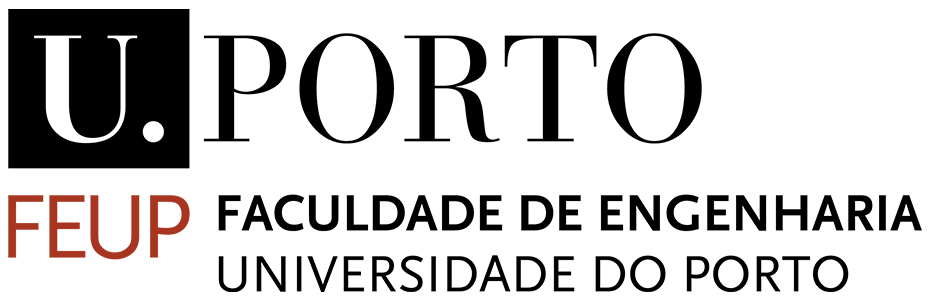
Configuração de uma rede e desenvolvimento de uma aplicação de download

**Relatório Final**

****

Mestrado Integrado Em Engenharia Informática e Computação

Redes de Computadores

**Professor:**

Sérgio Reis Cunha

**Turma 3, Bancada 4:**

Miguel Rodrigues Pires – up201406989

Diogo Henrique Pereira – up201505318

André Fernando Fernandes – up201505821

Faculdade de Engenharia da Universidade do Porto

Rua Roberto Frias, sn, 4200-465 Porto, Portugal

## Sumário

Este relatório incide sobre o segundo projeto da Unidade Curricular de RCOM (Redes de Computadores), do curso MIEIC. O projeto consiste na configuração de uma rede bem como no desenvolvimento de uma aplicação de *download*. O projeto divide-se, então, em duas fases de desenvolvimento, as quais são:

* Desenvolvimento da aplicação *download*, onde vai ser descrita a sua arquitetura e apresentados os resultados da sua execução;
* A configuração de uma rede, onde vão ser apresentados as seis experiências elaboradas durantes as aulas da UC.

As experiências acima referidas basearam-se na configuração de um IP de rede, da configuração de duas LAN’s (*Local Area Network*), as VLAN’s 0 e 1, virtuais no *switch*, onde os computadores se encontravam em série, e através do NAT (*Network Address Translation*), e ainda de um *router* em Linux, de um *router* comercial e do DNS (*Domain Name System*).

**Índice**

[Sumário1](#_Toc501509228)

[Índice2](#_Toc501509229)

[1. Introdução3](#_Toc501509230)

[2. Parte 1 – Aplicação de *download*3](#_Toc501509231)

[2.1 Arquitetura3](#_Toc501509232)

[2.2 Resultados de download8](#_Toc501509233)

[3. Parte 2 – Configuração da rede e análise9](#_Toc501509234)

[3.1 Experiência 1 – Configurar um IP de rede9](#_Toc501509235)

[3.2 Experiência 2 – Implementar duas redes *LAN* virtuais num *switch*10](#_Toc501509236)

[3.3 Experiência 3 – Configurar um *router* em *Linux*11](#_Toc501509237)

[3.4 Experiência 4 – Configurar um router comercial implementando o NAT12](#_Toc501509238)

[3.5 Experiência 5 – *DNS*13](#_Toc501509239)

[3.6 Experiência 6 – Ligações *TCP*14](#_Toc501509240)

[4. Conclusão15](#_Toc501509241)

[5. Referências15](#_Toc501509242)

[Anexos16](#_Toc501509243)

[Imagens relativas à Parte 2 do relatório16](#_Toc501509244)

[Código da aplicação de *download*30](#_Toc501509245)

## 1. Introdução

O segundo projeto de RCOM desenvolveu-se ao longo de diversas aulas laboratoriais, tendo a primeira aula servido de exemplo dos protocolos de IETF (*Internet Engineering Task Force*), tais como o *Telnet* e FTP. Estes protocolos promovem a solução de problemas relacionados com ligações á *Internet* usando documentos RFC que descrevem os protocolos. O protocolo que implementamos neste projeto foi o FTP usando o servidor da faculdade como, por exemplo, o tom.fe.up.pt. Aqui tivemos que criar uma aplicação de *download*, seguindo as instruções do guião, que procede à transferência de um ficheiro e que implemente o protocolo *FTP*.

Quanto à configuração de uma rede de computadores, o seu principal objetivo é executar uma aplicação, através de duas *VLAN’s*.

O relatório tem a seguinte estrutura:

* **Introdução**, onde é explicado o objetivo do projeto;
* **Aplicação de *download*,** onde é explicado o cliente FTP e a sua implementação;
* **Configuração de rede,** onde são explicados todos os passos feitos e todos os objetivos, experiência a experiência;
* **Conclusão,** onde é abordado o trabalho na sua totalidade, analisando o que podia ser feito, e a opinião geral do grupo;
* **Referências,** que consiste em todos os documentos e web*sites* utilizados pelo grupo;
* **Anexos,** que contêm o código, imagens, comandos corridos e *logs* gravados.

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

Como referido anteriormente, a primeira parte deste trabalho consiste numa aplicação que transfere um ficheiro utilizando o protocolo FTP descrito no ficheiro RFC959.

A aplicação desenvolvida permite que seja feito um *download* em modo anónimo ou com login de credenciais.

### 2.1 Arquitetura

A aplicação divide-se em três partes. Primeiramente processa a ***string*** fornecida pelo utilizador que contém o ***url***. Depois, processa-se o IP e a porta. A última parte diz respeito à conexão com o servidor **FTP**.

**Estruturas de dados:**

Foram usadas duas ***structs*** para guardar informação:

***Struct* FTP**

1. **typedef** **struct** FTP
2. {
3. **int** control\_socket\_fd; // file descriptor to control socket
4. **int** data\_socket\_fd; // file descriptor to data socket
5. } ftp;

e ***Struct* URL**.

1. **typedef** **char** url\_parameter[MAX\_STRING];
2. **typedef** **struct** URL {
3. url\_parameter username;
4. url\_parameter password;
5. url\_parameter hostname;
6. url\_parameter ip;
7. url\_parameter path;
8. url\_parameter filename;
9. **int** port;
10. }url;

A *struct* URL guarda informação extraída da *string* input do utilizador., através das funções **parseURL** e **getip**.

1. **int** parseURL(**char** \* url\_param, url\* url){
2. **int** reti;
3. **size\_t** nmatch = strlen(url\_param);
4. regmatch\_t pmatch[nmatch];
5. **char** \* tempURL = (**char** \* ) malloc(strlen(url\_param));
6. memcpy(tempURL, url\_param, strlen(url\_param));
7. regex\_t \* regexUser = (regex\_t\*) malloc(**sizeof**(regex\_t));
8. regex\_t  \* regexAnonymous = (regex\_t\*) malloc(**sizeof**(regex\_t));
9. **if** ((reti = regcomp(regexUser, regExpUser, REG\_EXTENDED)) != 0) {
10. printf("URL format is wrong: regExpUser");
11. **return** -1;
12. }
13. **if** ((reti = regcomp(regexAnonymous, regExpAnonymous, REG\_EXTENDED)) != 0) {
14. printf("URL format is wrong: regExpAnonymous");
15. **return** -1;
16. }
17. **if** ((reti = regexec(regexUser, tempURL, nmatch, pmatch, REG\_EXTENDED)) == 0) {
18. printf("URL is of type User\n");
19. parseUser(tempURL, url);
20. }
21. **else** **if** ((reti = regexec(regexAnonymous, tempURL, nmatch, pmatch, REG\_EXTENDED)) == 0) {
22. printf("URL is of type anonymous\n");
23. parseAnonymous(tempURL, url);
24. }
25. **else**{
26. printf("Error invalid URL!\n");
27. **return** -1;
28. }
30. free(regexUser);
31. free(regexAnonymous);
33. parseHost(tempURL, url);
34. parsePath(tempURL, url);
35. parseFilename(tempURL, url);
37. free(tempURL);
38. **return** 0;
39. }
40. **int** getip(url \* url){
41. **struct** hostent \*h;
43. **if** ((h=gethostbyname(url->hostname)) == NULL) {
44. herror("gethostbyname ERROR");
45. **return** -1 ;
46. }
48. strcpy(url->ip, inet\_ntoa(\*((**struct** in\_addr \*)h->h\_addr)));
49. url->port = PORT;
50. **return** 0;
51. }

A função parseURL valida o input, e extrai toda a informação possível da *string*. Depois, a função getip obtém o ip e a porta.

De seguida passa para a última parte da aplicação, que se destina ao contacto com o **servidor FTP**.

Primeiro conecta-se ao servidor FTP, com a função **connectftp**,

1. **int** connectftp(**const** **char** \* ip, **int** port){
2. **struct**  sockaddr\_in server\_addr;
3. **int** sock;
4. /\*server address handling\*/
5. bzero((**char**\*)&server\_addr,**sizeof**(server\_addr));
6. server\_addr.sin\_family = AF\_INET;
7. server\_addr.sin\_addr.s\_addr = inet\_addr(ip);    /\*32 bit Internet address network byte ordered\*/
8. server\_addr.sin\_port = htons(port);     /\*server TCP port must be network byte ordered \*/
10. /\*open an TCP socket\*/
11. **if** ((sock= socket(AF\_INET,SOCK\_STREAM,0)) < 0) {
12. perror("socket() ERROR");
13. **return** -1;
14. }
16. /\*connect to the server\*/
17. **if**(connect(sock,(**struct** sockaddr \*)&server\_addr, **sizeof**(server\_addr)) < 0){
18. perror("connect() ERROR");
19. **return** -1;
20. }
22. printf("Connected!\n");
23. **return** sock;
24. }

que retorna a ***socket*** de controlo de conexão. Depois dá-se o login, com a função(loginftp), usando o comando “USER \_username\_” e “PASS \_password\_”.

Seguidamente muda-se de diretório para o diretório desejado, usando o comando “CWD \_directory\_” na função **changedirectoryftp**,

1. **int** changedirectoryftp(ftp \* ftp, **const** **char** \* path){
2. **int** sendPath;
3. **char** cwd[1024];
4. sprintf(cwd, "CWD %s\r\n", path);
6. **if**((sendPath = sendftp(ftp, cwd)) == -1)
7. **return** -1;
9. **else** printf("Path was sent\n");
11. memset(cwd, 0 ,**sizeof**(cwd));
12. **int** receiveCWDResponse;
13. **if**((receiveCWDResponse = receiveftp(ftp, cwd, **sizeof**(cwd))) == -1)
14. **return** -1;
16. printf("Changed directory!\n");
17. **return** 0;
18. }

seguindo-se a entrada em modo passivo com a função **passiveMode**.

