.0.0.2	10.0.0.1	ICMP	114	Echo (ping) request id=0x0001, seq=2/512, ttl=255 (reply in 51)
49	65.2456880	10.0.0.1	10.0.0.2	ICMP	114	Echo (ping) reply id=0x0001, seq=0/0, ttl=255 (request in 44)
50	65.2559490	10.0.0.1	10.0.0.2	ICMP	114	Echo (ping) reply id=0x0001, seq=1/256, ttl=255 (request in 45)
51	65.2660580	10.0.0.1	10.0.0.2	ICMP	114	Echo (ping) reply id=0x0001, seq=2/512, ttl=255 (request in 48)
52	65.2712660	10.0.0.2	10.0.0.1	ICMP	114	Echo (ping) request id=0x0001, seq=3/768, ttl=255 (reply in 53)
53	65.2862460	10.0.0.1	10.0.0.2	ICMP	114	Echo (ping) reply id=0x0001, seq=3/768, ttl=255 (request in 52)
54	65.2914490	10.0.0.2	10.0.0.1	ICMP	114	Echo (ping) request id=0x0001, seq=4/1024, ttl=255 (reply in 55)
55	65.2963930	10.0.0.1	10.0.0.2	ICMP	114	Echo (ping) reply id=0x0001, seq=4/1024, ttl=255 (request in 54)

Frame 44: 114 bytes on wire (912 bits), 114 bytes captured (912 bits) on interface 0
Ethernet II, Src: ca:04:19:6c:00:00 (ca:04:19:6c:00:00), Dst: c0:02:1b:14:00:00 (c0:02:1b:14:00:00)
Internet Protocol Version 4, Src: 10.0.0.2 (10.0.0.2), Dst: 10.0.0.1 (10.0.0.1)
Internet Control Message Protocol
Type: 8 (Echo (ping) request)
Code: 0
Checksum: 0x71eb [correct]
Identifier (BE): 1 (0x0001)
Identifier (LE): 256 (0x0100)
Sequence number (BE): 0 (0x0000)
Sequence number (LE): 0 (0x0000)
[Response frame: 49]
Data (72 bytes)
Data: 00000000000a0c54abcdabcbcdabcbcdabcbcdabcbcd...
[Length: 72]

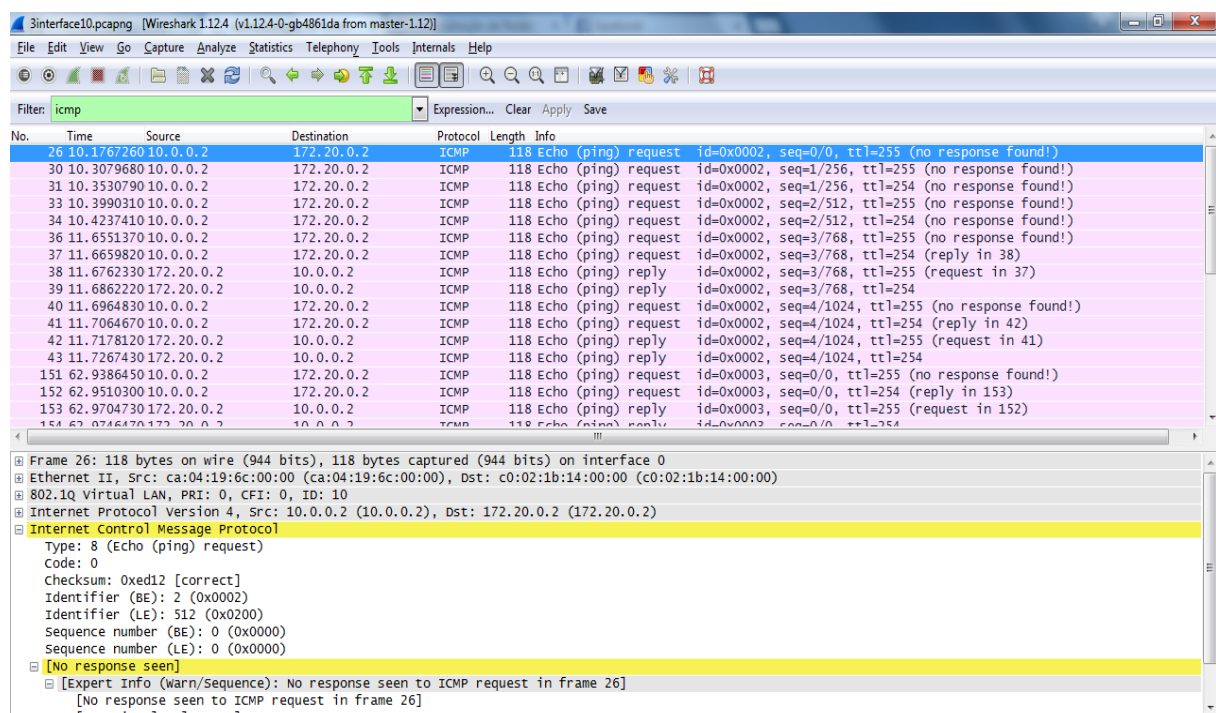
Figura 3: Captura *wireshark* na interface f1/0 de R1.

- b. A diferença dos pacotes que passam entre R1-SW1 e SW1-groucho consiste na existência do campo de 4 *bytes* dedicado ao encapsulamento 802.1Q nos pacotes entre R1-SW1, devido à porta entre estes estar configurada no modo *trunk*. Como entre SW1-groucho há uma VLAN dedicada, a trama é descartada.

2. Na interface f1/0 de R1 podemos observar que são enviadas várias mensagens ARP (ARP *Request*) em *Broadcast*, (Who has 10.0.0.2? Tell 10.0.0.4), que significa que o *host* 10.0.0.4 está a tentar descobrir quem é a máquina 10.0.0.2, à qual não se obteve nenhuma resposta.

Isto acontece porque as portas de SW1 estão configuradas em **modo acesso** na VLAN 10 (*groucho*) e na VLAN20 (*averell*), apesar de ambos os IPs (de *groucho* e *averell*) pertencerem à mesma subnet, o *ping* nunca chega à máquina do *groucho* (na captura *wireshark* na interface f0/0 não chega nenhuma mensagem) uma vez que as máquinas pertencem a VLANs diferentes.

3. É possível notar que o *request* ICMP é bem sucedido de *groucho* a *averell* mas não no sentido oposto. O ICMP *reply* não consegue voltar ao destino, mesmo com o *routing* de R1, pois *groucho* não está na VLAN *default* (ao contrário de *averell*).



