Aplicação de download e configuração de rede

RELATÓRIO
REDES DE COMPUTADORES

Francisco Teixeira Lopes
ei11056
Nelson André Garrido da Costa
up201403128

Sumário

Este relatório incide sobre a criação de uma aplicação de *download* e configuração de uma rede.

A aplicação utiliza o protocolo FTP com ligações TCP para efetuar a transferência, implementando uma versão leve do RFC959.

A rede consiste numa série de experiências que culminaram em configurar um *router* comercial com NAT, um *switch* com duas LAN virtuais, e cada computador da rede com respetivo endereço IP e DNS.

O objetivo final, foi efetuar uma transferência usando a aplicação na rede configurada.

Índice

Introdução **1**

Parte 1 – Aplicação de *download* **2**

Arquitetura da aplicação **2**

Exemplo de *download* **3**

Parte 2 – Configuração de rede **4**

Experiência 1 – Configurar uma rede IP **4**

Experiência 2 – Implementar duas LAN virtuais num *switch* **5**

Experiência 3 – Configurar um *router* em Linux **5**

Experiência 4 – Configurar um *router* comercial e implementar NAT **6**

Experiência 5 – DNS **6**

Experiência 6 – Ligações TCP **7**

Conclusão **9**

Anexos **10**

Anexo 1 – Comandos de configuração **10**

Experiência 1 10

Experiência 2 10

Experiência 3 11

Experiência 4 (sem NAT) 11

Experiência 4 (NAT) 12

Experiência 5 12

Anexo 2 – Capturas **13**

Experiência 1 13

Experiência 2 13

Experiência 3 15

Experiência 4 15

Experiência 5 16

Experiência 6 16

Anexo 3 – Código da aplicação de download **17**

Download.h 17

Download.c 20

Makefile 32

Introdução

O segundo trabalho laboratorial de Redes de Computadores divide-se em duas partes: criação de uma aplicação de *download* e configuração de uma rede.

A primeira parte, tem por objetivo efetuar a transferência de um ficheiro a partir de um servidor, para isto, é utilizado o protocolo FTP com implementação do RFC959 e conexões TCP ao servidor.

A segunda parte, tem por objetivo configurar uma rede de forma a poder efetuar a transferência de um ficheiro. Foram efetuadas várias experiências incrementais, que eventualmente, permitiram acesso à Internet a partir da rede configurada.

Este relatório encontra-se dividido em duas seções principais:

- Aplicação de *download*: nesta seção descreve-se a arquitetura da aplicação e apresenta-se um relato de uma transferência feita com sucesso.
- Configuração de rede: nesta seção apresentam-se os passos para configuração da rede e reflete-se sobre os objetivos de aprendizagem de cada experiência.

Parte 1 – Aplicação de *download*

Arquitetura da aplicação

O processamento da aplicação acontece na função main do programa, esta chama, sequencialmente, todas as funções auxiliares necessárias para descodificar o argumento passado pelo utilizador, a obtenção dos dados necessários para poder fazer a ligação TCP ao servidor FTP, e ainda, as operações necessárias para conseguir fazer o *download* do ficheiro pedido.

Para este efeito são usadas duas *structs* com dados utéis ao programa, a primeira *struct*, FTPFile_t, contém o caminho para onde guardar o ficheiro localmente, e também, o descritor de ficheiro do *socket* de dados.

A segunda *struct*, FTPArgument_t, contém os comandos FTP necessários para obter o ficheiro, bem como, um valor booleano que determina se o modo é *anonymous*.

```
typedef struct {

  char path[FILEPATH_SIZE];
  int datafd;
} FTPFile_t;
```

```
typedef struct {

  char user[BUFFER_SIZE];
  char password[BUFFER_SIZE];
  char host[BUFFER_SIZE];
  char path[FILEPATH_SIZE];
  uint8_t anonymous_f;
} FTPArgument_t;
```

Figura 1 - estruturas de dados da aplicação

A aplicação é executada recorrendo a um argumento, que indica o nome de utilizador e a palavra-passe, juntamente com, o URL do servidor e ficheiro a transferir. Pode-se também omitir o utilizador e palavra-passe, sendo que, é assumido o modo *anonymous*.

```
download: usage: ./download ftp://[<user>:<password>@]<host>/<url-path>
ubuntu@ubuntu-VirtualBox:/media/sf_MintShare$
```

Figura 2 - utilização da aplicação

Após executar a aplicação, é chamada uma sequência de funções com a finalidade de transferir o ficheiro pedido. Primeiramente, é validado o argumento fornecido, para esse efeito, a função "*parseArgument*" verifica que todos os dados necessários estão presentes. A *struct* FTPArgument_t é preenchida por esta função, no caso de argumento válido. De seguida, é obtido o endereço IP do servidor através da função "*getAddress*", o qual é fornecido à função "*getTCPSocket*". Esta, cria uma ligação TCP à porta 21 do servidor, isto é, a porta de controlo de FTP. Após estabelecer ligação ao servidor, são enviados os comandos necessários para transferir o ficheiro, a função "*FTPLogin*" envia os comandos "USER" e "PASS", de seguida "*FTPCommand*" envia "TYPE I" para usar modo binário e finalmente "*FTPPassive*" envia o comando "PASV" e retorna a porta de dados. Sabendo a porta de dados, "*getTCPSocket*" cria outra ligação TCP com o mesmo endereço IP mas porta distinta. Criada esta ligação, "*FTPCommand*" envia "RETR" e inicia a transferência do ficheiro, isto é feito através de um valor booleano na função, que indica a possibilidade do comando ativar uma transferência. Termina, assim, a aplicação com o ficheiro transferido na pasta onde foi executada, qualquer erro é devidamente mostrado ao utilizador e, tratando-se da seção FTP, é tratado de acordo com o RFC959.

Exemplo de *download*

Para fazer transferência de um ficheiro, basta fornecer à aplicação um URL correto e, opcionalmente, o utilizador e palavra-passe. Seguem-se imagens de testes bem sucedidos, um em modo anónimo, e o outro com dados de utilizador.

Os comandos enviados foram:

- ./download ftp://demo:password@test.rebex.net/pub/example/mail-editor.png
- ./download ftp://speedtest.tele2.net/20MB.zip

```
ubuntu@ubuntu-VirtualBox:/media/sf_MintShare$ ./download ftp://demo:password@tes
t.rebex.net/pub/example/mail-editor.png
Host Name  : test.rebex.net
IP Address : 195.144.107.198

220 Microsoft FTP Service
331 Password required for demo.
230 User logged in.
200 Type set to I.
227 Entering Passive Mode (195,144,107,198,4,5)
125 Data connection already open; Transfer starting.
226 Transfer complete.
```

Figura 3 - download com utilizador

```
ubuntu@ubuntu-VirtualBox:/media/sf_MintShare$ ./download ftp://speedtest.tele2.n
et/20MB.zip
Host Name  : speedtest.tele2.net
IP Address : 90.130.70.73

220 (vsFTPd 2.3.5)
331 Please specify the password.
230 Login successful.
200 Switching to Binary mode.
227 Entering Passive Mode (90,130,70,73,101,162)
150 Opening BINARY mode data connection for /20MB.zip (20971520 bytes).
226 Transfer complete.
```

Figura 4 - download em modo anónimo

Parte 2 – Configuração de rede

Nos seguintes subcapítulos são elucidados os objetivos de cada experiência, sendo que, a configuração exaustiva é apresentada em anexo no final do relatório, assim como, os respetivos *logs* de captura de pacotes.

Experiência 1 – Configurar uma rede IP

Esta experiência teve por objetivo a configuração de uma rede local, na qual, se ligava um computador a outro. Para este efeito, ligaram-se os computadores diretamente através da porta eth0, e de seguida, configuraram-se os seus endereços IP de forma a situarem-se na mesma sub-rede (172.16.10.x).

O que são pacotes ARP e para que servem?

Os pacotes ARP servem para mapear um endereço de rede (IPv4) a um endereço físico (MAC).

Qual é o MAC e endereço IP de pacotes ARP?

Existem pacotes ARP de pedido e de resposta, num pacote de pedido, é enviado o endereço IP do computador que fez o pedido, e também, o endereço IP do computador do qual se quer saber o MAC. Num pacote de resposta, é enviado ao computador que fez o pedido, o endereço IP e MAC do computador destino.

```
1 0.000000    G-ProCom_8b:e4:a7    Broadcast            ARP         42 Who has 172.16.50.254? Tell 172.16.50.1
2 0.000115    HewlettP_c3:78:70    G-ProCom_8b:e4:a7    ARP         60 172.16.50.254 is at 00:21:5a:c3:78:70
```

Figura 5 - exemplo de pacotes ARP

Que pacotes gera o comando *ping*?

O comando *ping* gera pacotes ICMP, estes podem ser do tipo *echo request* ou do tipo *echo reply*.

```
3 0.000136    172.16.50.1      172.16.50.254    ICMP    98 Echo (ping) request  id=0x08d9, seq=1/256, ttl=64 (repl…
4 0.000278    172.16.50.254    172.16.50.1      ICMP    98 Echo (ping) reply    id=0x08d9, seq=1/256, ttl=64 (requ…
5 0.999001    172.16.50.1      172.16.50.254    ICMP    98 Echo (ping) request  id=0x08d9, seq=2/512, ttl=64 (repl…
6 0.999088    172.16.50.254    172.16.50.1      ICMP    98 Echo (ping) reply    id=0x08d9, seq=2/512, ttl=64 (requ…
```

Figura 6 - exemplo de pacotes ICMP

Qual é o MAC e endereço IP de pacotes ping?

Os pacotes ICMP contêm o endereço IP do computador origem e do computador destino, isto é válido tanto para *echo request*, como para *echo reply*.

Como se determina se uma trama Ethernet é do tipo ARP, IP, ICMP?

No cabeçalho da trama Ethernet, no campo *EtherType*, é possível verificar se se trata de um pacote ARP (0x0806) ou IP (0x0800). Para pacotes ICMP (tipo 1), esta informação, vem no cabeçalho do pacote IP no campo *Protocol*.

Como se determina o tamanho de uma trama Ethernet?

O tamanho da trama é indicado no campo *EtherType* da trama, para evitar ambiguidade com o outro uso deste campo, utilizam-se valores até 1500 para referir o tamanho, e valores acima de 1536 para o tipo.

O que é a interface de *loopback* e para que serve?

A interface de *loopback* é utilizada para um computador poder comunicar consigo mesmo, isto tem a utilidade de poder realizar testes de diagnóstico, e também, de poder aceder a servidores no próprio computador.

Experiência 2 – Implementar duas LAN virtuais num *switch*

Esta experiência teve por objetivo a implementação de LAN virtual, dois computadores seriam ligados a uma VLAN e um terceiro a outra VLAN sem rota entre as duas VLAN.

Para este efeito, configurou-se um *switch* da Cisco e respetivas portas para ligar o computador 1 e 4 numa VLAN, e o computador 2 noutra VLAN.

Quantos domínios de *broadcast* existem? Como se pode concluir isso a partir dos *logs*?