1. **int** passiveMode(ftp \* ftp){
2. **char** pasv[1024] = "PASV\r\n";
3. **int** sendPassive = sendftp(ftp, pasv);
4. **if** (sendPassive == -1)
5. **return** -1;
7. sendPassive = receiveftp(ftp, pasv, **sizeof**(pasv));
8. **if** (sendPassive == -1)
9. **return** -1;
11. **int** ip1, ip2, ip3, ip4, p1, p2;
12. **if** ((sscanf(pasv, "227 Entering Passive Mode (%d,%d,%d,%d,%d,%d)", &ip1,
13. &ip2, &ip3, &ip4, &p1, &p2)) < 0) {
14. printf("ERROR parsing information from Passive Mode response.\n");
15. **return** -1;
16. }
18. memset(pasv, 0, **sizeof**(pasv));
20. **if** ((sprintf(pasv, "%d.%d.%d.%d", ip1, ip2, ip3, ip4))
21. < 0) {
22. printf("ERROR forming ip address\n");
23. **return** -1;
24. }
26. **int** portResult = p1 \* 256 + p2;
28. // printf("IP: %s\n", pasv);
29. // printf("port1:%d\n", p1);
30. // printf("port2:%d\n", p2);
31. // printf("PORT: %d\n", portResult);
33. ftp->data\_socket\_fd = connectftp(pasv, portResult);
34. **if** (ftp->data\_socket\_fd == -1)
35. **return** -1;
37. printf("Entered passive mode!\n");
39. **return** 0;
40. }

Esta função envia o comando “PASV” e recebendo como resposta, um IP e uma porta, que irão ser usados para criar a socket de transferência de dados. Esta e a outra *socket* são guardadas na struct FTP.

Prepara-se de seguida o ***download*** do ficheiro usando o comando “RETR \_filename\_” na função **retrftp**

1. **int** retrftp(ftp\* ftp, **const** **char**\* filename) {
2. **char** retr[1024];
3. sprintf(retr, "RETR %s\r\n", filename);
4. **if** (sendftp(ftp, retr))
5. **return** -1;
7. **if** (receiveftp(ftp, retr, **sizeof**(retr)))
8. **return** -1;
10. printf("Preparing download of %s\n", filename);
11. **return** 0;
12. }

e faz-se o download com a função **downloadftp**.

1. **int** downloadftp(ftp\* ftp, **const** **char** \* filename){
2. **FILE** \* file = fopen(filename, "w");
3. **if**(file == NULL)
4. {
5. printf("Could not create local file: %s\n", filename);
6. **return** -1;
7. }
9. **char** buffer[1024];
10. **int** bytes = 1;
11. printf("Downloading file: ");
12. **while**(bytes > 0){
13. printf("...");
14. bytes = recv(ftp->data\_socket\_fd, buffer, 1024, 0);
15. **int** fwriteStatus = fwrite(buffer, bytes, 1, file);
16. **if**(fwriteStatus < 0)
17. {
18. printf("Error on fwrite!\n");
19. **return** -1;
20. }
21. }
22. printf("\n");
23. fclose(file);
25. **char** response[1024];
26. **int** receiveUserResponse;
27. **if**((receiveUserResponse = receiveftp(ftp, response, **sizeof**(response))) == -1)
28. **return** -1;
29. **return** 0;
30. }

No fim faz-se a desconexão usando a função **disconnectftp**.

1. **int** disconnectftp(ftp \* ftp){
2. **int** sendQuit;
3. **char** response[1024];
4. sprintf(response, "QUIT\r\n");
6. **if**((sendQuit = sendftp(ftp, response)) == -1)
7. **return** -1;
9. close(ftp->data\_socket\_fd);
10. close(ftp->control\_socket\_fd);
11. printf("QUIT\n");
12. **return** 0;
13. }

### 2.2 Resultados de download

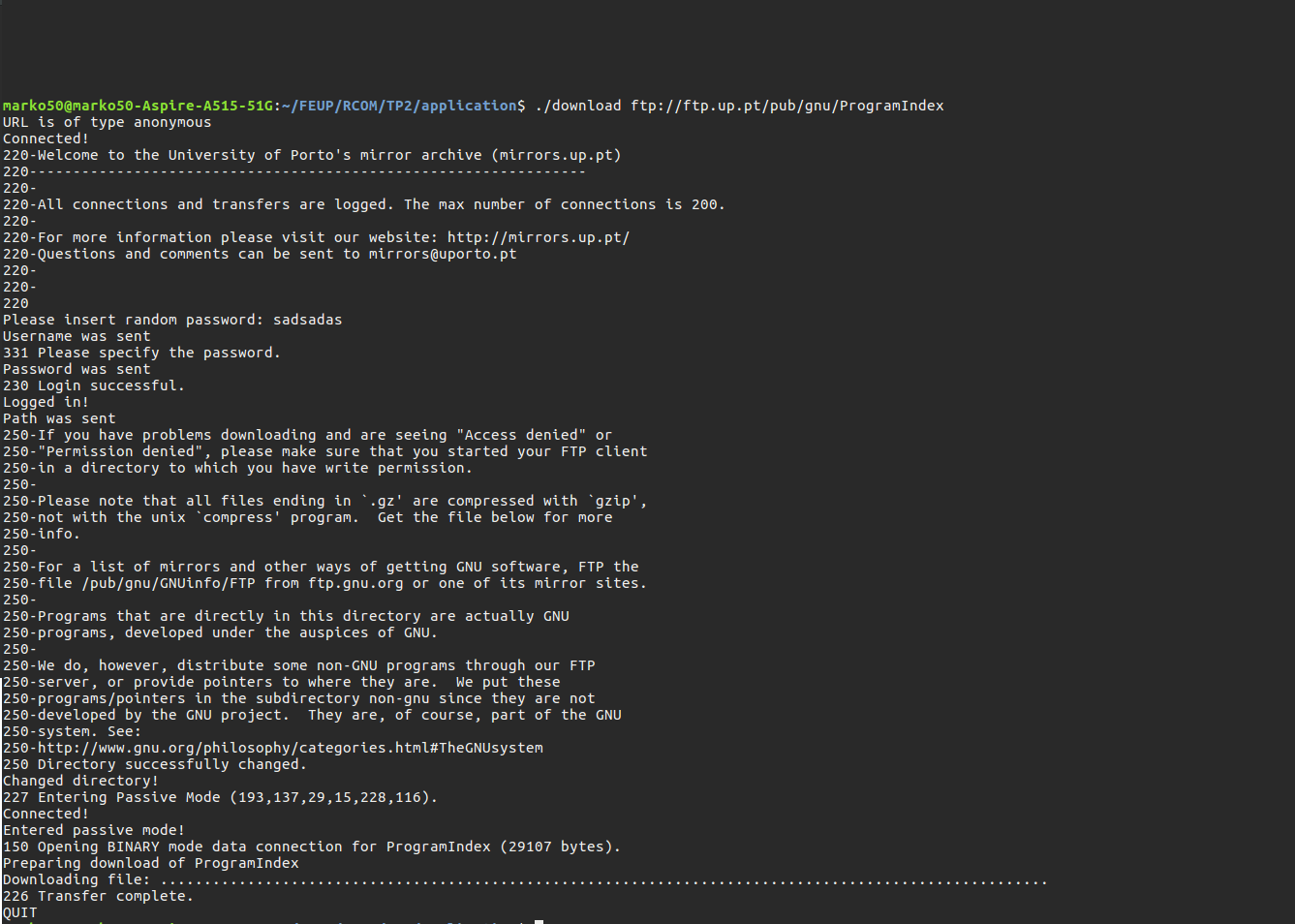
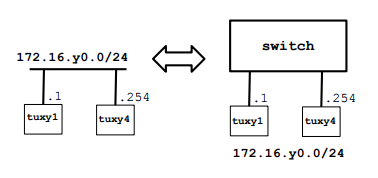
**** Este é um exemplo da execução da aplicação para <ftp://ftp.up.pt/pub/gnu/ProgramIndex> , transferindo um ficheiro de 24Kb.

Figura 1 - Transferência de um ficheiro

## 3. Parte 2 – Configuração da rede e análise

### 3.1 Experiência 1 – Configurar um IP de rede



Esta primeira experiência teve como finalidade compreendermos a configuração de **IP’s** em máquinas diferentes, de modo a que estas consigam comunicar entre si.

Para a configuração dos dois computadores **tux41** e **tux44** foi utilizado o comando ***<ifconfig ‘ipaddress’ eth0>***, que atribui ao IP da interface o IP passado como argumento e ativa as portas eth0 às quais foram ligados os cabos de rede. Neste caso o tux41 assume o IP de 172.16.40.1 e o tux44 assume 172.16.40.254. Como os computadores se encontram na mesma rede, não foi necessário usar o comando ***route***. Também foi necessário configurarmos a Vlan 40 para as portas 1 e 2, usando os comandos ***<interface fastethernet 0/nº porta>***, ***<switchport mode access>*** e ***<switchport access vlan 40>***.

Após esta configuração, executamos um comando ***ping*** de uma máquina para a outra com os IP’s definidos, no qual verificamos que existia ligação entre as duas. De seguida inspecionamos as tabelas **ARP** com o comando ***<arp –a>***, e depois apagamos todas as entradas na tabela ARP do tux41 através do comando ***<arp –d 172.16.40.254>***.

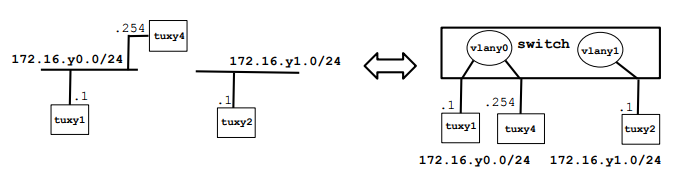
Por fim repetiu-se o comando *ping* registando o processo através do programa *wireshark*. Analisando o *log* do *wireshark* (ver figura nos Anexos), podemos verificar que, tendo apagado as entradas da tabela ARP é perguntado à rede qual o endereço **MAC** com um endereço **IP** igual a 172.16.40.254. Este computador responde com o seu endereço MAC e, a partir de aí, sempre que o primeiro faz um *request* **ICMP**, este é seguido de uma resposta do segundo.

Um endereço ip dos pacotes refere-se ao endereço ip dos computadores na rede, o endereço MAC, é um endereço físico da interface de comunicação que conecta o computador cada computador à rede.

Para descobrir o tamanho de um pacote recebido, é contar os bytes desde a flag inicial do header até ao end of frame.

A interface *loopback* é uma interface de rede virtual que o computador utiliza para comunicar com ele próprio, com o objetivo de realizar testes de diagnóstico, ou aceder a servidores na própria máquina, como se fosse um cliente. Assim, uma interface *loopback*, permite a existência de um endereço IP no *router*, que está sempre ativo, em vez de ser dependente de uma interface física.

### 3.2 Experiência 2 – Implementar duas redes *LAN* virtuais num *switch*



Esta experiência consiste na criação de duas *VLAN’s* diferentes:

* *VLAN 40 – 172.16.40.0/24* – é constituída pelos computadores **tux41** e **tux44**;
* *VLAN 41 – 172.16.41.0/24* – é constituída pelo **tux42**.