No.	Time	Source	Destination	Protocol	Length	Info
26	10.1767260	10.0.0.2	172.20.0.2	ICMP	118	Echo (ping) request id=0x0002, seq=0/0, ttl=255 (no response found!)
30	10.3079680	10.0.0.2	172.20.0.2	ICMP	118	Echo (ping) request id=0x0002, seq=1/256, ttl=255 (no response found!)
31	10.3530790	10.0.0.2	172.20.0.2	ICMP	118	Echo (ping) request id=0x0002, seq=1/256, ttl=254 (no response found!)
33	10.3990310	10.0.0.2	172.20.0.2	ICMP	118	Echo (ping) request id=0x0002, seq=2/512, ttl=255 (no response found!)
34	10.4237410	10.0.0.2	172.20.0.2	ICMP	118	Echo (ping) request id=0x0002, seq=2/512, ttl=254 (no response found!)
36	11.6551370	10.0.0.2	172.20.0.2	ICMP	118	Echo (ping) request id=0x0002, seq=3/768, ttl=255 (no response found!)
37	11.6659820	10.0.0.2	172.20.0.2	ICMP	118	Echo (ping) request id=0x0002, seq=3/768, ttl=254 (reply in 38)
38	11.6762330	172.20.0.2	10.0.0.2	ICMP	118	Echo (ping) reply id=0x0002, seq=3/768, ttl=255 (request in 37)
39	11.6862220	172.20.0.2	10.0.0.2	ICMP	118	Echo (ping) reply id=0x0002, seq=3/768, ttl=254
40	11.6964830	10.0.0.2	172.20.0.2	ICMP	118	Echo (ping) request id=0x0002, seq=4/1024, ttl=255 (no response found!)
41	11.7064670	10.0.0.2	172.20.0.2	ICMP	118	Echo (ping) request id=0x0002, seq=4/1024, ttl=254 (reply in 42)
42	11.7178120	172.20.0.2	10.0.0.2	ICMP	118	Echo (ping) reply id=0x0002, seq=4/1024, ttl=255 (request in 41)
43	11.7267430	172.20.0.2	10.0.0.2	ICMP	118	Echo (ping) reply id=0x0002, seq=4/1024, ttl=254
151	62.9386450	10.0.0.2	172.20.0.2	ICMP	118	Echo (ping) request id=0x0003, seq=0/0, ttl=255 (no response found!)
152	62.9510300	10.0.0.2	172.20.0.2	ICMP	118	Echo (ping) request id=0x0003, seq=0/0, ttl=254 (reply in 153)
153	62.9704730	172.20.0.2	10.0.0.2	ICMP	118	Echo (ping) reply id=0x0003, seq=0/0, ttl=255 (request in 152)
154	62.9746470	172.20.0.2	10.0.0.2	ICMP	118	Echo (ping) reply id=0x0003, seq=0/0, ttl=254

Figura 4: Captura *wireshark* na interface f1/0 de R1.

4.

a. Para que a ligação entre R1 e o terminal linux funcione em modo *trunk* tivemos de realizar as seguintes configurações.

No *router* R1:

```
interface FastEthernet 1/3
switchport trunk encapsulation dot1q
switchport mode trunk
```

No terminal linux:

```
[root@Labs5610 ar]# modprobe 8021q
[root@Labs5610 ar]# vconfig add tap0 10
Added VLAN with VID == 10 to IF -:tap0:-
[root@Labs5610 ar]#
[root@Labs5610 ar]# ifconfig tap0.10 10.0.0.5 netmask 255.255.255.0 up
```

b. Captura de um pacote do ping (ICMP) enviado em ambas as interfaces:

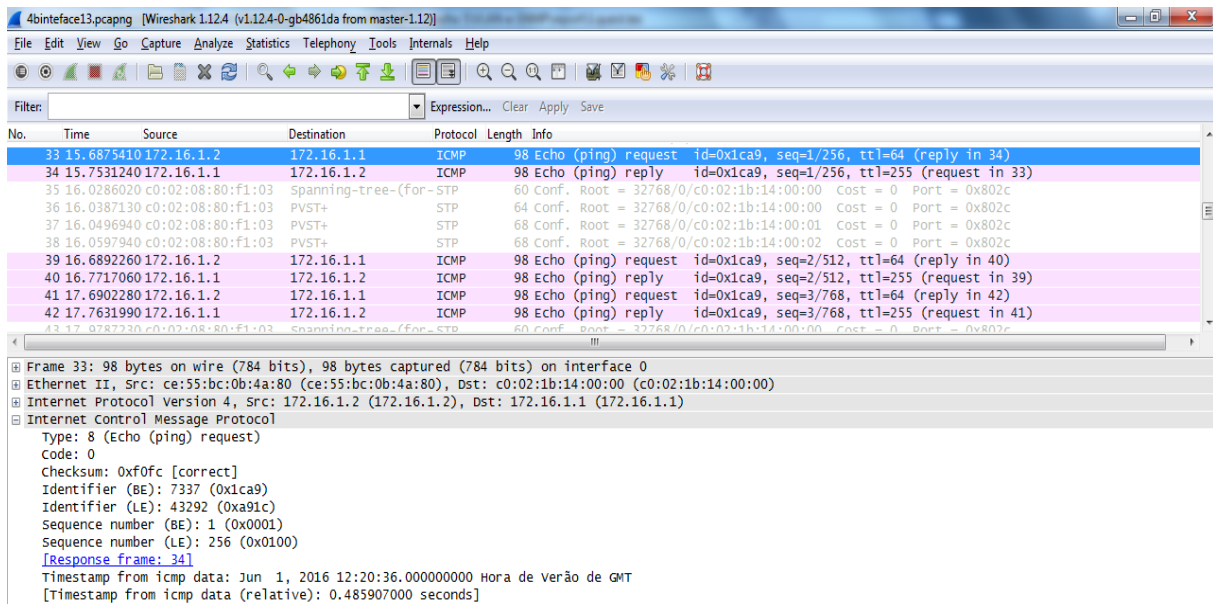


Figura 5: Captura *wireshark* de um pacote do ping enviado do terminal linux para a VLAN 1.