Existem dois domínios de *broadcast* na rede configurada, pois a divisão em VLAN cria duas sub-redes. Nos *logs* pode-se ver isto pois ao fazer *ping broadcast* a partir do computador 1, chega-se apenas ao computador 4, e ao fazer ping broadcast do computador 2, não se chega a outro computador. Isto deve-se às redes criadas préviamente.

```
45 69.786320    172.16.50.1      172.16.50.255    ICMP    98 Echo (ping) request  id=0x0c16, seq=1/256, ttl=64…
46 69.786515    172.16.50.254    172.16.50.1      ICMP    98 Echo (ping) reply    id=0x0c16, seq=1/256, ttl=64
```

Figura 7 - computador 1 fazendo broadcast, resposta de computador 4

Experiência 3 – Configurar um *router* em Linux

Esta experiência teve por objetivo utilizar o computador 4 como *router*, de forma a ligar as duas sub-redes criadas na experiência anterior.

Para este efeito, configurou-se a porta eth1 do computador 4 com um endereço IP da sub-rede do computador 2, e também, alteraram-se algumas configurações de forma a ativar *IP Forwarding* e desativar o *ICMP_echo_ignore_broadcast*. Definiram-se também as rotas necessárias para cada sub-rede ter acesso à outra, usando o computador 4 como *gateway*.

Que rotas existem nos computadores? Qual o seu significado?

O computador 1 tem uma rota para a sub-rede do computador 2 e vice-versa, por exemplo, "172.16.11.1 172.16.10.254 255.255.255.0". Esta rota permite ao computador 1 saber que para o destino 172.16.11.1 (computador 2), deve usar o endereço 172.16.10.254 como *gateway*, neste caso é o computador 4 (*router*).

Que informação contém uma entrada na tabela de encaminhamento?

As tabelas de encaminhamento contém a informação necessária para fazer chegar pacotes ao endereço IP destino. Para isto contêm informação de destino e para onde deve ser encaminhado o pacote para chegar ao destino.

```
ubuntu@ubuntu-VirtualBox:/media/sf_MintShare$ route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         10.0.2.2        0.0.0.0         UG    1024   0        0 eth0
10.0.2.0        0.0.0.0         255.255.255.0   U     0      0        0 eth0
169.254.0.0     0.0.0.0         255.255.0.0     U     1000   0        0 eth0
```

Figura 8 - exemplo de tabela de encaminhamento

Qual o comportamento dos pacotes ARP e ICMP nesta rede?

Estando no computador 1 e fazendo ping ao computador 2, verifica-se que o computador 1 envia para o computador 4 um pacote ICMP com o endereço IP do computador 2 no campo de destino. O computador 4, caso não tenha em cache o MAC do computador 2, envia um pacote ARP para descobrir o MAC associado ao endereço IP do ping recebido, após descobrir este MAC envia o ping original ao computador 2. O processo de resposta é idêntico com os valores invertidos, o computador 4 serve então de *router*, como é demonstrado por este comportamento.

Experiência 4 – Configurar um *router* comercial e implementar NAT

Esta experiência teve por objetivo adicionar à rede um *router* comercial, este, configurado com NAT para permitir acesso à Internet.

Para este efeito, adicionou-se à VLAN do computador 2/4, o *router* comercial, de seguida, configuraram-se as rotas necessárias para ser possível receber pacotes de qualquer combinação de computadores. Finalmente, configurou-se NAT no *router*, para permitir acesso a redes externas.

Qual o caminho seguido pelos pacotes na experiência?

Inicialmente, o computador 2 tem apenas uma rota através do *router* comercial, fazendo *ping* a um endereço da outra sub-rede tem por resultado o pacote parar primeiro no *router* comercial, de seguida pára no computador 4 e daí é feita a ligação à outra sub-rede. Adicionando a rota à outra sub-rede, usando o computador 4 como *gateway*, faz com que o caminho anterior se torne mais curto, pois não tem de passar no *router* comercial primeiro.

O que faz o NAT?

NAT permite mapear um espaço de endereçamento IP a outro espaço, modificando o cabeçalho de datagramas, enquanto estes transitam num dispositivo de *routing*. O seu uso atual consiste em possibilitar que uma rede interna seja endereçada apenas por um único endereço IP público, sendo que esta utilização é denomeada de *IP masquerading*.

Experiência 5 – DNS

Esta experiência teve por objetivo a configuração de DNS nos computadores da rede.

Para este efeito, bastou editar um ficheiro de configuração, para adicionar um *nameserver*, o qual traduz um *hostname* para um endereço IP.

```
1  search netlab.fe.up.pt
2  nameserver 172.16.1.1
```

Figura 9 - resolv.conf

Que pacotes são trocados pelo DNS e que informação é transportada?

Quando se executa um *ping* a um *hostname*, é enviado um pacote "DNS *standard query*" que contém parâmetros *name*, *type* e *class*. A resposta a este pacote, é um pacote "DNS *standard query response*" que contém, além dos parâmetros mencionados, um campo *Time to live* e *Data length*.

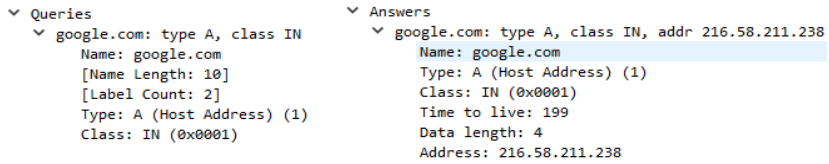


Figura 10 - exemplo de pacotes DNS

Experiência 6 – Ligações TCP

Esta experiência teve por objetivo testar a aplicação de *download* na rede configurada. Para este efeito, foram testados diversos servidores FTP, nos quais, se experimentou fazer transferência de diferentes tipos de ficheiro, como por exemplo, zip, png e txt.

Quantas ligações TCP são criadas pela aplicação?

O protocolo FTP funciona com uma ligação de dados e uma ligação de controlo, por isso, são abertas duas ligações TCP por transferência.

Em que ligação é transportada a informação de controlo de FTP?

A informação de controlo é transportada na ligação TCP da porta 21, que é a porta por defeito de serviços FTP.

Quais são as fases de uma ligação TCP?

Uma ligação TCP pode consistir de três fases, as quais são:

- Um processo de *handshake* com vários passos, SYN -> SYN-ACK -> ACK.
- Uma fase de transferência de dados.
- Finalmente, uma fase de término de ligação, que consiste noutro *handshake*.

Como funciona o mecanismo ARQ de TPC? Quais sãos os campos relevantes a TCP?

O mecanismo ARQ de TCP funciona com uma "janela", o recetor envia o tamanho desta "janela" no pacote TCP, e o emissor envia no máximo, até este valor de bytes. Após enviar os bytes, espera por um ACK do recetor e nova "janela". Os campos relevantes no pacote TCP a este mecanismo são *Sequence number*, *Acknowledgment number* e *Window Size*.

Como funciona o mecanismo de controlo de congestão de TCP? Quais são os campos relevantes?

O mecanismo de congestão de TCP usa o sistema *slow-start*, o qual começa por enviar dados em número reduzido. Este número é duplicado a cada ACK recebido até ocorrer uma falha na transmissão, ao fim da qual, a ligação entra em fase de *congestion avoidance*, na qual o algoritmo utilizado varia com a implementação, o mais simplista sendo reduzir o número de bytes enviados de volta ao ínicio do *slow-start*. Os campos relevantes no pacote TCP são as *flags* CWR, ECE e URG, juntamente com o campo *Urgent pointer*.

A taxa de transferência de uma ligação de dados TCP é perturbada pela aparência de uma segunda ligação TCP? Em que sentido?

A aparência de uma segunda ligação reduz para cerca de metade a taxa de transferência, isto pode ser visualizado na captura seguinte.

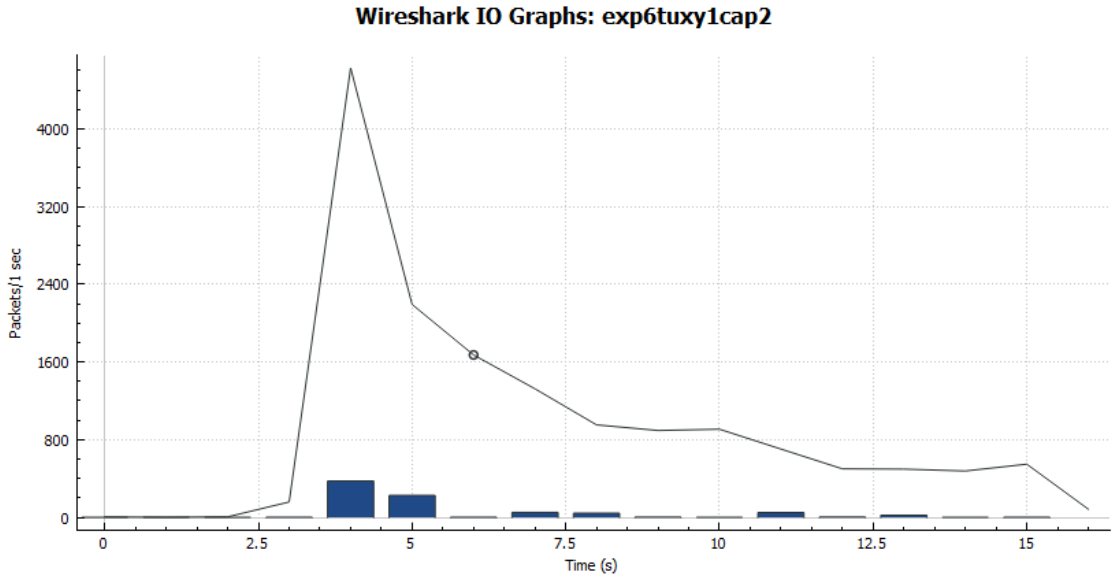


Figura 11 - taxa de transferência

Conclusão

Com a conclusão do relatório e da análise da aplicação de *download*, como das experiências laboratoriais, considera-se que o grupo conseguiu alcançar os objetivos do segundo trabalho.

Como descrito na primeira seção, o grupo implementou uma aplicação que utiliza o protocolo FTP, de acordo com o RFC959. Isto permitiu consolidar o conhecimento sobre FTP e sobre ligações TCP.

Na segunda seção foi descrita a configuração de uma rede, sendo que, o grupo aprendeu, os conhecimentos base necessários, para conseguir configurar redes complexas. Estes conhecimentos passaram por configurar endereços IP de computadores, implementação de VLAN num *switch* e configuração de rotas e NAT num *router* comercial. Finalizando, com a utilização da aplicação para efetuar uma transferência na rede, confirmando assim, a apreensão dos conhecimentos do trabalho.

Posto isto, é considerado que o trabalho foi bem sucedido e os seus objetivos, maioritariamente, compreendidos.

Anexos

Anexo 1 – Comandos de configuração

Para todos os comandos seguintes, y é o número de bancada da sala I321. Os comandos assumem uma configuração incremental começando na primeira experiência.