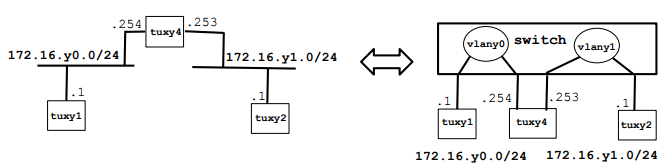
No final desta configuração, o computador **tux42** deixará de ter acesso aos computadores tux41e tux44, uma vez que se encontram em sub-redes diferentes. Para configurar os computadores da forma referida basta proceder do mesmo modo que se procedeu para a primeira experiência, mas desta vez fazê-lo no computador tux42 e atribuído **172.16.41.1** como endereço de **IP** (tendo em conta que os outros dois computadores continuam com a configuração da primeira experiência).

Para a configuração do ***switch*** foi necessário entrar na sua consola de configuração, através da aplicação ***GkTerm***, e executar o comando ***<vlan [n]>***, onde ***n*** é o número identificador da VLAN. Após a configuração das VLAN’s foi necessário adicionar as portas do *switch* às respetivas VLAN’s, para criar duas sub-redes individuais. Para isso usaram-se os comandos ***<interface fastethernet 0/[i]>***, onde ***i*** é o identificador da porta do *switch*, seguido de ***<switchport mode access>*** e ***<switchport access VLAN [n]>***, onde ***n*** é o identificador da VLAN criada. Os cabos ligados às portas do *switch* estão ligados à interface 0 dos computadores.

Estando os dois computadores na mesma rede, ao fazer ***ping*** do tux41 para o tux44, é enviado o pacote **ARP** para saber o endereço **MAC**. Executando um *ping* para o computador tux42, esse *ping* falhou como era esperado, uma vez que se encontrava numa sub-rede diferente, inacessível tanto pelo computador tux41 como pelo tux44.

Pode também verificar-se que cada VLAN tem um ***broadcast domain*** diferente, tendo em conta que o tux42 não detectou o pacote enviado, como tal, conclui-se que a configuração destas redes foi feita da forma correta pois verifica-se a falta de comunicação entre as redes, como referimos inicialmente. Logo, existem dois *broadcast domains*, em qualquer um dos computadores consegue-se dar ping ao respectivo *broadcast domain,* mas não se obtém um IP de resposta.

### 3.3 Experiência 3 – Configurar um *router* em *Linux*



Esta experiência consiste na configuração do computador **tux44** como ***router*** de forma a ligar as duas *VLAN’s* existentes criadas na experiência anterior.

Para isso foi necessário ativar a interface *ethernet* 1 do computador tux44 e configurá-la com um IP dentro da mesma gama que o computador tux42, e adicionar esta interface à sub-rede do tux42. Esta porta é a que será ligada á *vlan 41*. Configuramos a porta com o endereço IP **172.16.41.253/24**. Sendo este computador aquele que pretendemos que sirva como *router*, é necessário ativar o reencaminhamento de IP’s através do comando ***<echo 1 > /proc/sys/net/ipv4/ip\_forward>***.

Por fim, adicionou-se as rotas necessárias no tux41 e no tux42 de forma a que estes, através do tux44, pudessem aceder à rede a que não pertencem. Para o tux41 usamos o comando ***<route add -net 172.16.41.0/24 gw 172.16.40.254>***. O primeiro endereço indica a gama de endereços para a qual se quer adicionar a rota e o segundo endereço identifica o IP para o qual se deve reencaminhar o pacote, neste caso o IP do tux44. Repetiu-se o mesmo procedimento para o tux42, mas usando os endereços ***<route add -net 172.16.40.0/24 gw 172.16.41.253>***.

Finalmente, foi possível *pingar* o tux42 a partir do tux41. O pedido para o IP do tux42 (172.16.41.1) é reencaminhado para o tux44 (172.16.40.254). Como o tux44 está ligado à sub-rede de ambos os computadores, consegue aceder ao tux42 através da sua interface eth1, que está nessa sub-rede, e assim reencaminha o pacote para o tux42. Na resposta o processo é idêntico, sendo o pacote reencaminhado do tux42 para o tux41.

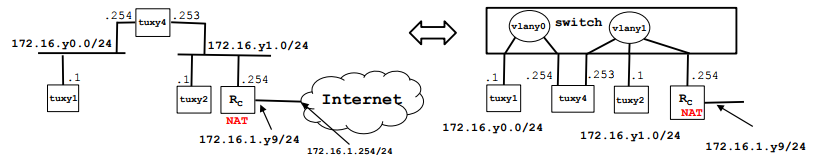
No tux41 são visíveis a ***default route*** da rede em que está e também uma rota para a sub-rede 172.16.41.0/24 pelo tux44 (172.16.40.254), no tux42 verifica-se também uma *default route* para a sub-rede onde esta inserido e uma rota para 172.16.40.0/24 pelo tux44 (172.16.41.253). Finalmente no tux44 verifica-se que existem duas rotas *default* para as duas sub-redes.

Uma entrada na ***routing table*** contém informação sobre rotas para uma determinada rede destinatária, tal como a ***gateway*** a utilizar, a interface e a máscara.

Quando o tux41 dá *ping* ao tux42, observam-se pacotes ARP, nos 3 computadores que indicam endereços **MAC** de vários computadores para se poder enviar informação na ida e na vinda. Para o tux41 comunicar com o tux42, é preciso saber o seu endereço MAC, e antes disso o endereço MAC da interface eth0 do tux44. O tux44 responde e o pacote é reencaminhado para lá, e depois tem de enviar pela interface eth1, que já sabe o seu endereço NAT. A interface eth1 depois tem de reencaminhar para o tux42, mas precisa do seu endereço NAT, que aparece num pacote ARP, construído, tal como todos após um ***ping broadcast***, e o pacote chega ao tux42. Depois é preciso o tux42 comunicar com o tux41 seguindo uma lógica semelhante.

Os pacotes **IMCP** aparecem porque na tabela de rotas existem rotas por *gateways* específicas para enviar pacotes. O destinatário é o IP do tux42 e o emissor é o IP do tux41.

### 3.4 Experiência 4 – Configurar um router comercial implementando o NAT



Esta experiência tem como objetivo a configuração de um router comercial dentro da rede 41 de forma a que computadores tanto na *VLAN 40* como na *VLAN 41* tenham acesso à *Internet*. A implementação do **NAT (*Network Address Translation*)** tem como objetivo possibilitar a comunicação entre os computadores da rede criada com redes externas. Por se tratar de uma rede privada, os IP’s nunca seriam reconhecidos fora da rede. Por isso criou-se um método que permite reescrever os IP’s de origem de uma rede interna, para que possam aceder a uma rede externa. Este procedimento gera um número de 16 bits, utilizando esse valor numa ***hash table***, e escrevendo-o no campo da porta de origem. Na resposta, o processo é revertido e o *router* sabe para qual computador da rede interna deve enviar a resposta.

Para configurar o *router*, depois de fazer *login*, foi necessário configurar a interface interna no processo de NAT. Para isso entramos na consola de configuração da interface *fastethernet 0/0* do *router*, com o comando ***<interface fastethernet 0/0>***. De seguida teve de ser especificado qual o IP para essa interface, introduzindo o comando ***<ip address 172.16.41.254 255.255.255.0>***. Posteriormente, foi configurada a interface externa, atribuindo um IP à interface 1, que estava ligada ao *router* da sala, introduzindo os seguintes comandos ***<interface fastethernet 0/1>***, ***<ip address 172.16.2.49 255.255.255.0>***. Para ambos os casos foi necessário introduzir o comando ***<no shutdown>***, para que estas configurações se mantivessem caso o *router* fosse desligado.

De seguida para que fosse garantida a gama de endereços, introduziram-se os comandos ***<ip nat pool ovrld 172.16.2.49 172.16.2.49 prefix 24>*** e ***<ip nat inside source list 1 pool ovrld overload>***. Foi criada também uma lista de acessos e permissões de pacotes, para cada uma das sub-redes, com o comando ***<access-list 1 permit ip [máximo]>***. Neste caso, o IP utilizado foi 172.16.40.0 e 172.16.41.0, que poderia ir até 172.16.4X.255, colocando 0.0.0.255 no campo máximo.

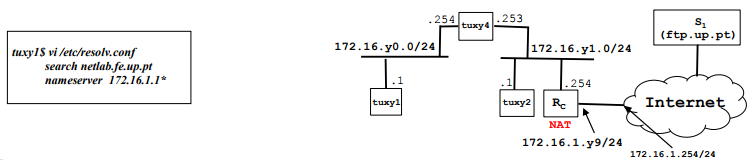
Depois de permitir que os computadores das redes criadas tenham acesso, é configurada uma rota pré-definida para o endereço *Internet*. Da mesma forma é adicionada aos computadores a rota pré-definida para o endereço 172.16.41.254, endereço do *router* comercial. Definimos rotas internas e externas, usando os comandos ***<ip route 0.0.0.0 0.0.0.0 172.16.2.254>*** e ***<ip route 172.16.40.0 255.255.255.0 172.16.41.253>***. Isto cria uma nova rota, quando o IP de destino for 172.16.40.0-255 deve redirecionar os pacotes para o IP 172.16.41.253.

Para testarmos fizemos um *ping* no tux41 para o *router* da sala e verificou-se que os pacotes enviados para este computador passavam pelo tux42, onde eram reencaminhados para o *router* no IP 172.16.41.254.

Quando se dá *ping* do tux41 para um endereço IP, por exemplo 8.8.8.8, que é o caso mais complexo na rede criada, este tem uma rota para enviar pacotes cujo destinatário é deste género para a *gateway* da interface eth0 do tux44. Por sua vez o tux44, como reconhece o IP do router, envia os pacotes para lá. O router, para responder, tem uma rota, que quando quer enviar pacotes para a sub-rede do tux41, envia pela interface eth1 do tux44.

NAT reescreve os endereços IP privados de uma rede local, como públicos de modo a que os computadores desta rede tenham acesso à rede mundial de computadores.

### 3.5 Experiência 5 – *DNS*



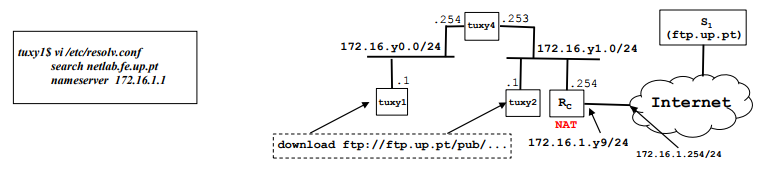
O objetivo desta experiência era conseguir aceder a redes externas, conseguindo desta forma aceder à Internet, através da rede interna criada. Para isto, foi apenas necessário configurar o **DNS**.

Esta configuração passa por, em todos os *hosts* da rede criada, aceder e editar o ficheiro **resolv.conf**, situado no diretório **/etc** nos computadores com o sistema operativo em *Linux*. Este ficheiro é lido cada vez que são invocadas rotinas que fornecem acesso à *Internet*. Neste caso, o ficheiro foi editado colocando as linhas:

* ***search services.netlab.fe.up.pt***
* ***nameserver 172.16.2.1***

Por fim foi testada a funcionalidade desta alteração fazendo o teste de *ping* usando *www.google.com*. Nos *logs*, consequentemente, verificou-se que o DNS pergunta a informação contida num dado *domain name*, e este responde com o tempo de vida e o tamanho do pacote de dados.