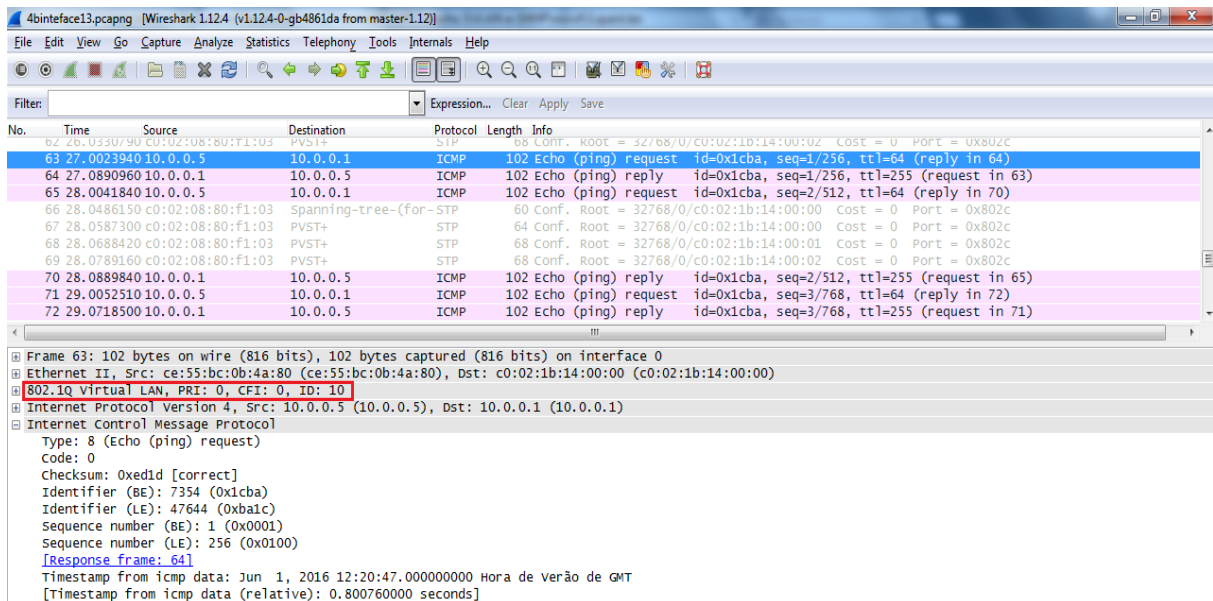


Figura 6: Captura *wireshark* de um pacote do ping enviado do terminal linux para a VLAN 10.

c. Como a conexão R1-linux é feita em modo *trunk* com uma interface para a VLAN 1 e VLAN 10, ao fazer ping para um endereço da VLAN 10 este recebe a trama de 802.1Q identificando o pacote que passa pela conexão em modo *trunk* como pertencente à VLAN 10. O ping para uma interface na VLAN 1, como é a VLAN *default*, carece do campo de 802.1Q.

5.

a.

```
[root@Labs5610 ar]# snmpget -v 2c -c Leitura 172.16.1.1
iso.org.dod.internet.mgmt.mib-2.system.sysDescr
SNMPv2-MIB::sysDescr = No Such Instance currently exists at this OID
```

Não se conseguiu realizar o **get** porque foi solicitada uma classe e não uma instância.

Quando queremos usar um OID precisamos de adicionar um outro número para obter o valor dessa variável. Por isso, precisamos de acrescentar um ".0" que representa a primeira instância (e único, uma vez que um dispositivo não pode ter mais de uma descrição) desse objeto.

b.

```
[root@Labs5610 ar]# snmpget -v 2c -c Leitura 172.16.1.1
iso.org.dod.internet.mgmt.mib-2.system.sysDescr.0
SNMPv2-MIB::sysDescr.0 = STRING: Cisco IOS Software, 2600 Software
(C2691-ADVIPSERVICESK9-M), Version 12.4(15)T6, RELEASE SOFTWARE (fc2)
Technical Support: http://www.cisco.com/techsupport
Copyright (c) 1986-2008 by Cisco Systems, Inc.
Compiled Mon 07-Jul-08 04:30 by prod_rel_team
```

c. O **getnext** serve para retornar a instância da classe OID, na árvore de MIB de dados, a seguir à instância descrita no comando. Ou seja, retorna o valor das instâncias de variáveis cujos OID são imediatamente sucessoras lexicográfica aos OID indicados.

```
[root@Labs5610 ar]# snmpgetnext -v 2c -c Leitura 172.16.1.1
iso.org.dod.internet.mgmt.mib-2.system.sysDescr.0
SNMPv2-MIB::sysObjectID.0 = OID: SNMPv2-SMI::enterprises.9.1.122
```

d. O **walk** percorre todas as instâncias de todas as classes respeitando a hierarquia. É construído de acordo com as respostas que vai recebendo do **get-next-request** (ver na captura abaixo) para recolher a informação a seguir. Ou seja, o objetivo é recuperar uma sub-árvore de valores usando repetidas solicitações SNMP GetNext.

Se é dado um OID, este especifica qual parte do espaço serão pesquisadas usando as solicitações **getnext**. Todas as variáveis na sub-árvore abaixo desse OID serão consultadas e os seus valores mostrados ao utilizador.

Se não é dado nenhum OID, o **walk** irá procurar a sub-árvore a partir de SNMPv2-SMI :: mib-2.

O **walk** para quando retorna resultados que já não estão dentro do alcance do OID.

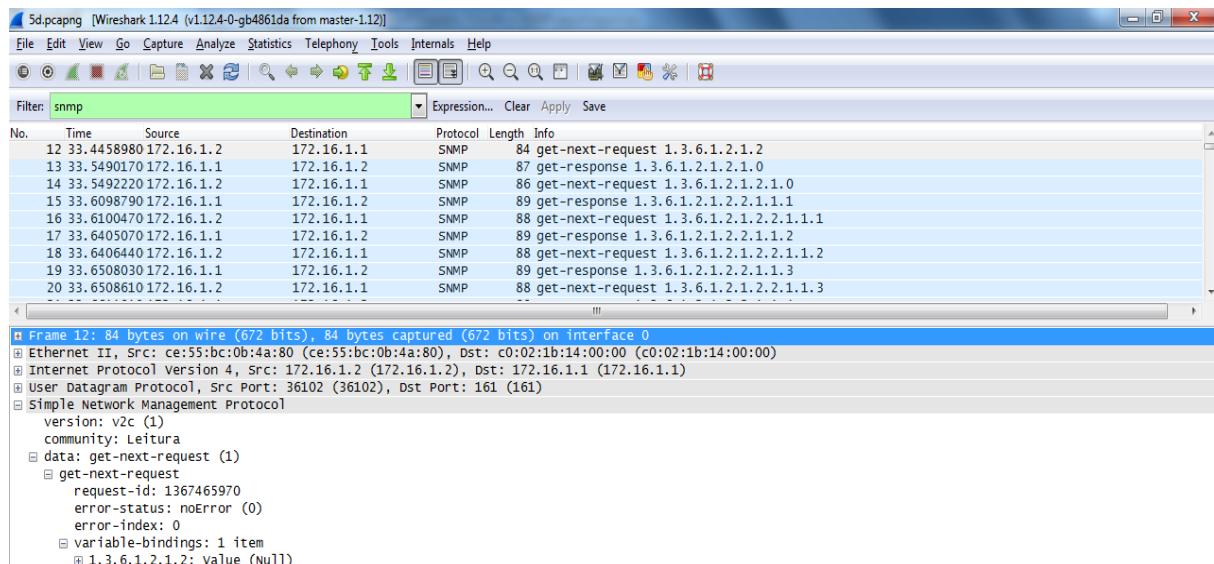


Figura 7: Captura *wireshark* de pacotes SNMP na interface f1/3 de R1.