Experiência 1

Computador 1
ifconfig eth0 down
ifconfig eth1 down
ifconfig eth0 172.16.y0.1/24

Computador 4
ifconfig eth0 down
ifconfig eth1 down
ifconfig eth0 172.16.y0.254/24

Experiência 2

Computador 2
ifconfig eth0 down
ifconfig eth1 down
ifconfig eth0 172.16.y1.1/24

Switch
conf t
vlan y0
exit
vlan y1
end

conf t
interface fastethernet 0/1
switchport mode access
switchport access vlan y0
end

conf t
interface fastethernet 0/4
switchport mode access
switchport access vlan y0
end

conf t
interface fastethernet 0/2
switchport mode access
switchport access vlan y1
end

Experiência 3

Computador 1
route add -net 172.16.y1.0/24 gw 172.16.y0.254

Computador 2
route add -net 172.16.y0.0/24 gw 172.16.y1.253

Computador 4
ifconfig eth1 172.16.y1.253/24
echo 1 > /proc/sys/net/ipv4/ip_forward
echo 0 > /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts

Switch
conf t
interface fastethernet 0/14
switchport mode access
switchport access vlan y1
end

Experiência 4 (sem NAT)

Switch
conf t
interface fastethernet 0/20
switchport mode access
switchport access vlan y1
end

Computador 1
route add -net default gw 172.16.y0.254
route del -net 172.16.y1.0/24

Computador 2
route add -net default gw 172.16.y1.254

Computador 4
route add -net default gw 172.16.y1.254

Router
conf t
interface gigabitethernet 0/0
ip address 172.16.y1.254 255.255.255.0
no shutdown
end

conf t
ip route 172.16.y0.0 255.255.255.0 172.16.y1.253
end

conf t
interface gigabitethernet 0/1
ip address 172.16.1.y9 255.255.255.0
no shutdown
end

conf t
ip route 0.0.0.0 0.0.0.0 172.16.1.254
end

Experiência 4 (NAT)

Router
conf t
interface gigabitethernet 0/0
ip address 172.16.y1.254 255.255.255.0
no shutdown
ip nat inside
exit

interface gigabitethernet 0/1
ip address 172.16.1.y9 255.255.255.0
no shutdown
ip nat outside
end
ip nat pool ovrld 172.16.1.y9 172.16.1.y9 prefix 24
ip nat inside source list 1 pool ovrld overload

access-list 1 permit 172.16.y0.0 0.0.0.255
access-list 1 permit 172.16.y1.0.0 0.0.0.255

ip route 0.0.0.0 0.0.0.0 172.16.1.254
ip route 172.16.y0.0 255.255.255.0 172.16.y1.253
end

Experiência 5

Computador 1/2/4
Editar /etc/resolv.conf para conter:
"search netlab.fe.up.pt
nameserver 172.16.1.1"

Anexo 2 – Capturas

Experiência 1

1	0.000000	G-ProCom_8b:e4:a7	Broadcast	ARP	42 Who has 172.16.50.254? Tell 172.16.50.1
2	0.000115	HewlettP_c3:78:70	G-ProCom_8b:e4:a7	ARP	60 172.16.50.254 is at 00:21:5a:c3:78:70
3	0.000136	172.16.50.1	172.16.50.254	ICMP	98 Echo (ping) request id=0x08d9, seq=1/256, ttl=64 (reply in 4)
4	0.000278	172.16.50.254	172.16.50.1	ICMP	98 Echo (ping) reply id=0x08d9, seq=1/256, ttl=64 (request in 3)
5	0.999001	172.16.50.1	172.16.50.254	ICMP	98 Echo (ping) request id=0x08d9, seq=2/512, ttl=64 (reply in 6)
6	0.999088	172.16.50.254	172.16.50.1	ICMP	98 Echo (ping) reply id=0x08d9, seq=2/512, ttl=64 (request in 5)

Figura 12 – ping computador 4

Experiência 2

23	18.252096	172.16.50.1	172.16.50.254	ICMP	98 Echo (ping) request id=0x0b5d, seq=6/1536, ttl=64 (reply in 24)
24	18.252250	172.16.50.254	172.16.50.1	ICMP	98 Echo (ping) reply id=0x0b5d, seq=6/1536, ttl=64 (request in 23)
25	19.252099	172.16.50.1	172.16.50.254	ICMP	98 Echo (ping) request id=0x0b5d, seq=7/1792, ttl=64 (reply in 26)
26	19.252338	172.16.50.254	172.16.50.1	ICMP	98 Echo (ping) reply id=0x0b5d, seq=7/1792, ttl=64 (request in 25)
27	20.048744	Cisco_3a:f6:03	Spanning-tree-(for-…	STP	60 Conf. Root = 32768/50/fc:fb:fb:3a:f6:00 Cost = 0 Port = 0x8003
28	20.252102	172.16.50.1	172.16.50.254	ICMP	98 Echo (ping) request id=0x0b5d, seq=8/2048, ttl=64 (reply in 29)
29	20.252258	172.16.50.254	172.16.50.1	ICMP	98 Echo (ping) reply id=0x0b5d, seq=8/2048, ttl=64 (request in 28)
30	20.585558	Cisco_3a:f6:03	Cisco_3a:f6:03	LOOP	60 Reply
31	21.252098	172.16.50.1	172.16.50.254	ICMP	98 Echo (ping) request id=0x0b5d, seq=9/2304, ttl=64 (reply in 32)
32	21.252298	172.16.50.254	172.16.50.1	ICMP	98 Echo (ping) reply id=0x0b5d, seq=9/2304, ttl=64 (request in 31)
33	21.266743	HewlettP_c3:78:70	G-ProCom_8b:e4:a7	ARP	60 Who has 172.16.50.1? Tell 172.16.50.254
34	21.266756	G-ProCom_8b:e4:a7	HewlettP_c3:78:70	ARP	42 172.16.50.1 is at 00:0f:fe:8b:e4:a7
35	22.058415	Cisco_3a:f6:03	Spanning-tree-(for-…	STP	60 Conf. Root = 32768/50/fc:fb:fb:3a:f6:00 Cost = 0 Port = 0x8003
36	22.252106	172.16.50.1	172.16.50.254	ICMP	98 Echo (ping) request id=0x0b5d, seq=10/2560, ttl=64 (reply in 37)
37	22.252267	172.16.50.254	172.16.50.1	ICMP	98 Echo (ping) reply id=0x0b5d, seq=10/2560, ttl=64 (request in 36)

Figura 13 - ping computador 4 em VLAN

45	69.786320	172.16.50.1	172.16.50.255	ICMP	98 Echo (ping) request id=0x0c16, seq=1/256, ttl=64 (no response found!)
46	69.786515	172.16.50.254	172.16.50.1	ICMP	98 Echo (ping) reply id=0x0c16, seq=1/256, ttl=64
47	70.186769	Cisco_3a:f6:03	Spanning-tree-(for-…	STP	60 Conf. Root = 32768/50/fc:fb:fb:3a:f6:00 Cost = 0 Port = 0x8003
48	70.421869	Cisco_3a:f6:03	Cisco_3a:f6:03	LOOP	60 Reply
49	70.785323	172.16.50.1	172.16.50.255	ICMP	98 Echo (ping) request id=0x0c16, seq=2/512, ttl=64 (no response found!)
50	70.785709	172.16.50.254	172.16.50.1	ICMP	98 Echo (ping) reply id=0x0c16, seq=2/512, ttl=64
51	71.784806	172.16.50.1	172.16.50.255	ICMP	98 Echo (ping) request id=0x0c16, seq=3/768, ttl=64 (no response found!)
52	71.785000	172.16.50.254	172.16.50.1	ICMP	98 Echo (ping) reply id=0x0c16, seq=3/768, ttl=64
53	72.191644	Cisco_3a:f6:03	Spanning-tree-(for-…	STP	60 Conf. Root = 32768/50/fc:fb:fb:3a:f6:00 Cost = 0 Port = 0x8003
54	72.784806	172.16.50.1	172.16.50.255	ICMP	98 Echo (ping) request id=0x0c16, seq=4/1024, ttl=64 (no response found!)
55	72.784996	172.16.50.254	172.16.50.1	ICMP	98 Echo (ping) reply id=0x0c16, seq=4/1024, ttl=64
56	73.784809	172.16.50.1	172.16.50.255	ICMP	98 Echo (ping) request id=0x0c16, seq=5/1280, ttl=64 (no response found!)
57	73.785002	172.16.50.254	172.16.50.1	ICMP	98 Echo (ping) reply id=0x0c16, seq=5/1280, ttl=64
58	74.196692	Cisco_3a:f6:03	Spanning-tree-(for-…	STP	60 Conf. Root = 32768/50/fc:fb:fb:3a:f6:00 Cost = 0 Port = 0x8003
59	74.784800	172.16.50.1	172.16.50.255	ICMP	98 Echo (ping) request id=0x0c16, seq=6/1536, ttl=64 (no response found!)
60	74.784994	172.16.50.254	172.16.50.1	ICMP	98 Echo (ping) reply id=0x0c16, seq=6/1536, ttl=64
61	74.792394	HewlettP_c3:78:70	G-ProCom_8b:e4:a7	ARP	60 Who has 172.16.50.1? Tell 172.16.50.254
62	74.792406	G-ProCom_8b:e4:a7	HewlettP_c3:78:70	ARP	42 172.16.50.1 is at 00:0f:fe:8b:e4:a7
63	75.784805	172.16.50.1	172.16.50.255	ICMP	98 Echo (ping) request id=0x0c16, seq=7/1792, ttl=64 (no response found!)
64	75.785001	172.16.50.254	172.16.50.1	ICMP	98 Echo (ping) reply id=0x0c16, seq=7/1792, ttl=64

Figura 14 - pacotes no computador 1 recebidos por fazer ping broadcast no computador 1

2	2.004593	Cisco_3a:f6:04	Spanning-tree-(for-…	STP	60 Conf. Root = 32768/51/fc:fb:fb:3a:f6:00 Cost = 0 Port = 0x8004
3	4.009591	Cisco_3a:f6:04	Spanning-tree-(for-…	STP	60 Conf. Root = 32768/51/fc:fb:fb:3a:f6:00 Cost = 0 Port = 0x8004
4	6.014363	Cisco_3a:f6:04	Spanning-tree-(for-…	STP	60 Conf. Root = 32768/51/fc:fb:fb:3a:f6:00 Cost = 0 Port = 0x8004
5	7.051483	Cisco_3a:f6:04	Cisco_3a:f6:04	LOOP	60 Reply
6	8.019357	Cisco_3a:f6:04	Spanning-tree-(for-…	STP	60 Conf. Root = 32768/51/fc:fb:fb:3a:f6:00 Cost = 0 Port = 0x8004
7	10.024125	Cisco_3a:f6:04	Spanning-tree-(for-…	STP	60 Conf. Root = 32768/51/fc:fb:fb:3a:f6:00 Cost = 0 Port = 0x8004
8	12.029043	Cisco_3a:f6:04	Spanning-tree-(for-…	STP	60 Conf. Root = 32768/51/fc:fb:fb:3a:f6:00 Cost = 0 Port = 0x8004
9	14.033910	Cisco_3a:f6:04	Spanning-tree-(for-…	STP	60 Conf. Root = 32768/51/fc:fb:fb:3a:f6:00 Cost = 0 Port = 0x8004
10	16.038796	Cisco_3a:f6:04	Spanning-tree-(for-…	STP	60 Conf. Root = 32768/51/fc:fb:fb:3a:f6:00 Cost = 0 Port = 0x8004
11	17.059115	Cisco_3a:f6:04	Cisco_3a:f6:04	LOOP	60 Reply
12	18.043663	Cisco_3a:f6:04	Spanning-tree-(for-…	STP	60 Conf. Root = 32768/51/fc:fb:fb:3a:f6:00 Cost = 0 Port = 0x8004
13	20.048556	Cisco_3a:f6:04	Spanning-tree-(for-…	STP	60 Conf. Root = 32768/51/fc:fb:fb:3a:f6:00 Cost = 0 Port = 0x8004
14	21.743330	Cisco_3a:f6:04	CDP/VTP/DTP/PAgP/UD…	CDP	453 Device ID: tux-sw5 Port ID: FastEthernet0/2
15	22.053632	Cisco_3a:f6:04	Spanning-tree-(for-…	STP	60 Conf. Root = 32768/51/fc:fb:fb:3a:f6:00 Cost = 0 Port = 0x8004
16	24.058625	Cisco_3a:f6:04	Spanning-tree-(for-…	STP	60 Conf. Root = 32768/51/fc:fb:fb:3a:f6:00 Cost = 0 Port = 0x8004
17	26.063212	Cisco_3a:f6:04	Spanning-tree-(for-…	STP	60 Conf. Root = 32768/51/fc:fb:fb:3a:f6:00 Cost = 0 Port = 0x8004
18	27.061850	Cisco_3a:f6:04	Cisco_3a:f6:04	LOOP	60 Reply
19	28.068089	Cisco_3a:f6:04	Spanning-tree-(for-…	STP	60 Conf. Root = 32768/51/fc:fb:fb:3a:f6:00 Cost = 0 Port = 0x8004
20	30.072971	Cisco_3a:f6:04	Spanning-tree-(for-…	STP	60 Conf. Root = 32768/51/fc:fb:fb:3a:f6:00 Cost = 0 Port = 0x8004
21	32.077880	Cisco_3a:f6:04	Spanning-tree-(for-…	STP	60 Conf. Root = 32768/51/fc:fb:fb:3a:f6:00 Cost = 0 Port = 0x8004
22	34.082738	Cisco_3a:f6:04	Spanning-tree-(for-…	STP	60 Conf. Root = 32768/51/fc:fb:fb:3a:f6:00 Cost = 0 Port = 0x8004
23	36.087818	Cisco_3a:f6:04	Spanning-tree-(for-…	STP	60 Conf. Root = 32768/51/fc:fb:fb:3a:f6:00 Cost = 0 Port = 0x8004
24	37.061047	Cisco_3a:f6:04	Cisco_3a:f6:04	LOOP	60 Reply