### 3.6 Experiência 6 – Ligações *TCP*



Por fim, a experiência 6 tinha como objetivo juntar as duas partes deste trabalho, pondo em prática a **aplicação de *download****,* desenvolvida na Parte 1, na rede configurada ao longo destas experiências.

Para testar a aplicação, foi usado um servidor **FTP** e efetuado o *download* de um ficheiro, em modo anónimo e não anónimo. O *download* efetuou-se corretamente, o que demonstrou que a rede estava bem configurada, não trazendo qualquer problema no acesso por protocolo FTP, assim como à utilização de um servidor exterior à rede.

**TCP** utiliza ***Selective Repeat ARQ***, que é semelhante ao *GO-BACK-N ARQ*, com a diferença que o recetor não deixa de processar os **frames** recebidos quando deteta um erro. Quando a falha de um frame é detetada, o recetor continua a receber e a processar as frames seguintes, enviando sempre, no ***ack***, o número da frame que falhou primeiro. No final do envio, o emissor verifica os *ack* e reenvia os frames perdidos.

A aplicação usa duas conexões FTP, uma para comunicar com o servidor e outra para transferência de dados. A informação de controlo FTP é transportada na conexão de controlo com o servidor. Uma conexão FTP é iniciada com a conexão a um determinado IP numa determinada porta. Essa conexão é feita criando uma ***socket*** de controlo. Depois, dependendo do servidor, dá-se o login no servidor usando a *socket* de controlo. Entra-se em modo passivo, criando a *socket* de transferência de dados, transfere-se os dados a partir dessa *socket* e no fim, desconexão.

Em TCP, o emissor mantém uma janela de congestão que possui o número de *bytes* máximo(varia) que o emissor pode enviar ao recetor.

A abertura de uma nova conexão TCP para transferência de dados leva a uma queda na taxa de transmissão.

## 4. Conclusão

Após a conclusão do segundo projeto de Redes e Computadores, o grupo conseguiu captar quase todos os conceitos necessários para uma implementação estável e coerente daquilo que era pedido no guião.

A configuração de rede foi concluída com sucesso permitindo a todos os elementos do grupo ter uma noção de como tudo funciona e assim aplicar esta prática a nível profissional. A aplicação de download implicava um prévio estudo de alguns protocolos como os que são descritos nos RFC959 e RFC1738 sobre FTP e sintaxe do URL, respetivamente.

Todos os objetivos delineados pelo grupo foram atingidos com sucesso, com exceção da experiência 7, de cariz opcional, que não implementamos para a apresentação ao docente.

## 5. Referências

* Manuel Ricardo *Lab 2 – Computer Networks*, 2014 [link](https://moodle.up.pt/pluginfile.php/32668/mod_resource/content/7/lab2.pdf)
* Andrew Tanenbaum, David Wetherall *Computer Networks (5th Edition)*, 2010 [link](https://github.com/gsahinpi/acm361/blob/master/Computer%20Networks%20-%20A%20Tanenbaum%20-%205th%20edition.pdf)