e. O *router* tem 22 interfaces, que são as que se seguem (parte da resposta do *snmpwalk* no terminal LINUX):

```
IF-MIB::ifNumber.0 = INTEGER: 22
...
IF-MIB::ifDescr.1 = STRING: FastEthernet0/0
IF-MIB::ifDescr.2 = STRING: FastEthernet0/1
IF-MIB::ifDescr.3 = STRING: FastEthernet1/0
IF-MIB::ifDescr.4 = STRING: FastEthernet1/1
IF-MIB::ifDescr.5 = STRING: FastEthernet1/2
IF-MIB::ifDescr.6 = STRING: FastEthernet1/3
IF-MIB::ifDescr.7 = STRING: FastEthernet1/4
IF-MIB::ifDescr.8 = STRING: FastEthernet1/5
IF-MIB::ifDescr.9 = STRING: FastEthernet1/6
IF-MIB::ifDescr.10 = STRING: FastEthernet1/7
IF-MIB::ifDescr.11 = STRING: FastEthernet1/8
IF-MIB::ifDescr.12 = STRING: FastEthernet1/9
IF-MIB::ifDescr.13 = STRING: FastEthernet1/10
IF-MIB::ifDescr.14 = STRING: FastEthernet1/11
IF-MIB::ifDescr.15 = STRING: FastEthernet1/12
IF-MIB::ifDescr.16 = STRING: FastEthernet1/13
IF-MIB::ifDescr.17 = STRING: FastEthernet1/14
IF-MIB::ifDescr.18 = STRING: FastEthernet1/15
IF-MIB::ifDescr.20 = STRING: Null0
IF-MIB::ifDescr.21 = STRING: Vlan1
IF-MIB::ifDescr.22 = STRING: Vlan10
IF-MIB::ifDescr.23 = STRING: Vlan20
...
```

f. Resultado do get:

```
[root@Labs5610 ar]# snmpget -v 2c -c Leitura 172.16.1.1  
iso.org.dod.internet.mgmt.mib-2.system.sysName.0  
SNMPv2-MIB::sysName.0 = STRING: R1
```

Comando usado para fazer o set e respetivo resultado:

```
[root@Labs5610 ar]# snmpset -v 2c -c Escrita 172.16.1.1  
iso.org.dod.internet.mgmt.mib-2.system.sysName.0 s R2  
SNMPv2-MIB::sysName.0 = STRING: R2
```

g. Faça um walk e um bulkwalk à sub-árvore system da MIB-2. Qual é a diferença entre estes dois comandos e em que se traduz na prática????? ($2 \times \text{capRes} + \text{texRes}$)

O **bulkwalk** obtém a informação com apenas um pedido, uma vez que usa solicitações SNMP GetBulk (**get-bulk-request** que permite a transferência de grandes volumes de informação). Enquanto que o **walk** usa solicitações SNMP GetNext, que realiza um pedido para cada variável, como já foi explicado acima.

A vantagem do **bulkwalk** em relação ao **walk** é o ganho em termos de eficiência.

Nas capturas a seguir podemos reparar na diferença de quantidade de pedidos feitos entre os 2, através da barra lateral.

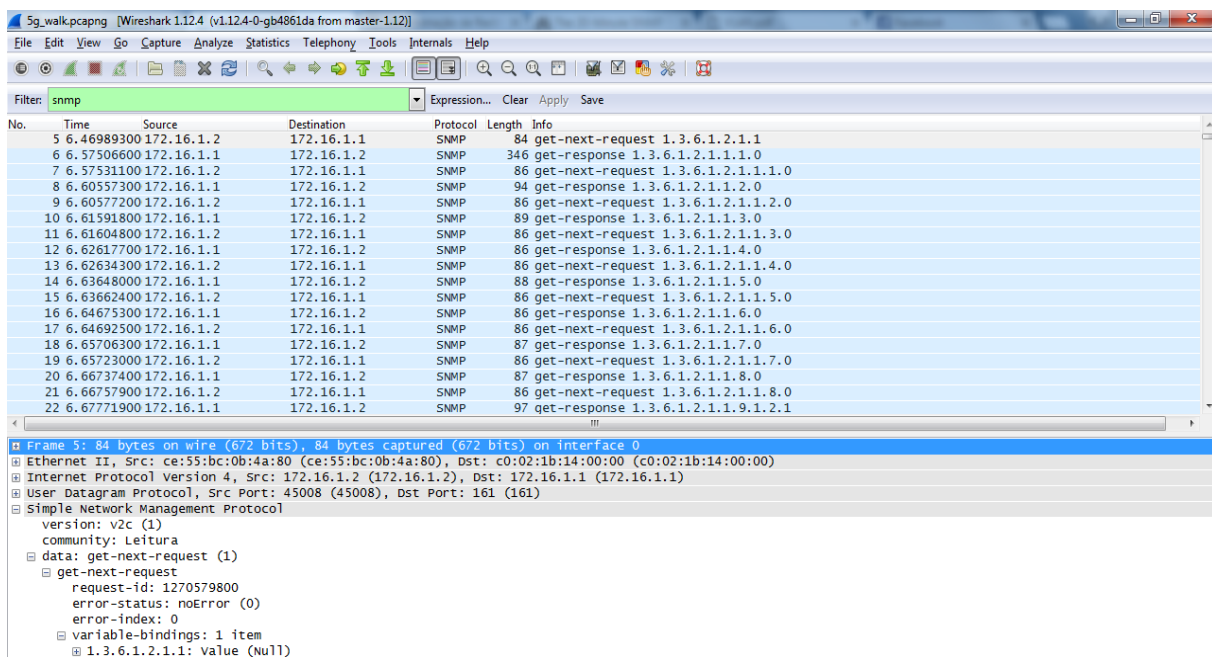


Figura 8: Captura *wireshark* de pacotes **snmpwalk**.