Figura 15 - pacotes no computador 2 recebidos por fazer ping broadcast no computador 1

14	21.652115	172.16.50.1	172.16.50.255	ICMP	98	Echo (ping) request id=0x0c16, seq=1/256, ttl=64 (no response found!)
15	21.652165	172.16.50.254	172.16.50.1	ICMP	98	Echo (ping) reply id=0x0c16, seq=1/256, ttl=64
16	22.052959	Cisco_3a:f6:06	Spanning-tree-(for-…	STP	60	Conf. Root = 32768/50/fc:fb:fb:3a:f6:00 Cost = 0 Port = 0x8006
17	22.287806	Cisco_3a:f6:06	Cisco_3a:f6:06	LOOP	60	Reply
18	22.651108	172.16.50.1	172.16.50.255	ICMP	98	Echo (ping) request id=0x0c16, seq=2/512, ttl=64 (no response found!)
19	22.651152	172.16.50.254	172.16.50.1	ICMP	98	Echo (ping) reply id=0x0c16, seq=2/512, ttl=64
20	23.650585	172.16.50.1	172.16.50.255	ICMP	98	Echo (ping) request id=0x0c16, seq=3/768, ttl=64 (no response found!)
21	23.650627	172.16.50.254	172.16.50.1	ICMP	98	Echo (ping) reply id=0x0c16, seq=3/768, ttl=64
22	24.058431	Cisco_3a:f6:06	Spanning-tree-(for-…	STP	60	Conf. Root = 32768/50/fc:fb:fb:3a:f6:00 Cost = 0 Port = 0x8006
23	24.650578	172.16.50.1	172.16.50.255	ICMP	98	Echo (ping) request id=0x0c16, seq=4/1024, ttl=64 (no response found!)
24	24.650618	172.16.50.254	172.16.50.1	ICMP	98	Echo (ping) reply id=0x0c16, seq=4/1024, ttl=64
25	25.650574	172.16.50.1	172.16.50.255	ICMP	98	Echo (ping) request id=0x0c16, seq=5/1280, ttl=64 (no response found!)
26	25.650621	172.16.50.254	172.16.50.1	ICMP	98	Echo (ping) reply id=0x0c16, seq=5/1280, ttl=64
27	26.045690	Cisco_3a:f6:06	CDP/VTP/DTP/PAgP/UD…	CDP	453	Device ID: tux-sw5 Port ID: FastEthernet0/4
28	26.063721	Cisco_3a:f6:06	Spanning-tree-(for-…	STP	60	Conf. Root = 32768/50/fc:fb:fb:3a:f6:00 Cost = 0 Port = 0x8006
29	26.650560	172.16.50.1	172.16.50.255	ICMP	98	Echo (ping) request id=0x0c16, seq=6/1536, ttl=64 (no response found!)
30	26.650600	172.16.50.254	172.16.50.1	ICMP	98	Echo (ping) reply id=0x0c16, seq=6/1536, ttl=64

Figura 16 - pacotes no computador 4 recebidos por fazer ping broadcast no computador 1

1	0.000000	Cisco_3a:f6:03	Spanning-tree-(for-…	STP	60	Conf. Root = 32768/50/fc:fb:fb:3a:f6:00 Cost = 0 Port = 0x8003
2	1.999986	Cisco_3a:f6:03	Spanning-tree-(for-…	STP	60	Conf. Root = 32768/50/fc:fb:fb:3a:f6:00 Cost = 0 Port = 0x8003
3	4.004865	Cisco_3a:f6:03	Spanning-tree-(for-…	STP	60	Conf. Root = 32768/50/fc:fb:fb:3a:f6:00 Cost = 0 Port = 0x8003
4	5.942593	Cisco_3a:f6:03	Cisco_3a:f6:03	LOOP	60	Reply
5	6.014892	Cisco_3a:f6:03	Spanning-tree-(for-…	STP	60	Conf. Root = 32768/50/fc:fb:fb:3a:f6:00 Cost = 0 Port = 0x8003
6	8.014369	Cisco_3a:f6:03	Spanning-tree-(for-…	STP	60	Conf. Root = 32768/50/fc:fb:fb:3a:f6:00 Cost = 0 Port = 0x8003
7	10.019576	Cisco_3a:f6:03	Spanning-tree-(for-…	STP	60	Conf. Root = 32768/50/fc:fb:fb:3a:f6:00 Cost = 0 Port = 0x8003
8	12.029397	Cisco_3a:f6:03	Spanning-tree-(for-…	STP	60	Conf. Root = 32768/50/fc:fb:fb:3a:f6:00 Cost = 0 Port = 0x8003
9	14.029226	Cisco_3a:f6:03	Spanning-tree-(for-…	STP	60	Conf. Root = 32768/50/fc:fb:fb:3a:f6:00 Cost = 0 Port = 0x8003
10	15.941805	Cisco_3a:f6:03	Cisco_3a:f6:03	LOOP	60	Reply
11	16.033917	Cisco_3a:f6:03	Spanning-tree-(for-…	STP	60	Conf. Root = 32768/50/fc:fb:fb:3a:f6:00 Cost = 0 Port = 0x8003
12	18.044244	Cisco_3a:f6:03	Spanning-tree-(for-…	STP	60	Conf. Root = 32768/50/fc:fb:fb:3a:f6:00 Cost = 0 Port = 0x8003
13	20.043679	Cisco_3a:f6:03	Spanning-tree-(for-…	STP	60	Conf. Root = 32768/50/fc:fb:fb:3a:f6:00 Cost = 0 Port = 0x8003
14	22.048565	Cisco_3a:f6:03	Spanning-tree-(for-…	STP	60	Conf. Root = 32768/50/fc:fb:fb:3a:f6:00 Cost = 0 Port = 0x8003
15	24.061045	Cisco_3a:f6:03	Spanning-tree-(for-…	STP	60	Conf. Root = 32768/50/fc:fb:fb:3a:f6:00 Cost = 0 Port = 0x8003
16	25.954236	Cisco_3a:f6:03	Cisco_3a:f6:03	LOOP	60	Reply
17	26.058206	Cisco_3a:f6:03	Spanning-tree-(for-…	STP	60	Conf. Root = 32768/50/fc:fb:fb:3a:f6:00 Cost = 0 Port = 0x8003
18	28.063085	Cisco_3a:f6:03	Spanning-tree-(for-…	STP	60	Conf. Root = 32768/50/fc:fb:fb:3a:f6:00 Cost = 0 Port = 0x8003
19	30.072984	Cisco_3a:f6:03	Spanning-tree-(for-…	STP	60	Conf. Root = 32768/50/fc:fb:fb:3a:f6:00 Cost = 0 Port = 0x8003
20	31.812873	Cisco_3a:f6:03	CDP/VTP/DTP/PAgP/UD…	CDP	453	Device ID: tux-sw5 Port ID: FastEthernet0/1
21	32.078122	Cisco_3a:f6:03	Spanning-tree-(for-…	STP	60	Conf. Root = 32768/50/fc:fb:fb:3a:f6:00 Cost = 0 Port = 0x8003
22	34.077703	Cisco_3a:f6:03	Spanning-tree-(for-…	STP	60	Conf. Root = 32768/50/fc:fb:fb:3a:f6:00 Cost = 0 Port = 0x8003
23	35.961711	Cisco_3a:f6:03	Cisco_3a:f6:03	LOOP	60	Reply
24	36.087454	Cisco_3a:f6:03	Spanning-tree-(for-…	STP	60	Conf. Root = 32768/50/fc:fb:fb:3a:f6:00 Cost = 0 Port = 0x8003
25	38.087557	Cisco_3a:f6:03	Spanning-tree-(for-…	STP	60	Conf. Root = 32768/50/fc:fb:fb:3a:f6:00 Cost = 0 Port = 0x8003
26	40.092258	Cisco_3a:f6:03	Spanning-tree-(for-…	STP	60	Conf. Root = 32768/50/fc:fb:fb:3a:f6:00 Cost = 0 Port = 0x8003
27	42.102183	Cisco_3a:f6:03	Spanning-tree-(for-…	STP	60	Conf. Root = 32768/50/fc:fb:fb:3a:f6:00 Cost = 0 Port = 0x8003
28	44.101930	Cisco_3a:f6:03	Spanning-tree-(for-…	STP	60	Conf. Root = 32768/50/fc:fb:fb:3a:f6:00 Cost = 0 Port = 0x8003

Figura 17 - pacotes no computador 1 recebidos por fazer ping broadcast no computador 2

4	4.614169	Cisco_3a:f6:04	Cisco_3a:f6:04	LOOP	60	Reply
5	6.014595	Cisco_3a:f6:04	Spanning-tree-(for-…	STP	60	Conf. Root = 32768/51/fc:fb:fb:3a:f6:00 Cost = 0 Port = 0x8004
6	7.894622	172.16.51.1	172.16.51.255	ICMP	98	Echo (ping) request id=0x7c70, seq=1/256, ttl=64 (no response found!)
7	8.024476	Cisco_3a:f6:04	Spanning-tree-(for-…	STP	60	Conf. Root = 32768/51/fc:fb:fb:3a:f6:00 Cost = 0 Port = 0x8004
8	8.894386	172.16.51.1	172.16.51.255	ICMP	98	Echo (ping) request id=0x7c70, seq=2/512, ttl=64 (no response found!)
9	9.894383	172.16.51.1	172.16.51.255	ICMP	98	Echo (ping) request id=0x7c70, seq=3/768, ttl=64 (no response found!)
10	10.024324	Cisco_3a:f6:04	Spanning-tree-(for-…	STP	60	Conf. Root = 32768/51/fc:fb:fb:3a:f6:00 Cost = 0 Port = 0x8004
11	10.894387	172.16.51.1	172.16.51.255	ICMP	98	Echo (ping) request id=0x7c70, seq=4/1024, ttl=64 (no response found!)
12	11.894384	172.16.51.1	172.16.51.255	ICMP	98	Echo (ping) request id=0x7c70, seq=5/1280, ttl=64 (no response found!)
13	12.029217	Cisco_3a:f6:04	Spanning-tree-(for-…	STP	60	Conf. Root = 32768/51/fc:fb:fb:3a:f6:00 Cost = 0 Port = 0x8004
14	12.894390	172.16.51.1	172.16.51.255	ICMP	98	Echo (ping) request id=0x7c70, seq=6/1536, ttl=64 (no response found!)
15	13.894383	172.16.51.1	172.16.51.255	ICMP	98	Echo (ping) request id=0x7c70, seq=7/1792, ttl=64 (no response found!)
16	14.039158	Cisco_3a:f6:04	Spanning-tree-(for-…	STP	60	Conf. Root = 32768/51/fc:fb:fb:3a:f6:00 Cost = 0 Port = 0x8004
17	14.626649	Cisco_3a:f6:04	Cisco_3a:f6:04	LOOP	60	Reply
18	14.894367	172.16.51.1	172.16.51.255	ICMP	98	Echo (ping) request id=0x7c70, seq=8/2048, ttl=64 (no response found!)
19	15.894386	172.16.51.1	172.16.51.255	ICMP	98	Echo (ping) request id=0x7c70, seq=9/2304, ttl=64 (no response found!)