## Anexos

### Imagens relativas à Parte 2 do relatório

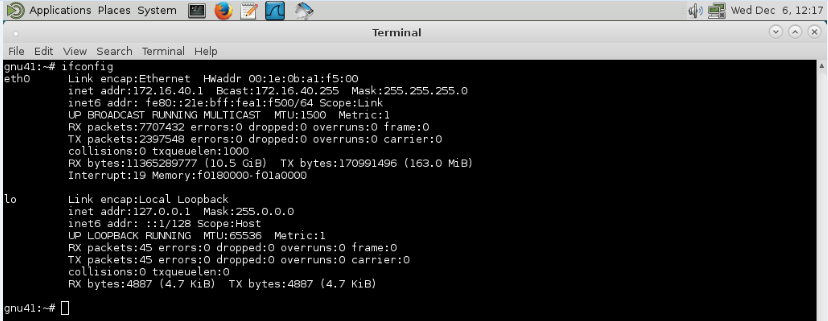


Figura 2 – Exp1: Configuração do pc1

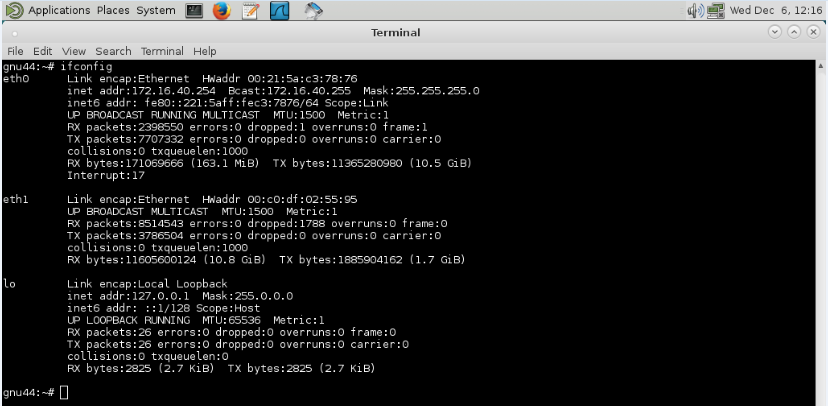


Figura 3 - Exp1: Configuração do pc4

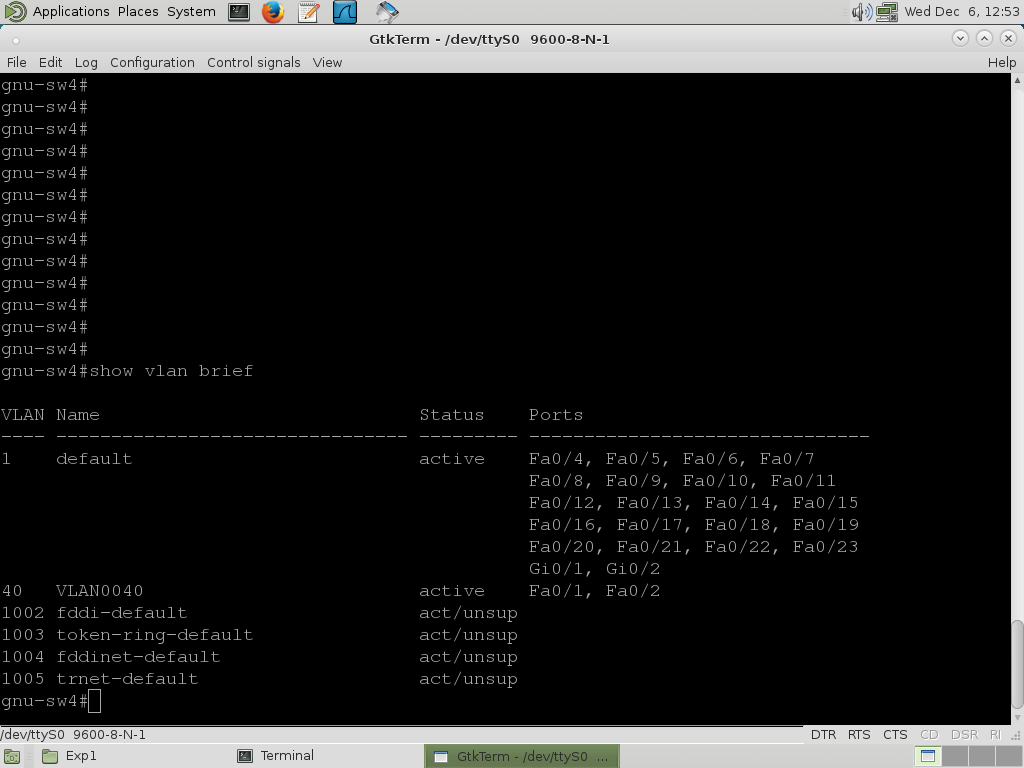


Figura 4 – Exp1: Configuração da VLAN 40 para as portas 1 e 2

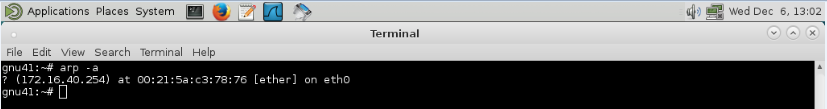


Figura 5 - Exp1: Inspeção das tabelas ARP

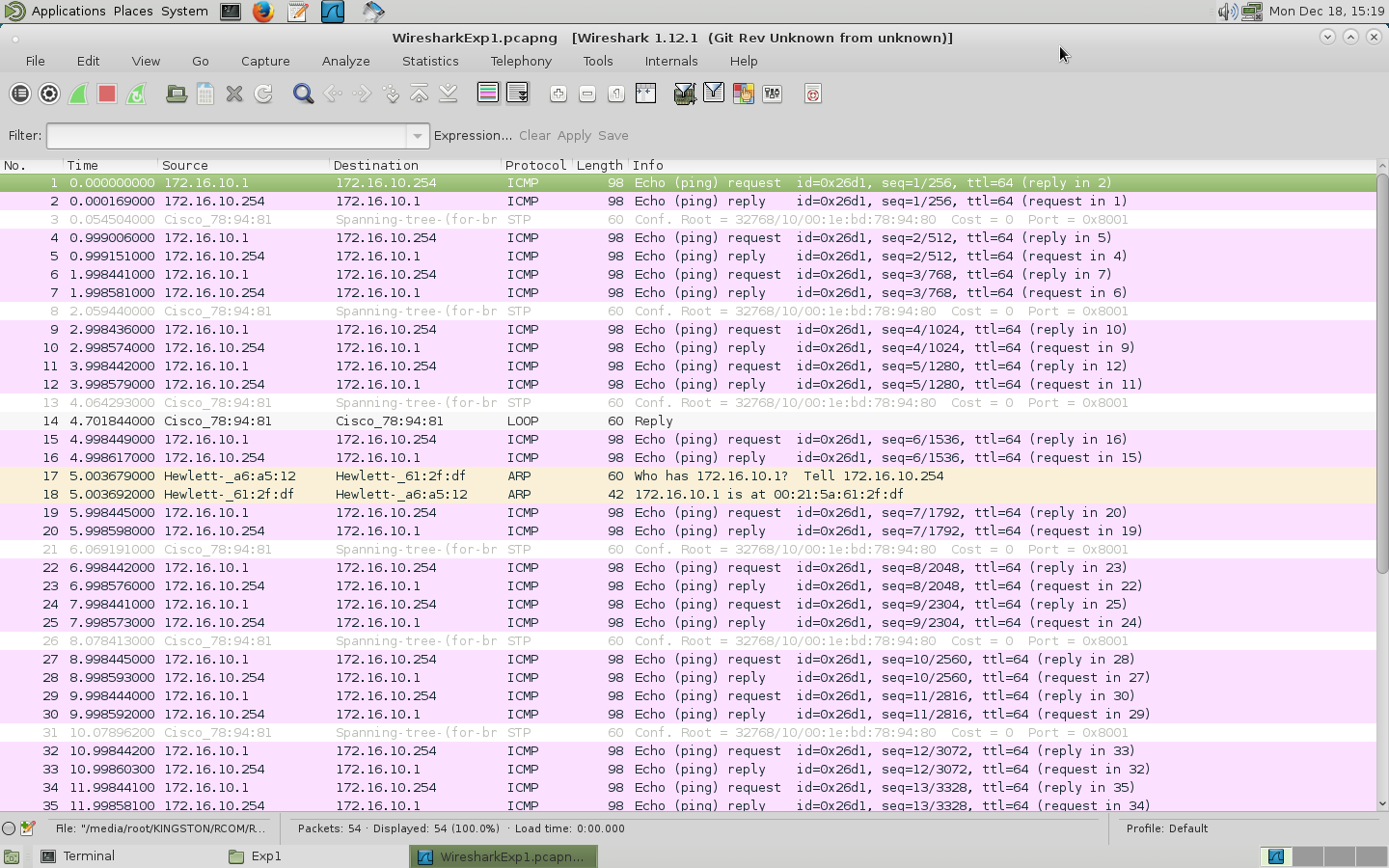


Figura 6 - Exp1: Wireshark log

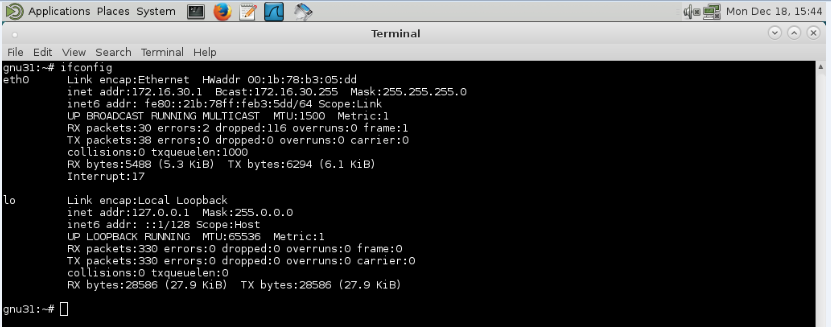


Figura 7 - Exp2: Configuração do pc1

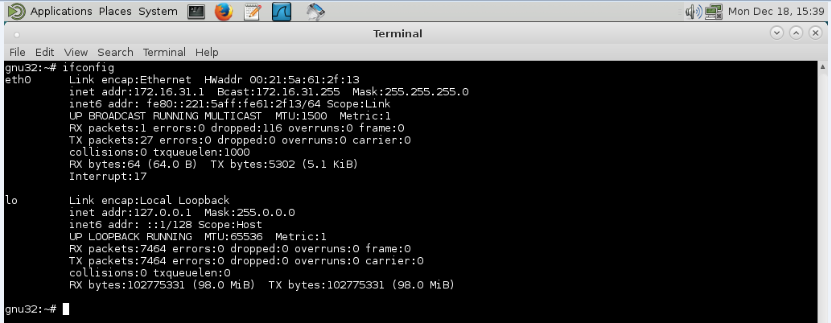