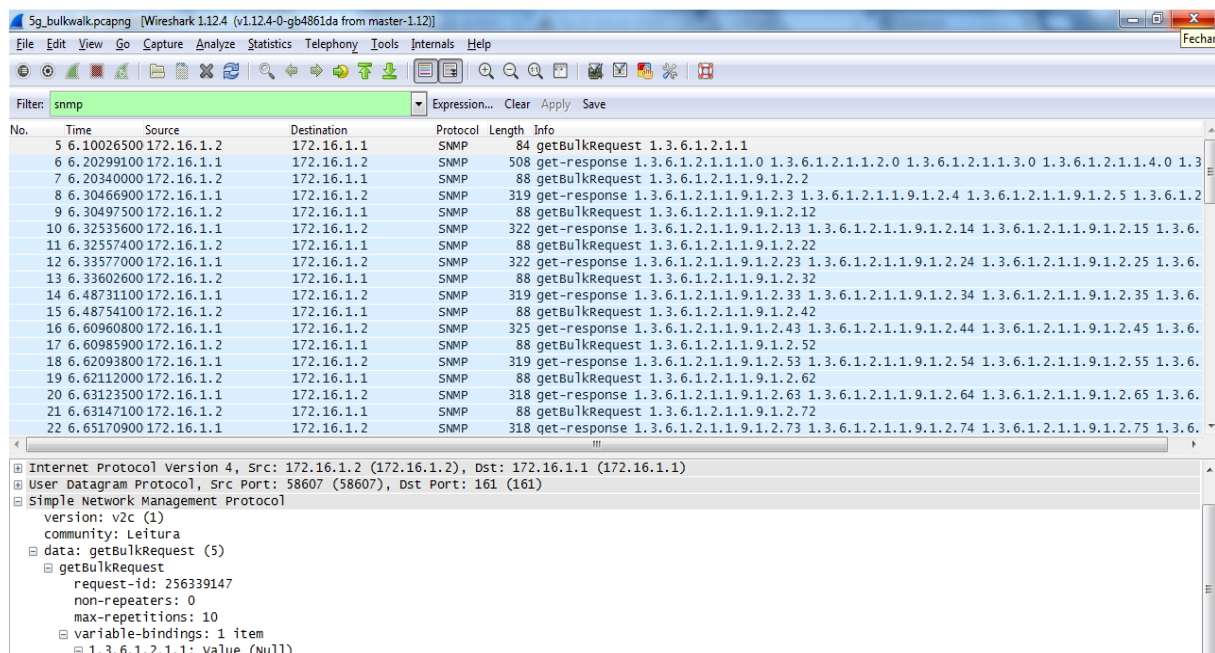


Figura 9: Captura *wireshark* de pacotes bulkwalk.

h. Informação sobre as interfaces de R1:

```
[root@Labs5610 ar]# snmpnetstat -v 2c -Ci -c Leitura 172.16.1.1
```

Name	Mtu	Network	Address	Ipkts	Ierrs	Opkts	Oerrs	Queue
Fa0/0	1500			0	0	0	0	0
Fa0/1	1500			0	0	0	0	0
Fa1/0	1500			0	0	0	0	0
Fa1/1	1500			0	0	0	0	0
Fa1/2	1500			0	0	0	0	0
Fa1/3	1500			0	0	0	0	0
Fa1/4	1500			0	0	0	0	0
Fa1/5	1500			0	0	0	0	0
Fa1/6	1500			0	0	0	0	0
Fa1/7	1500			0	0	0	0	0
Fa1/8	1500			0	0	0	0	0
Fa1/9	1500			0	0	0	0	0
Fa1/10	1500			0	0	0	0	0
Fa1/11	1500			0	0	0	0	0
Fa1/12	1500			0	0	0	0	0
Fa1/13	1500			0	0	0	0	0
Fa1/14	1500			0	0	0	0	0
Fa1/15	1500			0	0	0	0	0
Nu0	1500			0	0	0	0	0
Vl1	1500	172.16.1/24	172.16.1.1	1178	0	1183	0	0
Vl10	1500	10.0.0/24	10.0.0.1	61	0	39	0	0
Vl20	1500	172.20.0/24	172.20.0.1	47	0	29	0	0

Informação sobre a tabela de encaminhamento de R1:

```
[root@Labs5610 ar]# snmpnetstat -v 2c -Cr -c Leitura 172.16.1.1
Routing tables (ipCidrRouteTable)
Destination          Gateway             Flags    Interface
10.0.0/24             *                  <U>      V110
172.16.1/24          *                  <U>      V11
172.20.0/24          *                  <U>      V120
```

6.

a. Tabela de informação sobre os endereços IP de R1:

```
[root@Labs5610 ar]# snmptable -v 2c -c Leitura 172.16.1.1 ip.ipAddrTable
SNMP table: IP-MIB::ipAddrTable
```

ipAdEntAddr	ipAdEntIfIndex	ipAdEntNetMask	ipAdEntBcastAddr	ipAdEntReasmMaxSize
10.0.0.1	22	255.255.255.0	1	18024
172.16.1.1	21	255.255.255.0	1	18024
172.20.0.1	23	255.255.255.0	1	18024

b. Com base no RFC 2011, é possível observarmos uma definição parcial do módulo IP-MIB. O que faz com que `ip.ipAddrTable` seja uma tabela, consiste na presença da característica opcional "INDEX `ipAdEntAddr`" existente na definição do tipo de cada objeto que `ipAddrTable` tem em sequência. Sendo assim, obtemos uma tabela indexada a `IpAddrEntry.ipAdEntAddr`.

`ipAddrTable` OBJECT-TYPE

SYNTAX SEQUENCE OF `IpAddrEntry`

MAX-ACCESS not-accessible

STATUS current

DESCRIPTION

"The table of addressing information relevant to this entity's IP addresses."

::= { ip 20 }

`ipAddrEntry` OBJECT-TYPE

SYNTAX `IpAddrEntry`

MAX-ACCESS not-accessible

STATUS current

DESCRIPTION

"The addressing information for one of this entity's IP addresses."

INDEX { `ipAdEntAddr` }

::= { `ipAddrTable` 1 }

`IpAddrEntry` ::= SEQUENCE {

`ipAdEntAddr` IpAddress,

`ipAdEntIfIndex` INTEGER,

`ipAdEntNetMask` IpAddress,

`ipAdEntBcastAddr` INTEGER,

`ipAdEntReasmMaxSize` INTEGER

}

c. Para mapear tabelas na estrutura em árvore é necessário definir parte da MIB como tabular e mapear a linha e a coluna da tabela no OID que contém a respetiva entrada.