Figura 18 - pacotes no computador 2 recebidos por fazer ping broadcast no computador 2

1	0.000000	Cisco_3a:f6:06	Cisco_3a:f6:06	LOOP	60	Reply
2	0.072632	Cisco_3a:f6:06	Spanning-tree-(for-…	STP	60	Conf. Root = 32768/50/fc:fb:fb:3a:f6:00 Cost = 0 Port = 0x8006
3	2.073342	Cisco_3a:f6:06	Spanning-tree-(for-…	STP	60	Conf. Root = 32768/50/fc:fb:fb:3a:f6:00 Cost = 0 Port = 0x8006
4	4.078367	Cisco_3a:f6:06	Spanning-tree-(for-…	STP	60	Conf. Root = 32768/50/fc:fb:fb:3a:f6:00 Cost = 0 Port = 0x8006
5	6.087395	Cisco_3a:f6:06	Spanning-tree-(for-…	STP	60	Conf. Root = 32768/50/fc:fb:fb:3a:f6:00 Cost = 0 Port = 0x8006
6	8.088137	Cisco_3a:f6:06	Spanning-tree-(for-…	STP	60	Conf. Root = 32768/50/fc:fb:fb:3a:f6:00 Cost = 0 Port = 0x8006
7	9.999242	Cisco_3a:f6:06	Cisco_3a:f6:06	LOOP	60	Reply
8	10.091349	Cisco_3a:f6:06	Spanning-tree-(for-…	STP	60	Conf. Root = 32768/50/fc:fb:fb:3a:f6:00 Cost = 0 Port = 0x8006
9	12.102488	Cisco_3a:f6:06	Spanning-tree-(for-…	STP	60	Conf. Root = 32768/50/fc:fb:fb:3a:f6:00 Cost = 0 Port = 0x8006
10	14.101455	Cisco_3a:f6:06	Spanning-tree-(for-…	STP	60	Conf. Root = 32768/50/fc:fb:fb:3a:f6:00 Cost = 0 Port = 0x8006
11	16.106532	Cisco_3a:f6:06	Spanning-tree-(for-…	STP	60	Conf. Root = 32768/50/fc:fb:fb:3a:f6:00 Cost = 0 Port = 0x8006
12	18.119829	Cisco_3a:f6:06	Spanning-tree-(for-…	STP	60	Conf. Root = 32768/50/fc:fb:fb:3a:f6:00 Cost = 0 Port = 0x8006
13	20.011512	Cisco_3a:f6:06	Cisco_3a:f6:06	LOOP	60	Reply

Figura 19 - pacotes no computador 4 recebidos por fazer ping broadcast no computador 2

Experiência 3

43	23.006181	172.16.50.1	172.16.51.253	ICMP	98 Echo (ping) request id=0x0f6e, seq=6/1536, ttl=64 (reply in 44)
44	23.006334	172.16.51.253	172.16.50.1	ICMP	98 Echo (ping) reply id=0x0f6e, seq=6/1536, ttl=64 (request in 43)
45	24.058403	Cisco_3a:f6:03	Spanning-tree-(for-…	STP	60 Conf. Root = 32768/50/fc:fb:fb:3a:f6:00 Cost = 0 Port = 0x8003
46	24.637390	Cisco_3a:f6:03	Cisco_3a:f6:03	LOOP	60 Reply
47	25.591989	172.16.50.1	172.16.51.1	ICMP	98 Echo (ping) request id=0x0f72, seq=1/256, ttl=64 (reply in 48)
48	25.592498	172.16.51.1	172.16.50.1	ICMP	98 Echo (ping) reply id=0x0f72, seq=1/256, ttl=63 (request in 47)
49	26.063201	Cisco_3a:f6:03	Spanning-tree-(for-…	STP	60 Conf. Root = 32768/50/fc:fb:fb:3a:f6:00 Cost = 0 Port = 0x8003
50	26.590995	172.16.50.1	172.16.51.1	ICMP	98 Echo (ping) request id=0x0f72, seq=2/512, ttl=64 (reply in 51)
51	26.591515	172.16.51.1	172.16.50.1	ICMP	98 Echo (ping) reply id=0x0f72, seq=2/512, ttl=63 (request in 50)
52	27.590179	172.16.50.1	172.16.51.1	ICMP	98 Echo (ping) request id=0x0f72, seq=3/768, ttl=64 (reply in 53)
53	27.590654	172.16.51.1	172.16.50.1	ICMP	98 Echo (ping) reply id=0x0f72, seq=3/768, ttl=63 (request in 52)

Figura 20 - ping da outra sub-rede a partir do computador 1

50	78.676281	G-ProCom_8b:e4:a7	Broadcast	ARP	60 Who has 172.16.50.254? Tell 172.16.50.1
51	78.676300	HewlettP_c3:78:70	G-ProCom_8b:e4:a7	ARP	42 172.16.50.254 is at 00:21:5a:c3:78:70
52	78.676445	172.16.50.1	172.16.51.1	ICMP	98 Echo (ping) request id=0x1026, seq=1/256, ttl=64 (reply in 53)
53	78.676757	172.16.51.1	172.16.50.1	ICMP	98 Echo (ping) reply id=0x1026, seq=1/256, ttl=63 (request in 52)
54	79.675275	172.16.50.1	172.16.51.1	ICMP	98 Echo (ping) request id=0x1026, seq=2/512, ttl=64 (reply in 55)
55	79.675458	172.16.51.1	172.16.50.1	ICMP	98 Echo (ping) reply id=0x1026, seq=2/512, ttl=63 (request in 54)
56	80.210541	Cisco_3a:f6:06	Spanning-tree-(for-…	STP	60 Conf. Root = 32768/50/fc:fb:fb:3a:f6:00 Cost = 0 Port = 0x8006
57	80.674692	172.16.50.1	172.16.51.1	ICMP	98 Echo (ping) request id=0x1026, seq=3/768, ttl=64 (reply in 58)
58	80.674847	172.16.51.1	172.16.50.1	ICMP	98 Echo (ping) reply id=0x1026, seq=3/768, ttl=63 (request in 57)
59	81.674702	172.16.50.1	172.16.51.1	ICMP	98 Echo (ping) request id=0x1026, seq=4/1024, ttl=64 (reply in 60)
60	81.674869	172.16.51.1	172.16.50.1	ICMP	98 Echo (ping) reply id=0x1026, seq=4/1024, ttl=63 (request in 59)
61	82.215584	Cisco_3a:f6:06	Spanning-tree-(for-…	STP	60 Conf. Root = 32768/50/fc:fb:fb:3a:f6:00 Cost = 0 Port = 0x8006
62	82.674669	172.16.50.1	172.16.51.1	ICMP	98 Echo (ping) request id=0x1026, seq=5/1280, ttl=64 (reply in 63)
63	82.674827	172.16.51.1	172.16.50.1	ICMP	98 Echo (ping) reply id=0x1026, seq=5/1280, ttl=63 (request in 62)
64	83.674679	172.16.50.1	172.16.51.1	ICMP	98 Echo (ping) request id=0x1026, seq=6/1536, ttl=64 (reply in 65)
65	83.674858	172.16.51.1	172.16.50.1	ICMP	98 Echo (ping) reply id=0x1026, seq=6/1536, ttl=63 (request in 64)

Figura 21 - pacotes no computador 4 ethernet 0 quando o computador 1 faz ping ao computador 2

48	75.271217	Kye_08:d5:b0	Broadcast	ARP	42 Who has 172.16.51.1? Tell 172.16.51.253
49	75.271337	HewlettP_61:2f:d6	Kye_08:d5:b0	ARP	60 172.16.51.1 is at 00:21:5a:61:2f:d6
50	75.271347	172.16.50.1	172.16.51.1	ICMP	98 Echo (ping) request id=0x1026, seq=1/256, ttl=63 (reply in 51)
51	75.271476	172.16.51.1	172.16.50.1	ICMP	98 Echo (ping) reply id=0x1026, seq=1/256, ttl=64 (request in 50)
52	76.200469	Cisco_3a:f6:10	Spanning-tree-(for-…	STP	60 Conf. Root = 32768/51/fc:fb:fb:3a:f6:00 Cost = 0 Port = 0x8010
53	76.270034	172.16.50.1	172.16.51.1	ICMP	98 Echo (ping) request id=0x1026, seq=2/512, ttl=63 (reply in 54)
54	76.270161	172.16.51.1	172.16.50.1	ICMP	98 Echo (ping) reply id=0x1026, seq=2/512, ttl=64 (request in 53)
55	77.269450	172.16.50.1	172.16.51.1	ICMP	98 Echo (ping) request id=0x1026, seq=3/768, ttl=63 (reply in 56)
56	77.269563	172.16.51.1	172.16.50.1	ICMP	98 Echo (ping) reply id=0x1026, seq=3/768, ttl=64 (request in 55)
57	78.211730	Cisco_3a:f6:10	Spanning-tree-(for-…	STP	60 Conf. Root = 32768/51/fc:fb:fb:3a:f6:00 Cost = 0 Port = 0x8010
58	78.269460	172.16.50.1	172.16.51.1	ICMP	98 Echo (ping) request id=0x1026, seq=4/1024, ttl=63 (reply in 59)
59	78.269575	172.16.51.1	172.16.50.1	ICMP	98 Echo (ping) reply id=0x1026, seq=4/1024, ttl=64 (request in 58)
60	79.269412	172.16.50.1	172.16.51.1	ICMP	98 Echo (ping) request id=0x1026, seq=5/1280, ttl=63 (reply in 61)
61	79.269529	172.16.51.1	172.16.50.1	ICMP	98 Echo (ping) reply id=0x1026, seq=5/1280, ttl=64 (request in 60)