Figura 8 - Exp2: Configuração do pc2

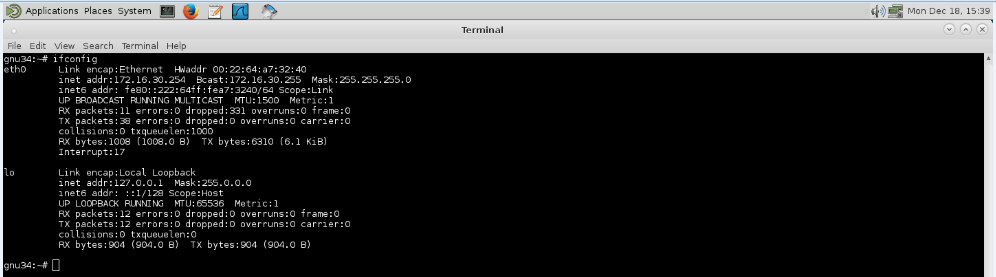


Figura 9 - Exp2: Configuração do pc4

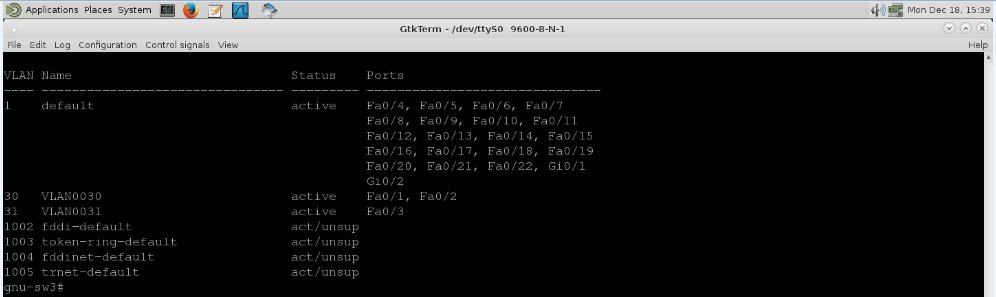


Figura 10 - Exp2: Configuração das VLANs

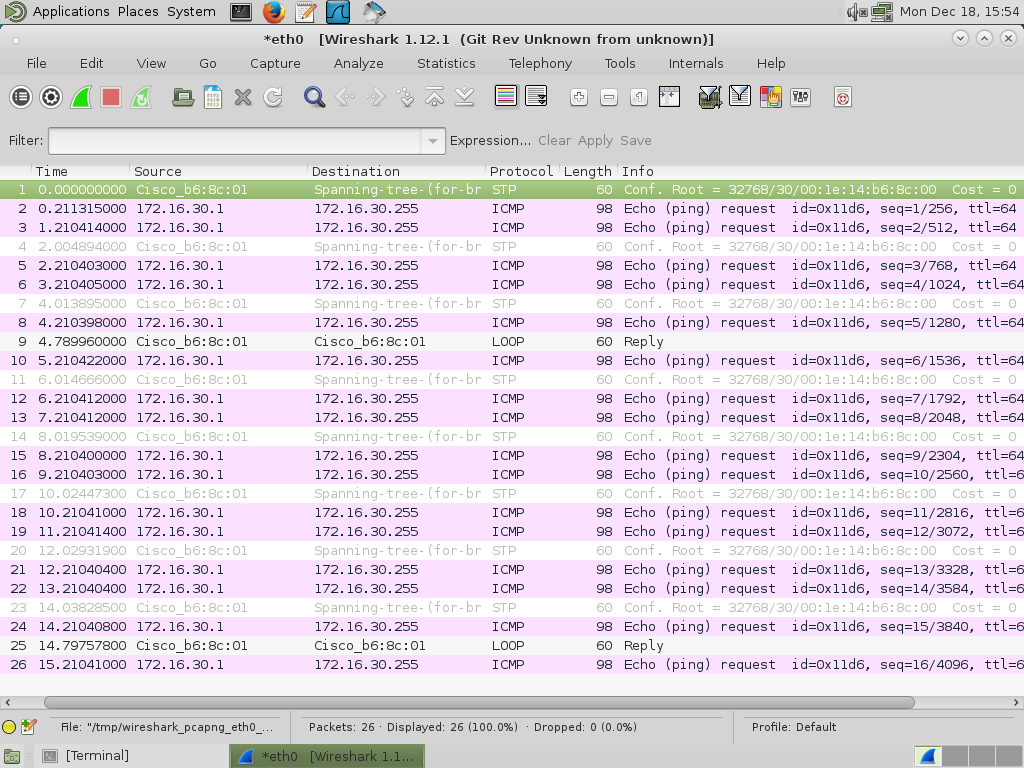


Figura 11 - Exp2: Wireshark ping broadcast

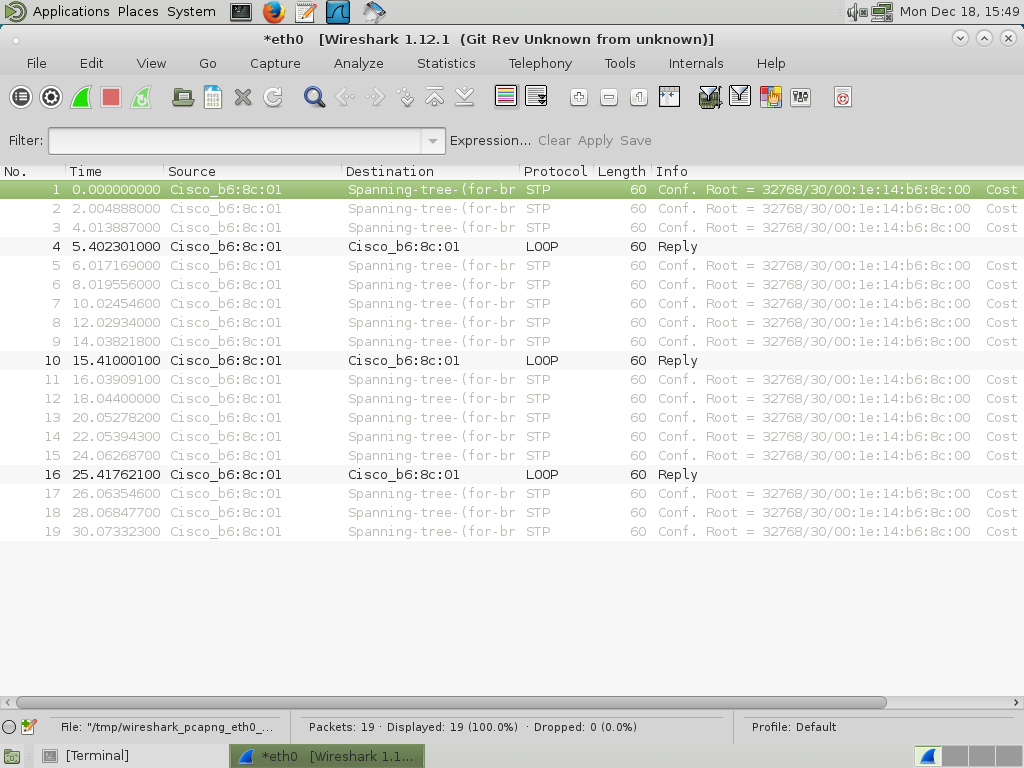


Figura 12 – Exp2: Wireshark ping pc2

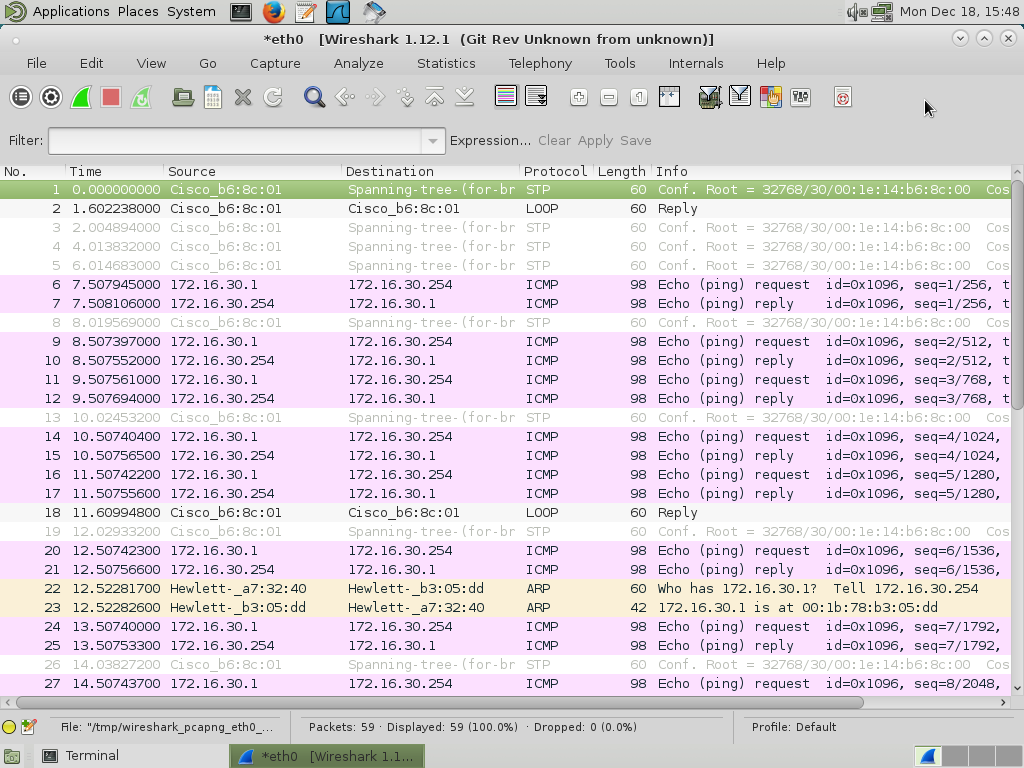


Figura 13 - Exp2: Wireshark ping pc4

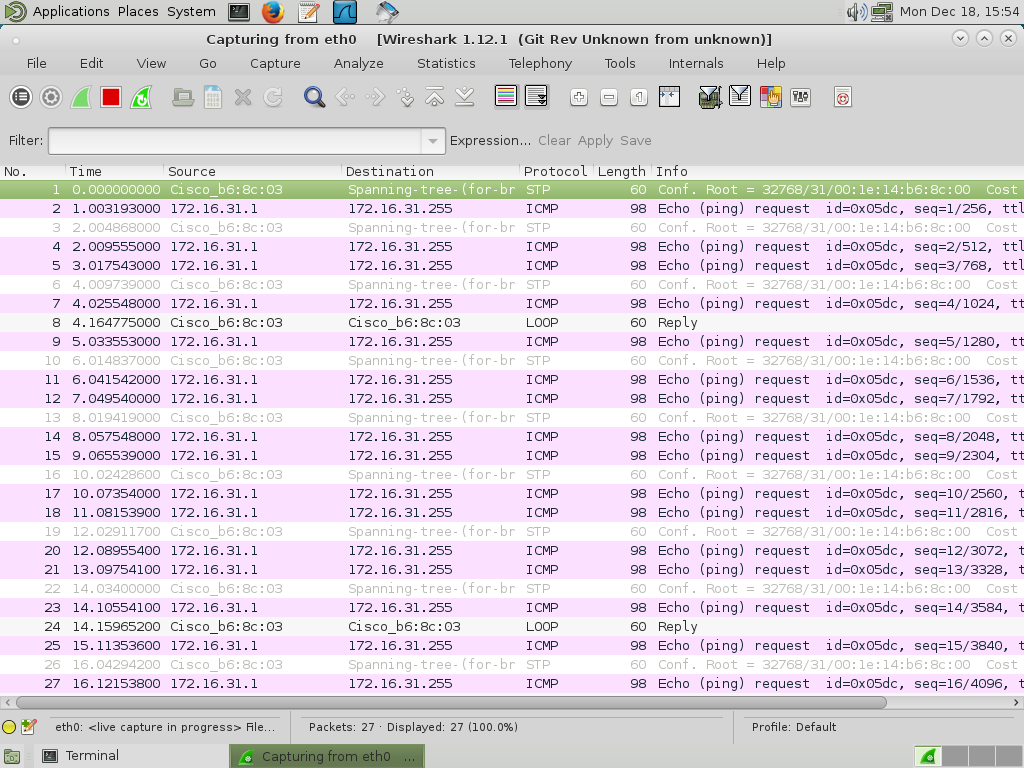


Figura 14 - Exp2: Wireshark pc2 ping broadcast

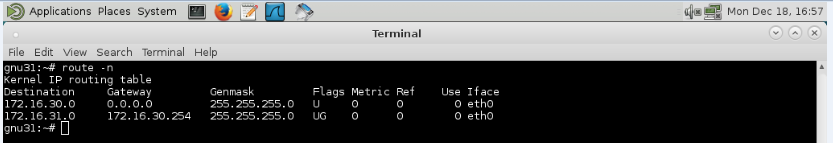


Figura 15 - Exp3: Route pc1