É de notar que, a coluna pode ser sempre identificada por um número (número da coluna), a linha é identificada pelo índice, e que no mapeamento para o OID, aparece primeiro a coluna e depois a linha.

7.

a. Na captura podemos observar que é enviado um *trap* a informar de uma *generic-trap: linkDown*, relativa à falha da ligação da interface f1/2.

No terminal linux podemos ver a receção desse *trap*:

```
[root@Labs5610 ar]# snmptrapd -n -f -Lo
NET-SNMP version 5.7.3
2016-06-02 10:38:04 172.16.1.1(via UDP: [172.16.1.1]:62413->[172.16.1.2]:162)
TRAP, SNMP v1, community traps7
SNMPv2-MIB::snmpTraps Link Down Trap (0) Uptime: 0:27:01.63
IF-MIB::ifIndex.5 = INTEGER: 5 IF-MIB::ifDescr.5 = STRING: FastEthernet1/2
IF-MIB::ifType.5 = INTEGER: ethernetCsmacd(6)
SNMPv2-SMI::enterprises.9.2.2.1.1.20.5 = STRING: "administratively down"
```

Onde podemos ver que é informado que a ligação da interface f1/2 (IF-MIB::ifDescr.5 = STRING: FastEthernet1/2) é desativada administrativamente (a string SNMPv2-SMI::enterprises.9.2.2.1.1.20.5 foi alterada para "administratively down").

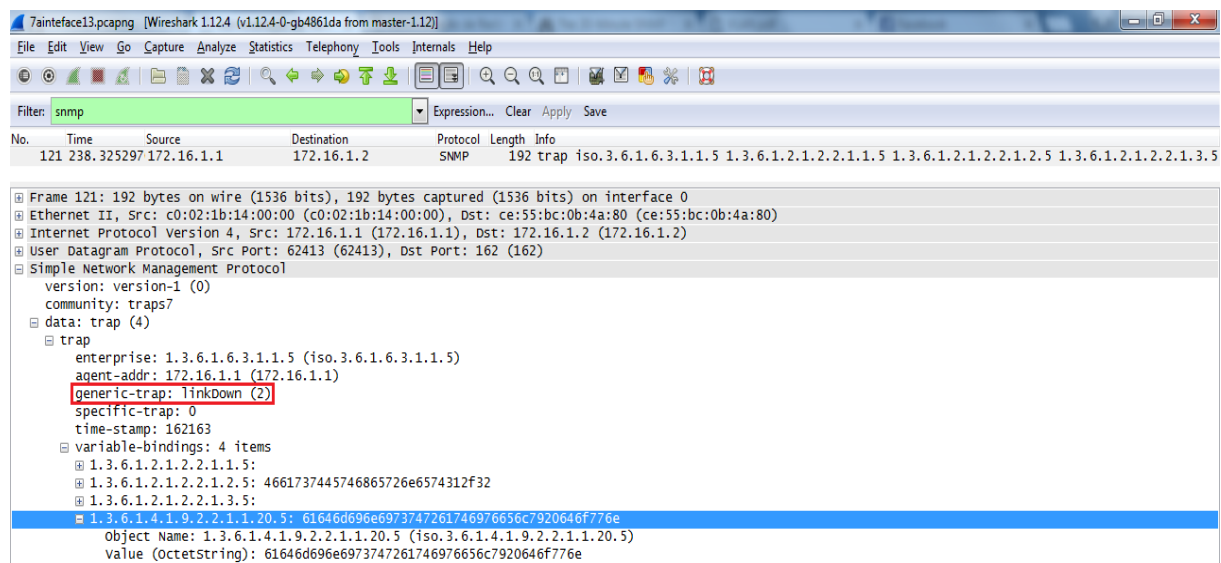


Figura 10: Captura *wireshark* na interface 1/3 de R1.

b. Captura de pacotes SNMP na interface f1/3 de R1:

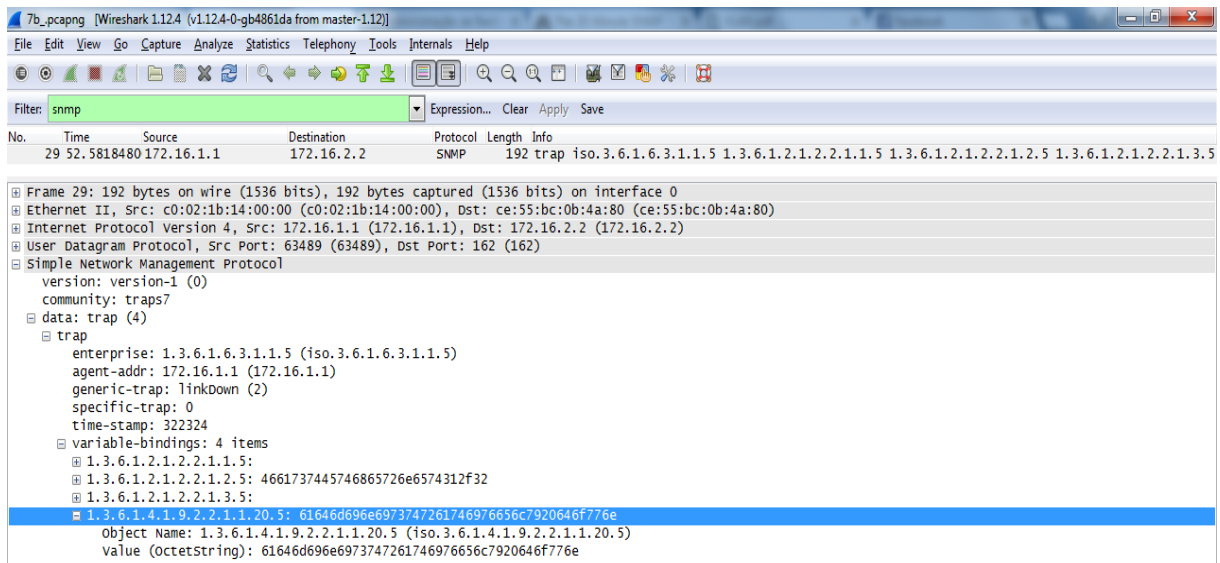


Figura 11: Captura *wireshark* na interface 1/3 de R1.

c. Captura de pacotes SNMP na interface f1/3 de R1:

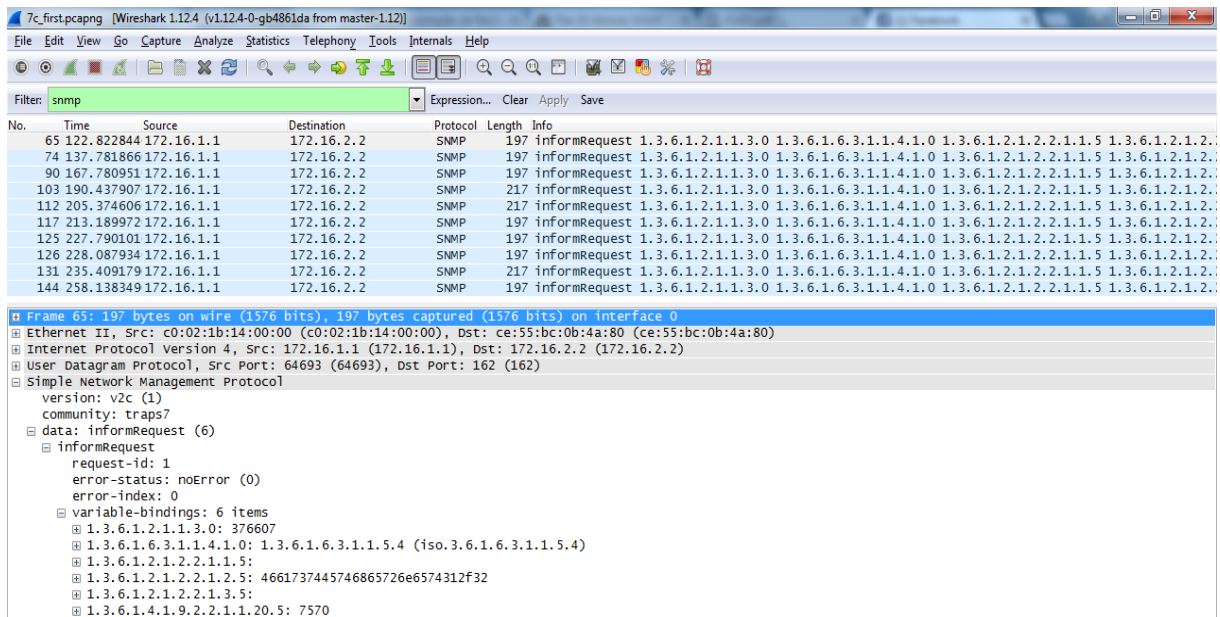


Figura 12: Captura *wireshark* na interface 1/3 de R1.

d. Captura de pacotes SNMP na interface f1/3 de R1:

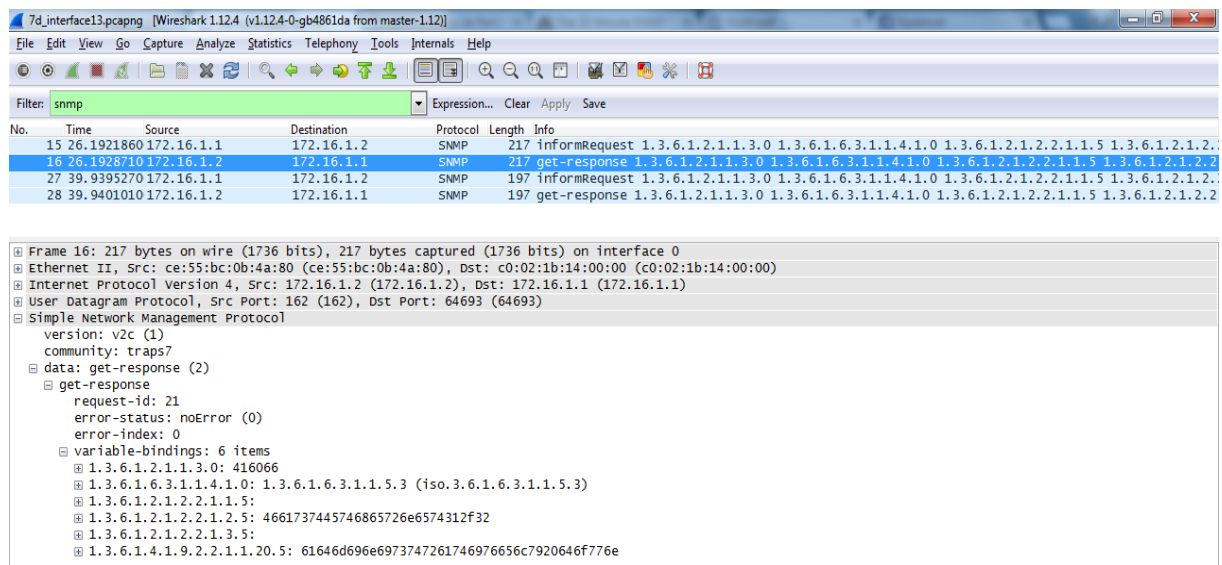


Figura 13: Captura *wireshark* na interface 1/3 de R1.

e. *traps* servem para notificar, de forma assíncrona, o gestor da ocorrência de um dado evento, aqui as mensagens não são confirmadas, o que implica uma entrega não-fiável. Em reação à receção de uma *trap*, o gestor pode fazer pedidos para obter mais variáveis que lhe dêem informação adicional.

informs são semelhantes à *trap*, mas com receção confirmada. São concebidos para comunicação entre gestores, mas pode também ser enviados por um agente a um gestor.

8.

a. Captura dos pacotes trocados quando executados os seguintes comandos:

- `snmpgetnext -v2c -c Leitura 172.16.1.1 system.sysName:`

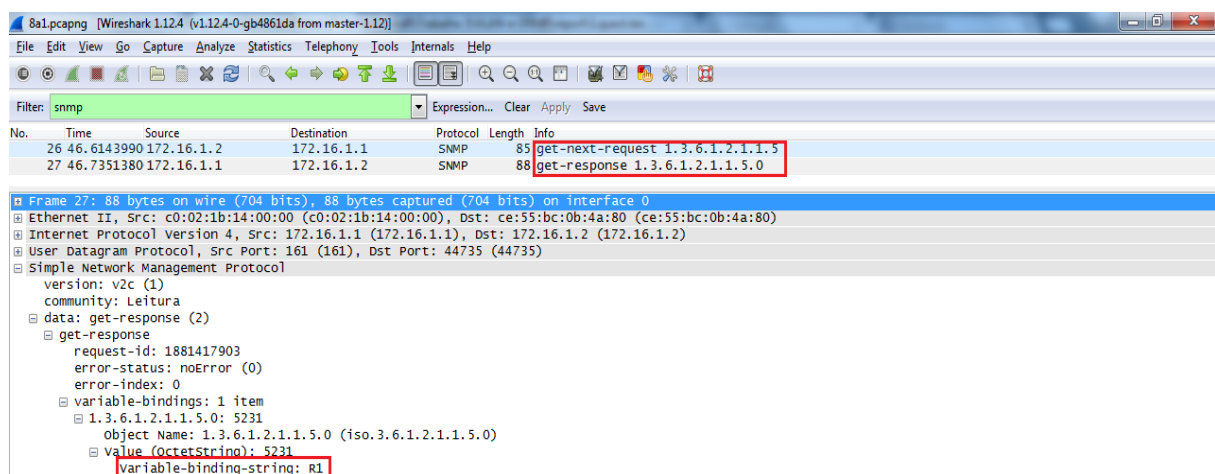


Figura 14: Captura *wireshark* na interface 1/3 de R1.