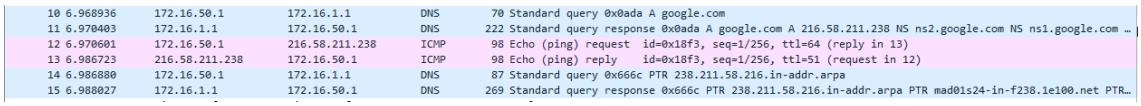
Figura 22 - pacotes no computador 4 ethernet 1 quando o computador 1 faz ping ao computador 2

Experiência 4

7	8.096043	172.16.51.1	172.16.50.1	ICMP	98 Echo (ping) request id=0x0689, seq=1/256, ttl=64 (reply in 9)
8	8.096357	172.16.51.254	172.16.51.1	ICMP	70 Redirect (Redirect for host)
9	8.096786	172.16.50.1	172.16.51.1	ICMP	98 Echo (ping) reply id=0x0689, seq=1/256, ttl=63 (request in 7)
10	9.095051	172.16.51.1	172.16.50.1	ICMP	98 Echo (ping) request id=0x0689, seq=2/512, ttl=64 (reply in 11)
11	9.095623	172.16.50.1	172.16.51.1	ICMP	98 Echo (ping) reply id=0x0689, seq=2/512, ttl=63 (request in 10)
12	10.024406	Cisco_3a:f6:04	Spanning-tree-(for-…	STP	60 Conf. Root = 32768/51/fc:fb:fb:3a:f6:00 Cost = 0 Port = 0x8004
13	10.094574	172.16.51.1	172.16.50.1	ICMP	98 Echo (ping) request id=0x0689, seq=3/768, ttl=64 (reply in 15)
14	10.094865	172.16.51.254	172.16.51.1	ICMP	70 Redirect (Redirect for host)
15	10.095059	172.16.50.1	172.16.51.1	ICMP	98 Echo (ping) reply id=0x0689, seq=3/768, ttl=63 (request in 13)
16	10.158848	Cisco_3a:f6:04	Cisco_3a:f6:04	LOOP	60 Reply
17	11.094550	172.16.51.1	172.16.50.1	ICMP	98 Echo (ping) request id=0x0689, seq=4/1024, ttl=64 (reply in 19)
18	11.094846	172.16.51.254	172.16.51.1	ICMP	70 Redirect (Redirect for host)
19	11.095253	172.16.50.1	172.16.51.1	ICMP	98 Echo (ping) reply id=0x0689, seq=4/1024, ttl=63 (request in 17)
20	12.029432	Cisco_3a:f6:04	Spanning-tree-(for-…	STP	60 Conf. Root = 32768/51/fc:fb:fb:3a:f6:00 Cost = 0 Port = 0x8004
21	12.094571	172.16.51.1	172.16.50.1	ICMP	98 Echo (ping) request id=0x0689, seq=5/1280, ttl=64 (reply in 23)
22	12.094873	172.16.51.254	172.16.51.1	ICMP	70 Redirect (Redirect for host)
23	12.095242	172.16.50.1	172.16.51.1	ICMP	98 Echo (ping) reply id=0x0689, seq=5/1280, ttl=63 (request in 21)
24	13.094567	172.16.51.1	172.16.50.1	ICMP	98 Echo (ping) request id=0x0689, seq=6/1536, ttl=64 (reply in 26)
25	13.094865	172.16.51.254	172.16.51.1	ICMP	70 Redirect (Redirect for host)
26	13.095242	172.16.50.1	172.16.51.1	ICMP	98 Echo (ping) reply id=0x0689, seq=6/1536, ttl=63 (request in 24)

Figura 23 - pacotes no computador 2 quando faz ping ao computador 1 sem rota direta para a sub-rede deste

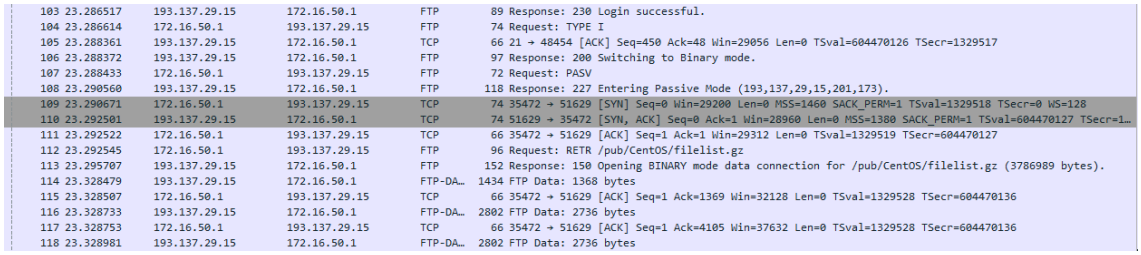
Experiência 5



10	6.968936	172.16.50.1	172.16.1.1	DNS	70 Standard query 0x0ada A google.com
11	6.970403	172.16.1.1	172.16.50.1	DNS	222 Standard query response 0x0ada A google.com A 216.58.211.238 NS ns2.google.com NS ns1.google.com …
12	6.970601	172.16.50.1	216.58.211.238	ICMP	98 Echo (ping) request id=0x18f3, seq=1/256, ttl=64 (reply in 13)
13	6.986723	216.58.211.238	172.16.50.1	ICMP	98 Echo (ping) reply id=0x18f3, seq=1/256, ttl=51 (request in 12)
14	6.986880	172.16.50.1	172.16.1.1	DNS	87 Standard query 0x666c PTR 238.211.58.216.in-addr.arpa
15	6.988027	172.16.1.1	172.16.50.1	DNS	269 Standard query response 0x666c PTR 238.211.58.216.in-addr.arpa PTR mad01s24-in-f238.1e100.net PTR…

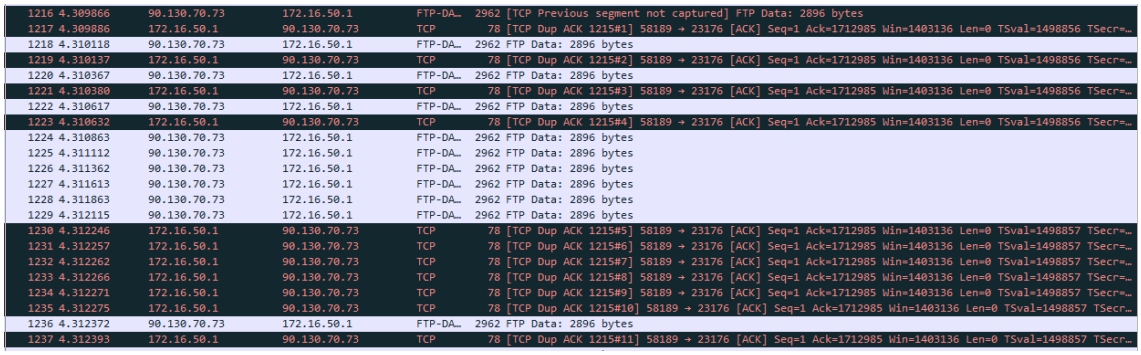
Figura 24 - pacotes DNS no computador 1 quando se faz ping a www.google.com

Experiência 6



103	23.286517	193.137.29.15	172.16.50.1	FTP	89 Response: 230 Login successful.
104	23.286614	172.16.50.1	193.137.29.15	FTP	74 Request: TYPE I
105	23.288361	193.137.29.15	172.16.50.1	TCP	66 21 → 48454 [ACK] Seq=450 Ack=48 Win=29056 Len=0 TSval=604470126 TSecr=1329517
106	23.288372	193.137.29.15	172.16.50.1	FTP	97 Response: 200 Switching to Binary mode.
107	23.288433	172.16.50.1	193.137.29.15	FTP	72 Request: PASV
108	23.290560	193.137.29.15	172.16.50.1	FTP	118 Response: 227 Entering Passive Mode (193,137,29,15,201,173).
109	23.290671	172.16.50.1	193.137.29.15	TCP	74 35472 → 51629 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=1329518 TSecr=0 WS=128
110	23.292501	193.137.29.15	172.16.50.1	TCP	74 51629 → 35472 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1380 SACK_PERM=1 TSval=604470127 TSecr=1…
111	23.292522	172.16.50.1	193.137.29.15	TCP	66 35472 → 51629 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=1329519 TSecr=604470127
112	23.292545	172.16.50.1	193.137.29.15	FTP	96 Request: RETR /pub/CentOS/filelist.gz
113	23.295707	193.137.29.15	172.16.50.1	FTP	152 Response: 150 Opening BINARY mode data connection for /pub/CentOS/filelist.gz (3786989 bytes).
114	23.328479	193.137.29.15	172.16.50.1	FTP-DA…	1434 FTP Data: 1368 bytes
115	23.328507	172.16.50.1	193.137.29.15	TCP	66 35472 → 51629 [ACK] Seq=1 Ack=1369 Win=32128 Len=0 TSval=1329528 TSecr=604470136
116	23.328733	193.137.29.15	172.16.50.1	FTP-DA…	2802 FTP Data: 2736 bytes
117	23.328753	172.16.50.1	193.137.29.15	TCP	66 35472 → 51629 [ACK] Seq=1 Ack=4105 Win=37632 Len=0 TSval=1329528 TSecr=604470136
118	23.328981	193.137.29.15	172.16.50.1	FTP-DA…	2802 FTP Data: 2736 bytes

Figura 25 - pacotes TCP no computador 1 quando transfere um ficheiro por FTP



1216	4.309866	90.130.70.73	172.16.50.1	FTP-DA…	2962 [TCP Previous segment not captured] FTP Data: 2896 bytes
1217	4.309886	172.16.50.1	90.130.70.73	TCP	78 [TCP Dup ACK 1215#1] 58189 → 23176 [ACK] Seq=1 Ack=1712985 Win=1403136 Len=0 TSval=1498856 TSecr=…
1218	4.310118	90.130.70.73	172.16.50.1	FTP-DA…	2962 FTP Data: 2896 bytes
1219	4.310137	172.16.50.1	90.130.70.73	TCP	78 [TCP Dup ACK 1215#2] 58189 → 23176 [ACK] Seq=1 Ack=1712985 Win=1403136 Len=0 TSval=1498856 TSecr=…
1220	4.310367	90.130.70.73	172.16.50.1	FTP-DA…	2962 FTP Data: 2896 bytes
1221	4.310380	172.16.50.1	90.130.70.73	TCP	78 [TCP Dup ACK 1215#3] 58189 → 23176 [ACK] Seq=1 Ack=1712985 Win=1403136 Len=0 TSval=1498856 TSecr=…
1222	4.310617	90.130.70.73	172.16.50.1	FTP-DA…	2962 FTP Data: 2896 bytes
1223	4.310632	172.16.50.1	90.130.70.73	TCP	78 [TCP Dup ACK 1215#4] 58189 → 23176 [ACK] Seq=1 Ack=1712985 Win=1403136 Len=0 TSval=1498856 TSecr=…
1224	4.310863	90.130.70.73	172.16.50.1	FTP-DA…	2962 FTP Data: 2896 bytes
1225	4.311112	90.130.70.73	172.16.50.1	FTP-DA…	2962 FTP Data: 2896 bytes
1226	4.311362	90.130.70.73	172.16.50.1	FTP-DA…	2962 FTP Data: 2896 bytes
1227	4.311613	90.130.70.73	172.16.50.1	FTP-DA…	2962 FTP Data: 2896 bytes
1228	4.311863	90.130.70.73	172.16.50.1	FTP-DA…	2962 FTP Data: 2896 bytes
1229	4.312115	90.130.70.73	172.16.50.1	FTP-DA…	2962 FTP Data: 2896 bytes
1230	4.312246	172.16.50.1	90.130.70.73	TCP	78 [TCP Dup ACK 1215#5] 58189 → 23176 [ACK] Seq=1 Ack=1712985 Win=1403136 Len=0 TSval=1498857 TSecr=…
1231	4.312257	172.16.50.1	90.130.70.73	TCP	78 [TCP Dup ACK 1215#6] 58189 → 23176 [ACK] Seq=1 Ack=1712985 Win=1403136 Len=0 TSval=1498857 TSecr=…
1232	4.312262	172.16.50.1	90.130.70.73	TCP	78 [TCP Dup ACK 1215#7] 58189 → 23176 [ACK] Seq=1 Ack=1712985 Win=1403136 Len=0 TSval=1498857 TSecr=…
1233	4.312266	172.16.50.1	90.130.70.73	TCP	78 [TCP Dup ACK 1215#8] 58189 → 23176 [ACK] Seq=1 Ack=1712985 Win=1403136 Len=0 TSval=1498857 TSecr=…
1234	4.312271	172.16.50.1	90.130.70.73	TCP	78 [TCP Dup ACK 1215#9] 58189 → 23176 [ACK] Seq=1 Ack=1712985 Win=1403136 Len=0 TSval=1498857 TSecr=…
1235	4.312275	172.16.50.1	90.130.70.73	TCP	78 [TCP Dup ACK 1215#10] 58189 → 23176 [ACK] Seq=1 Ack=1712985 Win=1403136 Len=0 TSval=1498857 TSecr…
1236	4.312372	90.130.70.73	172.16.50.1	FTP-DA…	2962 FTP Data: 2896 bytes
1237	4.312393	172.16.50.1	90.130.70.73	TCP	78 [TCP Dup ACK 1215#11] 58189 → 23176 [ACK] Seq=1 Ack=1712985 Win=1403136 Len=0 TSval=1498857 TSecr…