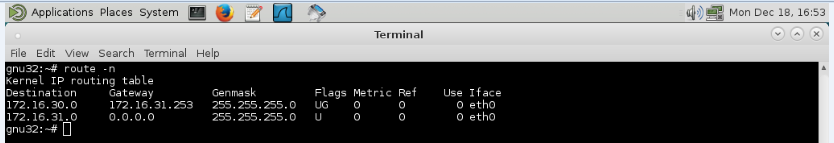


Figura 16 - Exp3: Route pc2

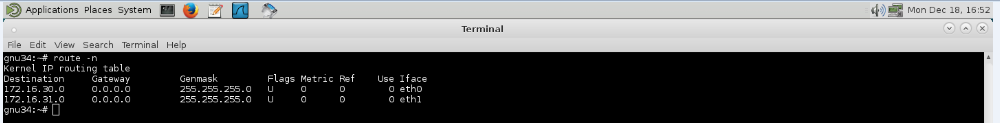


Figura 17 - Exp3: Route pc4

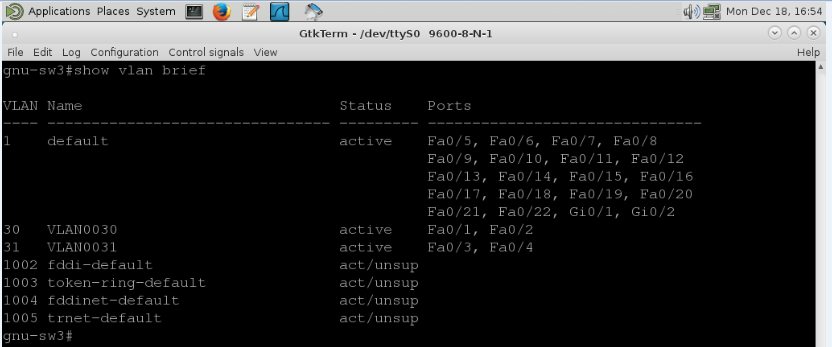


Figura 18 - Exp3: Configuração VLAN pc2

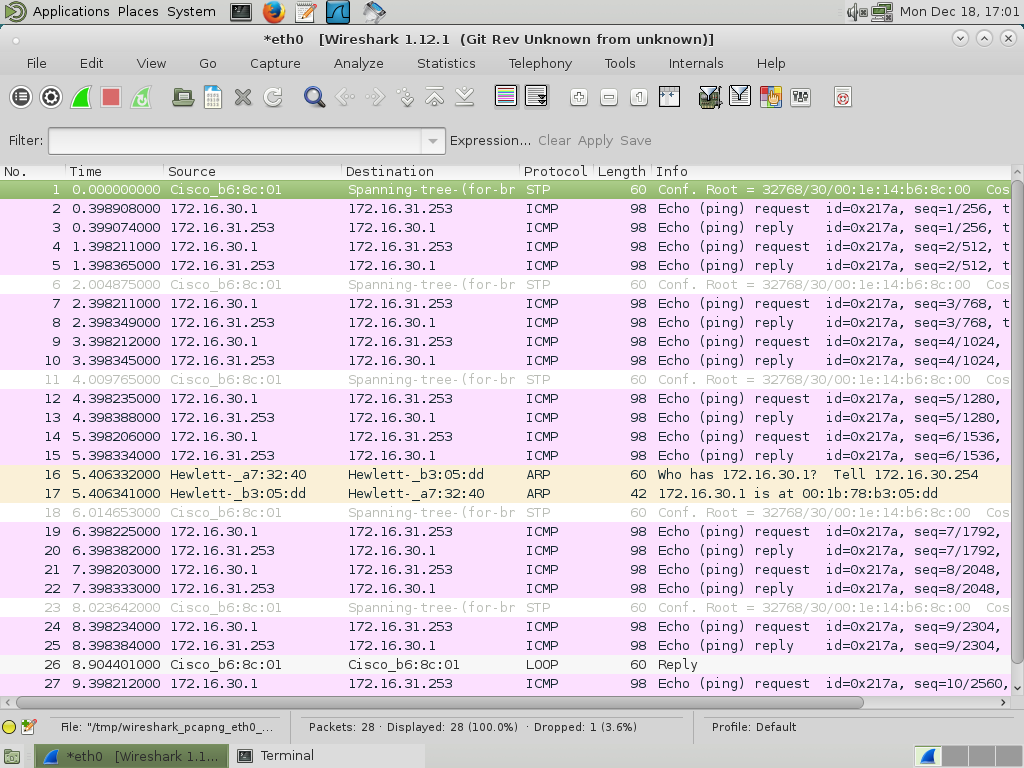


Figura 19 - Exp3: Pc1 pings 253

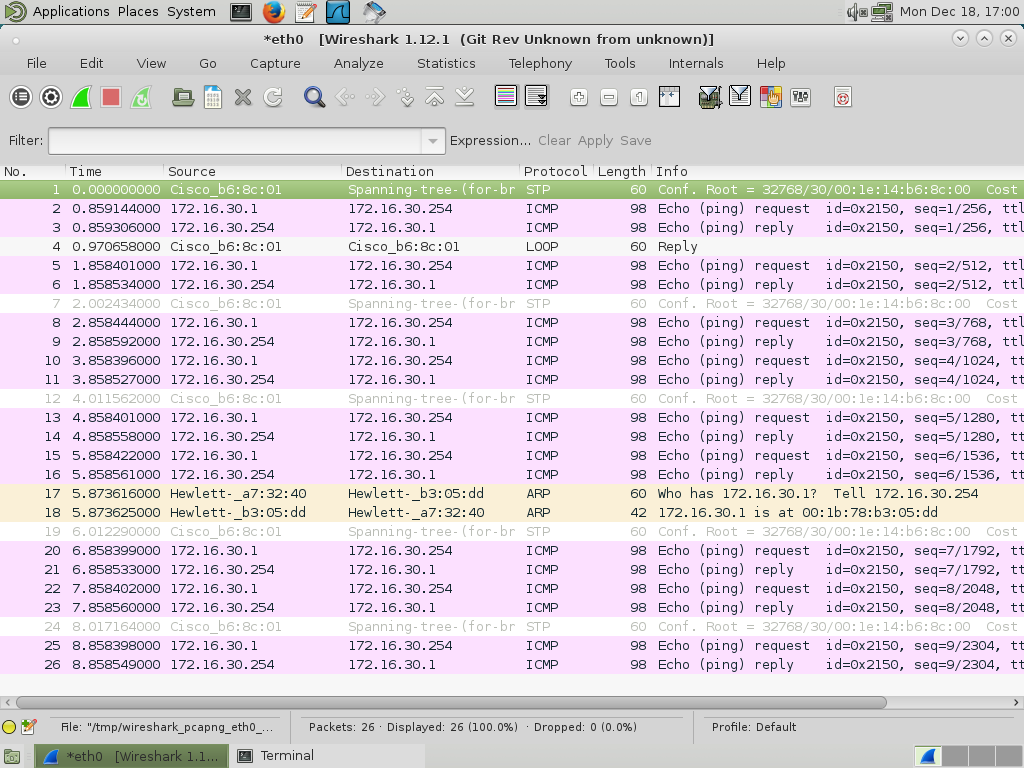


Figura 20 - Exp3: Pc1 pings 254

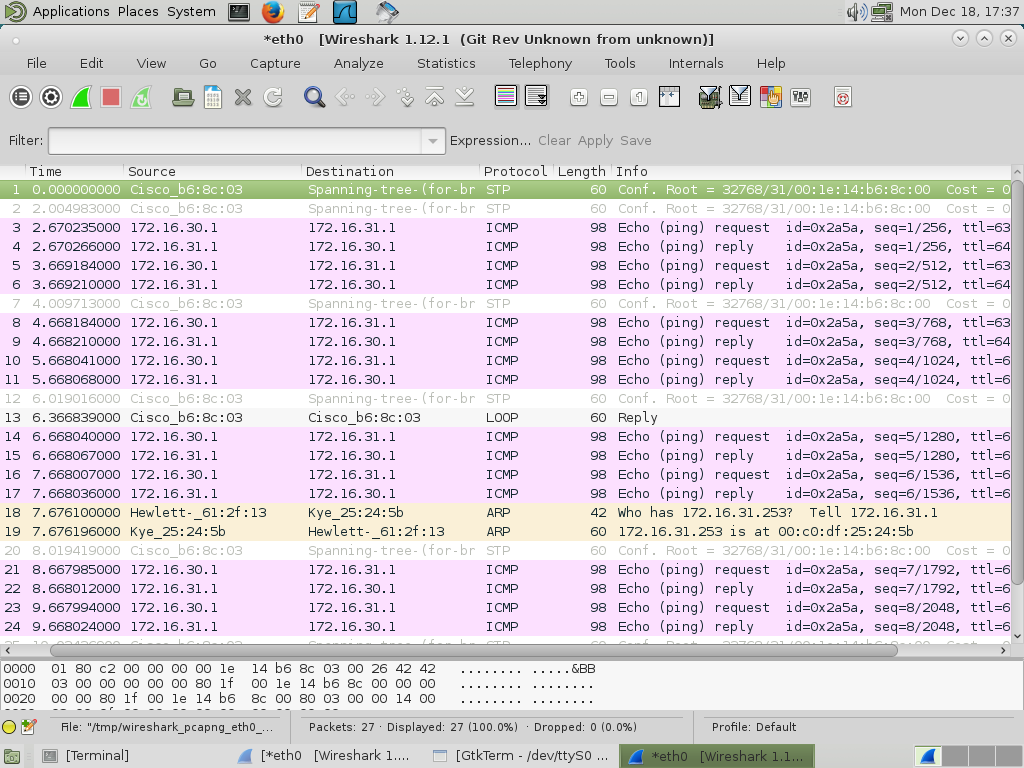


Figura 21 - Exp3: Pc2 wireshark log pc1 pings pc2

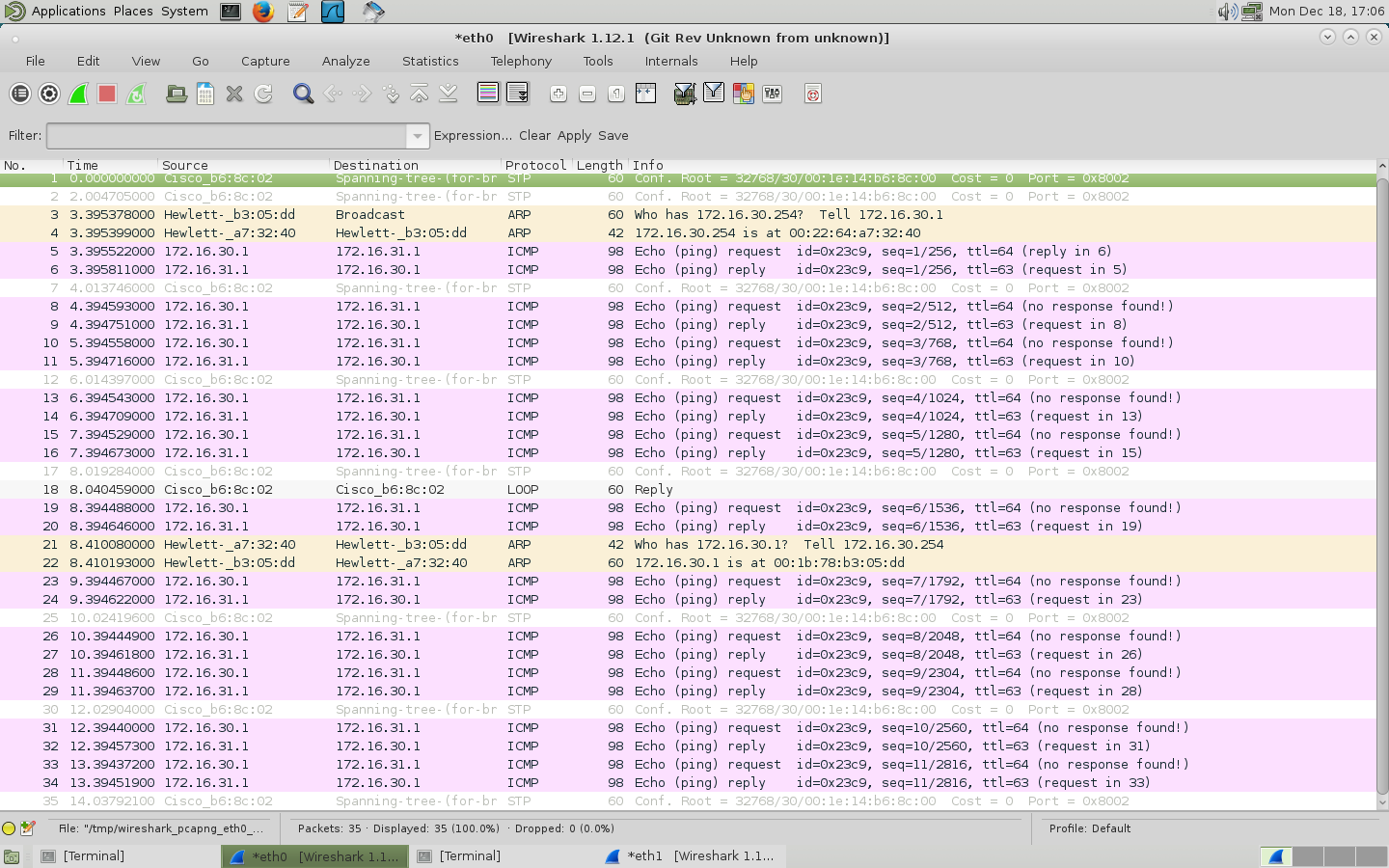


Figura 22 - Exp3: Pc4 Wireshark log eth0

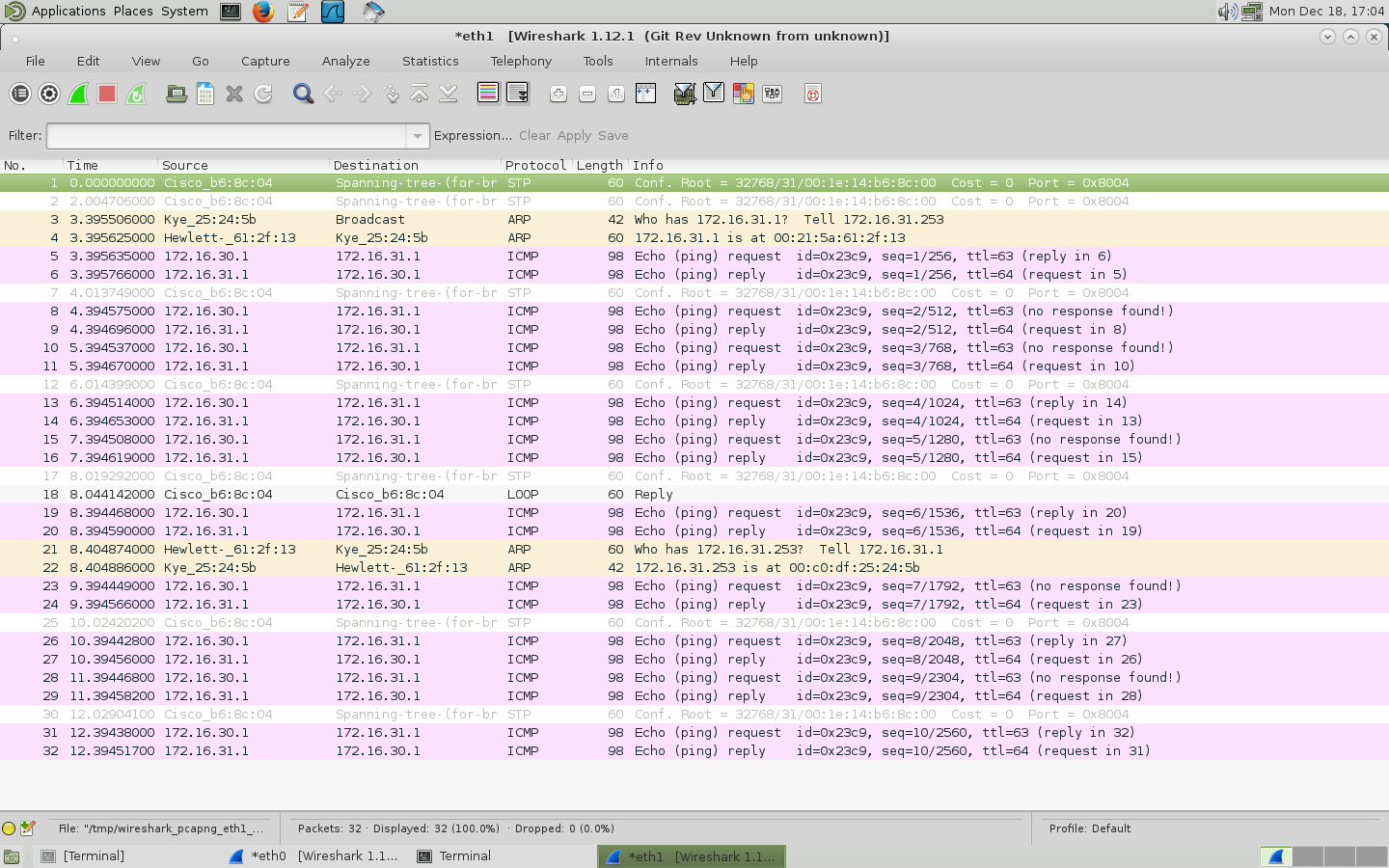