- `snmpgetnext -v3 -l authNoPriv -u grupo11 -a md5 -A passlimi 172.16.1.1 system.sysName:`

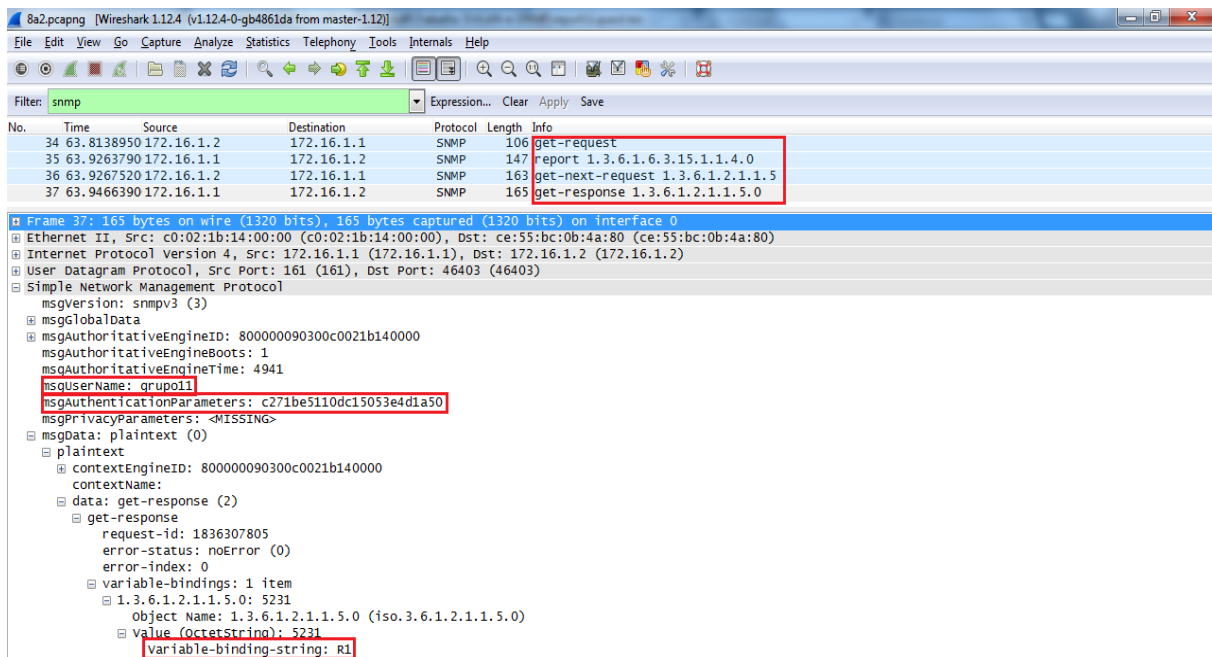


Figura 15: Captura *wireshark* na interface 1/3 de R1.

- `snmpgetnext -v3 -l authPriv -u cifragrupo11 -a md5 -A passlimi -x des -X cifralimi 172.16.1.1 system.sysName:`

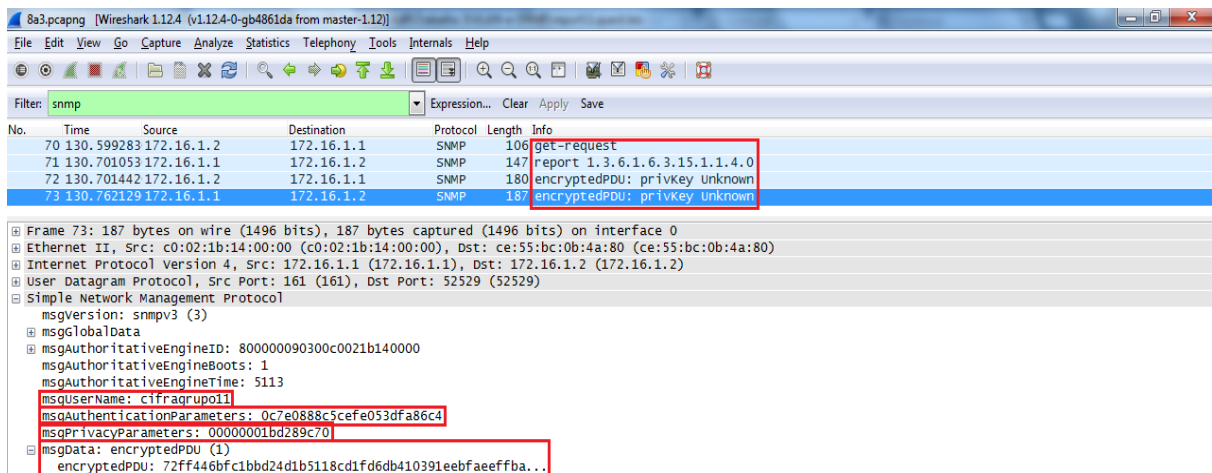


Figura 16: Captura *wireshark* na interface 1/3 de R1.

b. A segurança em SNMPv2 é baseada em autenticação por *strings* de comunicação, uma espécie de *password* partilhada que circula em *clear text*, sendo este o caso menos seguro.

O SNMPv3 suporta autenticação segura, recorrendo a *hashes* criptográficos, também permite definir diferentes níveis de acesso a diferentes partes da MIB para diferentes utilizadores, e suporta garantia de integridade das mensagens. Este possui 3 níveis de segurança, `noAuthNoPriv`, `AuthNoPriv` e `AuthPriv`, sendo os dois últimos experimentados neste trabalho.

Podemos afirmar que o nível de segurança `AuthPriv` é mais seguro de que o `AuthNoPriv`, uma vez que ele suporta cifragem das mensagens de modo a garantir privacidade, ao contrário do nível de segurança `AuthNoPriv`.