Figura 26 - pacotes TCP no computador 1 quando transfere um ficheiro por FTP ao mesmo tempo que o computador 2

Anexo 3 – Código da aplicação de download

Download.h

```
#ifndef __DOWNLOAD_H
#define __DOWNLOAD_H

/*****************************************************
                    MACROS
 *****************************************************/

#ifndef FALSE
#define FALSE 0
#endif
#ifndef TRUE
#define TRUE  1
#endif

#define FTP_PORT    21
                                        //Default FTP port
#define FTP_MSG_SIZE   512
                                                //FTP Message buffer size
#define R1XX                 1
                                                //FTP Reply code hundreds
digit
#define R2XX                 2
                                                //FTP Reply code hundreds
digit
#define R3XX                 3
                                                //FTP Reply code hundreds
digit
#define R4XX                 4
                                                //FTP Reply code hundreds
digit
#define R5XX                 5
                                                //FTP Reply code hundreds
digit

#define FTP_USER_START 6
                                                //User argument start index

#define BUFFER_SIZE    256
#define FILEPATH_SIZE  1024

/*****************************************************
                  ENUMERATORS
 *****************************************************/

typedef enum {MULTILINE, NON_TERMINATED, REPLY_OVER,
REPLY_TOO_LONG} FTPLineType_t;   //FTP Reply types
typedef enum {WAIT_CR, WAIT_LF, OVER} FTPLineState_t;
                                    //FTP Line states
typedef enum {WAIT_REPLY, CONTINUE, REPEAT, ABORT}
FTPReplyState_t;                                //FTP Reply code states
```

```
/***********************************************************
                    STRUCTS
 ***********************************************************/

//Stores data socket descriptor and file path needed for download
typedef struct {

        char path[FILEPATH_SIZE];
        int datafd;
} FTPFile_t;

//Stores command line arguments passed
typedef struct {

        char user[BUFFER_SIZE];
        char password[BUFFER_SIZE];
        char host[BUFFER_SIZE];
        char path[FILEPATH_SIZE];
        uint8_t anonymous_f;
} FTPArgument_t;

/***********************************************************
                    FUNCTIONS
 ***********************************************************/

/**
 * Gets the hostname and IPv4 address of a given host
 *
 * @param address host string
 * @return pointer to hostent struct that stores host info
 */
struct hostent* getAddress(char* address);

/**
 * Gets a TCP socket to specified address at specified port
 *
 * @param address 32 bit Internet address network byte ordered
 * @param port port number
 * @return TCP socket file descriptor
 */
int getTCPSocket(in_addr_t address, uint16_t port);

/**
 * Gets a TCP socket to specified address at specified port
 *
 * @param address 32 bit Internet address network byte ordered
 * @param port port number
 * @return TCP socket file descriptor
 */
FTPLineType_t readFTPLine(int sockfd, char* reply, size_t reply_size);
```

```
/**
 * Parses final line of FTP reply message for reply code
 *
 * @param reply reply message text
 * @return FTP reply code
 */
uint16_t parseFTPReplyCode(char* reply);

/**
 * Aborts program by closing the TCP socket and printing error message
before exiting
 *
 * @param sockfd TCP socket file descriptor connected to FTP port
 * @param message error message to print
 */
void FTPAbort(int sockfd, char* message);

/**
 * Handles FTP reply code and returns action to take
 *
 * @param reply_code numeric value of the FTP reply code
 * @return enumerator specifying action caller should take
 */
FTPReplyState_t handleFTPReplyCode(uint16_t reply_code);

/**
 * Sends a command through the TCP connection, if command triggers
download calls appropriate function
 *
 * @param sockfd TCP socket file descriptor connected to FTP port
 * @param command FTP command to send
 * @param download_f whether command will trigger a download from FTP
server
 * @result 0 on success
 */
int8_t FTPCommand(int sockfd, char* command, uint8_t download_f);

/**
 * Performs FTP login command sequence
 *
 * @param sockfd TCP socket file descriptor connected to FTP port
 * @param user FTP USER command ready to send
 * @param password FTP PASS command ready to send
 * @return non 0 on failure
 */
int8_t FTPLogin(int sockfd, char* user, char* password);

/**
 * Reads full FTP message and parses the reply code
 *
 * @param sockfd TCP socket file descriptor connected to FTP port
 * @return FTP reply code
```

```
 */
uint16_t readFTPReply(int sockfd);

/**
 * Gets numeric representation of data port from FTP message containing it
 *
 * @param message FTP message containing data port
 * @return data port on success, -1 otherwise
 */
int32_t getDataPort(char* message);

/**
 * Sends PASV command and parses the data port received
 *
 * @param sockfd TCP socket file descriptor connected to FTP port
 * @return data port to use
 */
uint16_t FTPPassive(int sockfd);

/**
 * Parses command line argument and fills FTPArgument_t struct
 *
 * @param argument command line argument received
 * @return 0 on success
 */
int8_t parseArgument(char* argument);

/**
 * Parses path to file to get the file name, copies it to FTPFile_t struct
 *
 * @param path file path
 */
void parseFilePath(char* path);

/**
 * Uses FTP data socket connection to download file requested
 */
void FTPDownload();

#endif /*__DOWNLOAD_H */
```

Download.c

```
#include <stdio.h>
#include <stdlib.h>
#include <netdb.h>
#include <ctype.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
```

```
#include <unistd.h>
#include <signal.h>
#include <string.h>
#include "Download.h"

static char last_message[FTP_MSG_SIZE]; //Most recent message received
static FTPFile_t file_info;                    //Information about data
socket and local file path
static FTPArgument_t argument_info;        //Information needed for FTP
handling

int main(int argc, char* argv[]) {

        /* Check program parameter count */
        if(argc != 2) {

                printf("download: Wrong number of arguments.\n");
                printf("download: usage: ./download
ftp://[<user>:<password>@]<host>/<url-path>\n");
                exit(1);
        }

        /* Parse argument */
        if(parseArgument(argv[1]) != 0) {

                printf("download: Invalid argument.\n");
                printf("download: usage: ./download
ftp://[<user>:<password>@]<host>/<url-path>\n");
                exit(1);
        }

        /* Default values for anonymous user */
        if(argument_info.anonymous_f == TRUE) {

                strncpy(argument_info.user, "USER ", strlen("USER "));
                strcpy(argument_info.user + strlen("USER "),
"anonymous\r\n");

                strncpy(argument_info.password, "PASS ", strlen("PASS "));
                strcpy(argument_info.password + strlen("PASS "),
"ei11056@fe.up.pt\r\n");
        }

        /* Get host IP address */
        struct hostent* host;
        host = getAddress(argument_info.host);

        if(host == NULL) {
                printf("download: Couldn't get info on host provided.\n");
                exit(1);
        }
```

```
        printf("Host Name  : %s\n", host->h_name);
        printf("IP Address : %s\n\n", inet_ntoa(*((struct in_addr*)host->h_addr)));

        /* Get TCP socket to host */
        int sockfd = getTCPSocket(inet_addr(inet_ntoa(*((struct in_addr*)host->h_addr))), FTP_PORT);

        if(sockfd < 0) FTPAbort(sockfd, "download: Failure connecting to FTP server control port.\n");

        /* Login to FTP server */
        FTPLogin(sockfd, argument_info.user, argument_info.password);

        /* Use BINARY mode */
        if(FTPCommand(sockfd, "TYPE I\r\n", FALSE) != 0) FTPAbort(sockfd, "download: TYPE command received 5XX code.\n");

        /* Enter passive mode and get data port */
        uint16_t data_port = FTPPassive(sockfd);
        file_info.datafd = getTCPSocket(inet_addr(inet_ntoa(*((struct in_addr*)host->h_addr))), data_port);

        if(file_info.datafd < 0) FTPAbort(sockfd, "download: Failure connecting to FTP server data port.\n");

        /* Send command to download file and download it */
        if(FTPCommand(sockfd, argument_info.path, TRUE) != 0) FTPAbort(sockfd, "download: RETR command received 5XX code.\n");

    printf("\n");
        close(file_info.datafd);
        close(sockfd);

        return 0;
}

/**
 * Parses path to file to get the file name, copies it to FTPFile_t struct
 *
 * @param path file path
 */
void parseFilePath(char* path) {

        size_t path_i = 0;
        size_t last_slash_i = 0;

        while(path[path_i] != '\0') {

                if(path[path_i] == '/') {
                        last_slash_i = path_i;
                }
```

```
            path_i++;
        }

        file_info.path[0] = '.';
        strcpy(file_info.path + 1, path + last_slash_i);
}

/**
 * Parses command line argument and fills FTPArgument_t struct
 *
 * @param argument command line argument received
 * @return 0 on success
 */
int8_t parseArgument(char* argument) {

        /* Check that start of argument is as expected */
        if(strncmp(argument, "ftp://", FTP_USER_START) != 0) {
                printf("download: Could not find \"ftp://\" in argument.\n");
                return -1;
        }

        size_t arg_i = 0;
        size_t param_i = 0;
        uint8_t at_f = FALSE;
        uint8_t separator_f = FALSE;

        /* Search for @ symbol in argument */
        while(argument[arg_i] != '\0') {

                if(argument[arg_i] == '@') {
                        at_f = TRUE;
                        break;
                }

                arg_i++;
        }

        arg_i = FTP_USER_START;
        argument_info.anonymous_f = TRUE;

        /* Parse user and password if @ found */
        if(at_f) {

                argument_info.anonymous_f = FALSE;

                /* Parse user */
                strncpy(argument_info.user, "USER ", strlen("USER "));
                param_i += strlen("USER ");

                while(argument[arg_i] != '\0') {
```

```
                if(argument[arg_i] != ':') {
                        argument_info.user[param_i] = argument[arg_i];
                        arg_i++;
                        param_i++;
                } else {
                        arg_i++;
                        separator_f = TRUE;
                        break;
                }
        }

        if(!separator_f) {
                printf("download: Could not find \":\" separator in
argument.\n");
                return -1;
        } else if(param_i <= strlen("USER ")) {
                printf("download: Could not find user in argument.\n");
                return -1;
        }

        strcpy(argument_info.user + param_i, "\r\n");
        separator_f = FALSE;
        param_i = 0;

        /* Parse password */
        strncpy(argument_info.password, "PASS ", strlen("PASS "));
        param_i += strlen("PASS ");

        while(argument[arg_i] != '\0') {

                if(argument[arg_i] != '@') {
                        argument_info.password[param_i] =
argument[arg_i];
                        arg_i++;
                        param_i++;
                } else {
                        arg_i++;
                        separator_f = TRUE;
                        break;
                }
        }

        if(!separator_f) {
                printf("download: Could not find \"@\" separator in
argument.\n");
                return -1;
        } else if(param_i <= strlen("PASS ")) {
                printf("download: Could not find password in
argument.\n");
                return -1;
        }
```

```
            strcpy(argument_info.password + param_i, "\r\n");
            separator_f = FALSE;
            param_i = 0;
        }

        /* Parse host name */
        while(argument[arg_i] != '\0') {

            if(argument[arg_i] != '/') {
                argument_info.host[param_i] = argument[arg_i];
                arg_i++;
                param_i++;
            } else {
                separator_f = TRUE;
                break;
            }
        }

        if(!separator_f) {
            printf("download: Could not find \"/\" separator in argument
for host.\n");
            return -1;
        } else if(param_i <= 0) {
            printf("download: Could not find host in argument.\n");
            return -1;
        }

        argument_info.host[param_i] = '\0';
        separator_f = FALSE;
        param_i = 0;

        /* Parse file path */
        strncpy(argument_info.path, "RETR ", strlen("RETR "));
        param_i += strlen("RETR ");

        while(argument[arg_i] != '\0') {

            argument_info.path[param_i] = argument[arg_i];
            arg_i++;
            param_i++;
        }

        if(param_i <= 0) {
            printf("download: Could not find file path in argument.\n");
            return -1;
        }

        argument_info.path[param_i] = '\0';

        /* Extract file name from path */
        parseFilePath(argument_info.path);
```

```
        strcpy(argument_info.path + param_i, "\r\n");

        return 0;
}

/**
 * Uses FTP data socket connection to download file requested
 */
void FTPDownload() {

        /* Open file with file name parsed previously */
        FILE* fd = fopen(file_info.path, "wb");

        char file_data[FTP_MSG_SIZE];
        int n;

        /* Read until no more data is returned */
        while((n = read(file_info.datafd, file_data, FTP_MSG_SIZE)) > 0) {

                fwrite(file_data, 1, n, fd);
        }

        fclose(fd);
}

/**
 * Gets numeric representation of data port from FTP message containing it
 *
 * @param message FTP message containing data port
 * @return data port on success, -1 otherwise
 */
int32_t getDataPort(char* message) {

        size_t msg_i = 0;
        size_t number_i = 0;
        size_t comma_count = 0;
        char msb[6];
        char lsb[6];
        uint8_t success_f = FALSE;

        /* Process message up to null terminator */
        while(message[msg_i] != '\0') {

                /* Count commas up to data port values */
                if(message[msg_i] == ',') comma_count++;
                msg_i++;

                /* Next 2 number values will be the data port */
                if(comma_count >= 4) {

                        /* Read MSB of data port */
                        while(isdigit(message[msg_i])) {
```

```
                        msb[number_i] = message[msg_i];
                        number_i++;
                        msg_i++;
                }

                msb[number_i] = '\0';
                number_i = 0;
                msg_i++;

                /* Read LSB of data port */
                while(isdigit(message[msg_i])) {

                        lsb[number_i] = message[msg_i];
                        number_i++;
                        msg_i++;
                }

                lsb[number_i] = '\0';
                success_f = TRUE;

                break;
            }
        }

        if(!success_f) return -1;

        unsigned long msb_value = strtoul(msb, NULL, 10);
        unsigned long lsb_value = strtoul(lsb, NULL, 10);

        return (msb_value * 256 + lsb_value);
}

/**
 * Sends PASV command and parses the data port received
 *
 * @param sockfd TCP socket file descriptor connected to FTP port
 * @return data port to use
 */
uint16_t FTPPassive(int sockfd) {

        if(FTPCommand(sockfd, "PASV\r\n", FALSE) != 0) FTPAbort(sockfd,
"download: PASV command received 5XX code.\n");

        int32_t data_port = getDataPort(last_message);

        if(data_port < 0) FTPAbort(sockfd, "download: data port could not be
parsed.\n");

        return data_port;
}
```

```
/**
 * Aborts program by closing the TCP socket and printing error message
before exiting
 *
 * @param sockfd TCP socket file descriptor connected to FTP port
 * @param message error message to print
 */
void FTPAbort(int sockfd, char* message) {

	printf("%s", message);
	close(file_info.datafd);
	close(sockfd);
	exit(1);
}

/**
 * Handles FTP reply code and returns action to take
 *
 * @param reply_code numeric value of the FTP reply code
 * @return enumerator specifying action caller should take
 */
FTPReplyState_t handleFTPReplyCode(uint16_t reply_code) {

	uint16_t hundreds = (reply_code / 100) % 100;

	switch(hundreds) {

		case R1XX:
			return WAIT_REPLY;
		break;
		case R2XX:
			return CONTINUE;
		break;
		case R3XX:
			return CONTINUE;
		break;
		case R4XX:
			return REPEAT;
		break;
		case R5XX:
			return ABORT;
		break;
		default:
			return ABORT;
		break;
	}
}

/**
 * Sends a command through the TCP connection, if command triggers
download calls appropriate function
 *
```

```
 * @param sockfd TCP socket file descriptor connected to FTP port
 * @param command FTP command to send
 * @param download_f whether command will trigger a download from FTP
server
 * @result 0 on success
 */
int8_t FTPCommand(int sockfd, char* command, uint8_t download_f) {

	FTPReplyState_t reply_state;

	/* Repeat until reply code signals completion */
	do {

		/* Write command and read response */
		write(sockfd, command, strlen(command));
		reply_state = handleFTPReplyCode(readFTPReply(sockfd));

		/* Handle reply code */
		if(reply_state == ABORT) return -1;
		else if(reply_state == WAIT_REPLY) {

			if(download_f) FTPDownload();

			reply_state =
handleFTPReplyCode(readFTPReply(sockfd));
			if(reply_state == ABORT) return -1;
		}

	} while(reply_state != CONTINUE);

	return 0;
}

/**
 * Performs FTP login command sequence
 *
 * @param sockfd TCP socket file descriptor connected to FTP port
 * @param user FTP USER command ready to send
 * @param password FTP PASS command ready to send
 * @return non 0 on failure
 */
int8_t FTPLogin(int sockfd, char* user, char* password) {

	FTPReplyState_t reply_state;

	/* Read message of the day */
	reply_state = handleFTPReplyCode(readFTPReply(sockfd));

	if(reply_state != CONTINUE) FTPAbort(sockfd, "download: Server is
not ready.\n");

	/* Send USER command */
```

```
	if(FTPCommand(sockfd, user, FALSE) != 0) FTPAbort(sockfd,
"download: USER command received 5XX code.\n");

	/* Send PASS command */
	if(FTPCommand(sockfd, password, FALSE) != 0) FTPAbort(sockfd,
"download: PASS command received 5XX code.\n");

	return 0;
}

/**
 * Reads full FTP message and parses the reply code
 *
 * @param sockfd TCP socket file descriptor connected to FTP port
 * @return FTP reply code
 */
uint16_t readFTPReply(int sockfd) {

	FTPLineType_t status;
	char message[FTP_MSG_SIZE];

	do {
		status = readFTPLine(sockfd, message, sizeof(message));
		printf("%s", message);
	} while(status != REPLY_OVER);

	strcpy(last_message, message);

	return parseFTPReplyCode(message);
}

/**
 * Parses final line of FTP reply message for reply code
 *
 * @param reply reply message text
 * @return FTP reply code
 */
uint16_t parseFTPReplyCode(char* reply) {

	/* Copy first 3 values of reply message which should be the reply
code */
	char code[4];
	strncpy(code, reply, 3);
	code[3] = '\0';

	unsigned long number = strtoul(code, NULL, 10);

	return number;
}

/**
 * Reads a single FTP reply message line, waits for Telnet end-of-line
```

```
 *
 * @param sockfd TCP socket file descriptor connected to FTP port
 * @param reply reply message text filled in by the function
 * @param reply_size buffer size for reply message text
 * @return state of the reply message (finished, multiline, buffer overflow,
non terminated reply)
 */
FTPLineType_t readFTPLine(int sockfd, char* reply, size_t reply_size) {

	FTPLineState_t state = WAIT_CR;
	char receive[1];
	size_t counter = 0;

	/* Read 1 byte at a time until Telnet end-of-line (CRLF) */
	while(read(sockfd, receive, 1) > 0) {

		/* State machine */
		if(receive[0] == '\r') state = WAIT_LF;
		else if(receive[0] == '\n' && state == WAIT_LF) state = OVER;

		reply[counter] = receive[0];
		counter++;

		/* Check if reply is larger than buffer */
		if(counter >= reply_size - 1) {
			reply[counter] = '\0';
			return REPLY_TOO_LONG;
		}

		/* Check if reply message is over by looking for "xyz " string at
the start */
		if(state == OVER) {

			reply[counter] = '\0';

			if(isdigit(reply[0]) && isdigit(reply[1]) &&
isdigit(reply[2]) && reply[3] == ' ') return REPLY_OVER;
			else return MULTILINE;
		}
	}

	return NON_TERMINATED;
}

/**
 * Gets a TCP socket to specified address at specified port
 *
 * @param address 32 bit Internet address network byte ordered
 * @param port port number
 * @return TCP socket file descriptor
 */
int getTCPSocket(in_addr_t address, uint16_t port) {
```

```
        int     sockfd;
        struct sockaddr_in server_addr;

        /* Server address handling */
        bzero((char*) &server_addr, sizeof(server_addr));
        server_addr.sin_family = AF_INET;
        server_addr.sin_addr.s_addr = address;
        server_addr.sin_port = htons(port);

        /* Open TCP socket */
        if((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
                perror("socket()");
                return -1;
        }

        /* Connect to the server */
        if(connect(sockfd, (struct sockaddr*) &server_addr,
sizeof(server_addr)) < 0) {
                perror("connect()");
                return -1;
        }

        return sockfd;
}

/**
 * Gets the hostname and IPv4 address of a given host
 *
 * @param address host string
 * @return pointer to hostent struct that stores host info
 */
struct hostent* getAddress(char* address) {

        struct hostent* host;

        if((host = gethostbyname(address)) == NULL) {
                herror("download");
                return NULL;
        }

        return host;
}
```

Makefile

```
CC = gcc
CFLAGS = -Wall
SRCS = Download.c

all: $(SRCS)
        $(CC) $(CFLAGS) $(SRCS) -lm -o download
```