Figura 23 - Exp3: Pc4 Wireshark log eth1

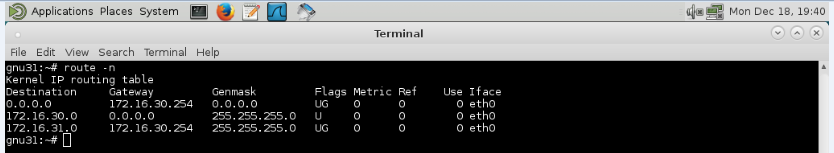


Figura 24 - Exp4: Route pc1

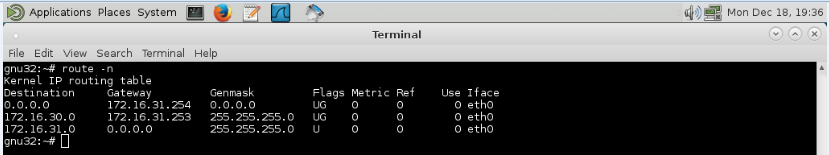


Figura 25 - Exp4: Route pc2

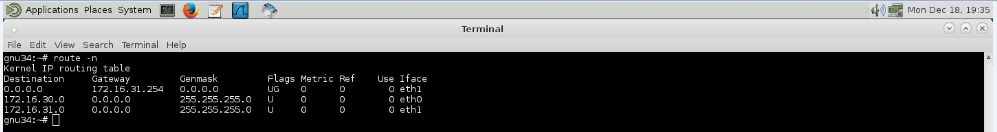


Figura 26 - Exp4: Route pc4

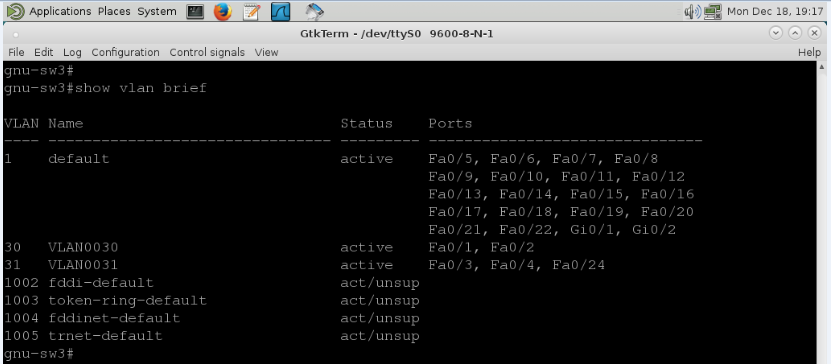


Figura 27 - Exp4: Configuração das VLANs

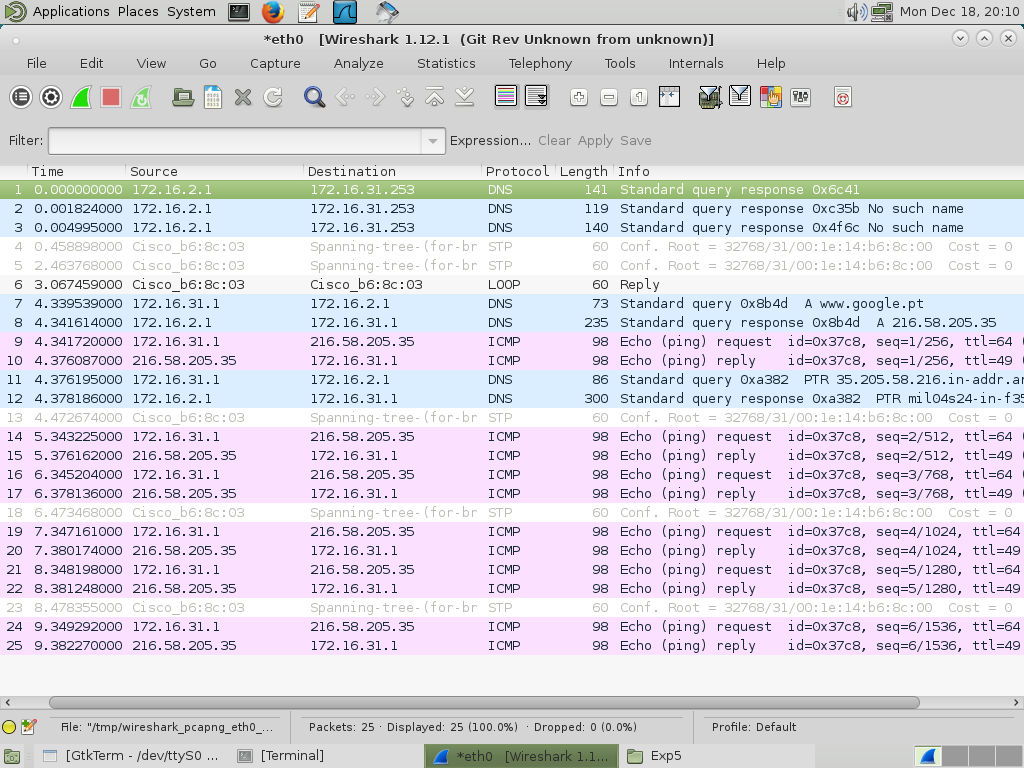


Figura 28 - Exp5: Pc2 Wireshark log após ping à Google

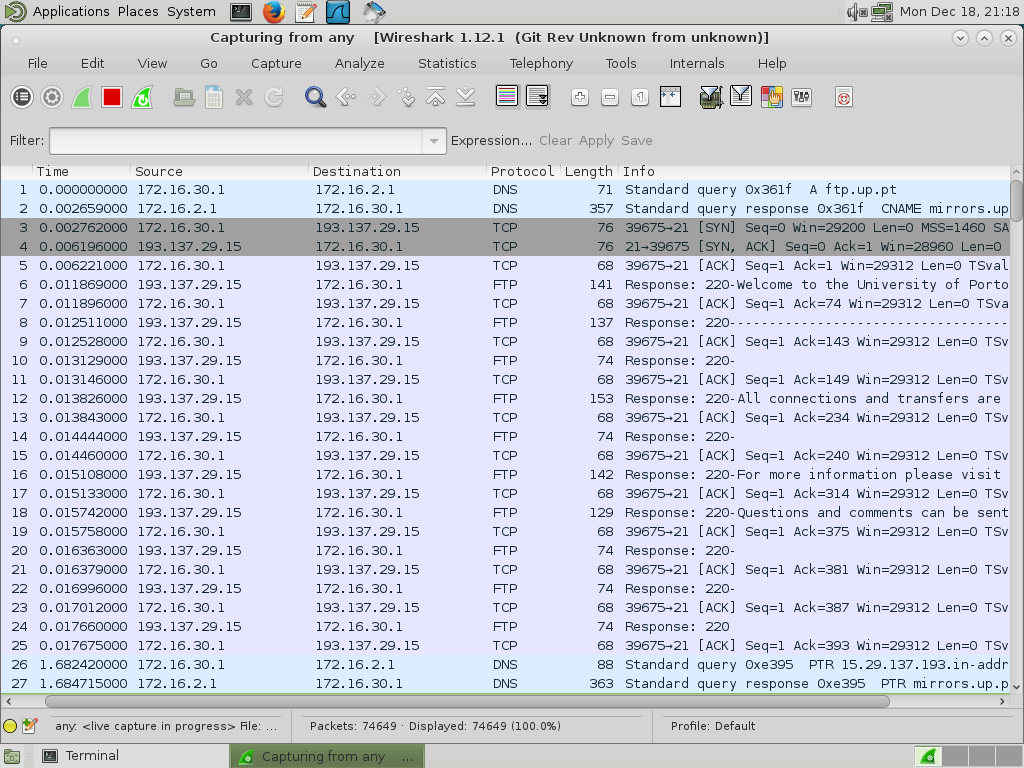


Figura 29 - Exp6: Pc1 Wireshark log download part 1

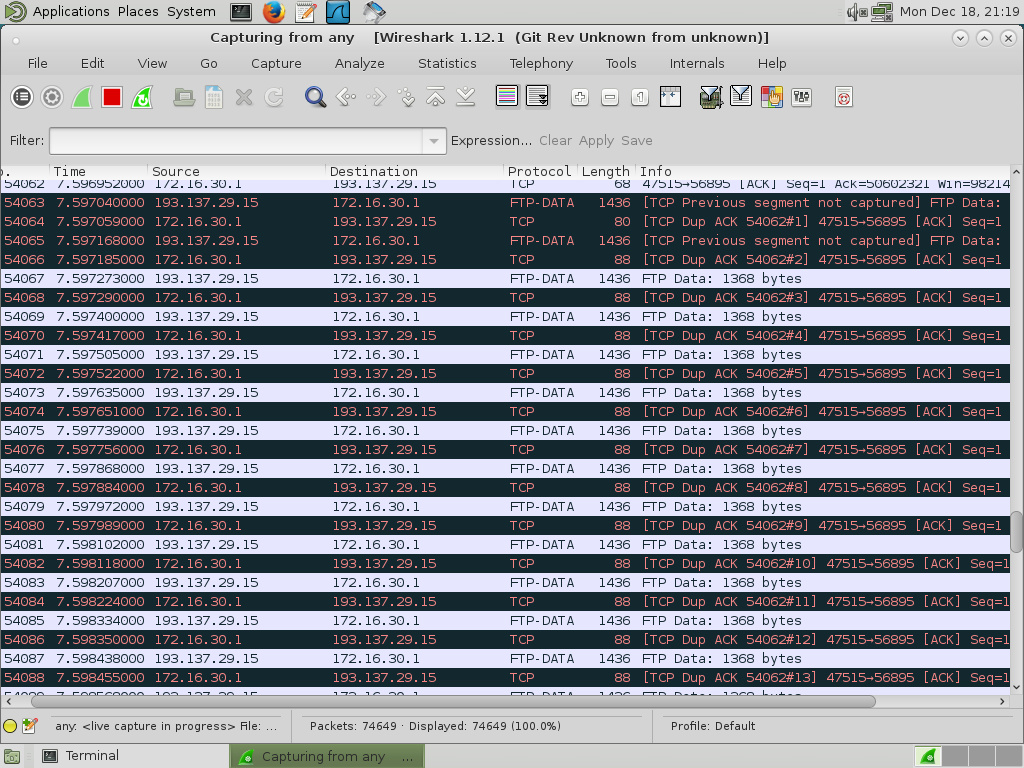


Figura 30 - Exp6: Pc1 Wireshark log download part 2

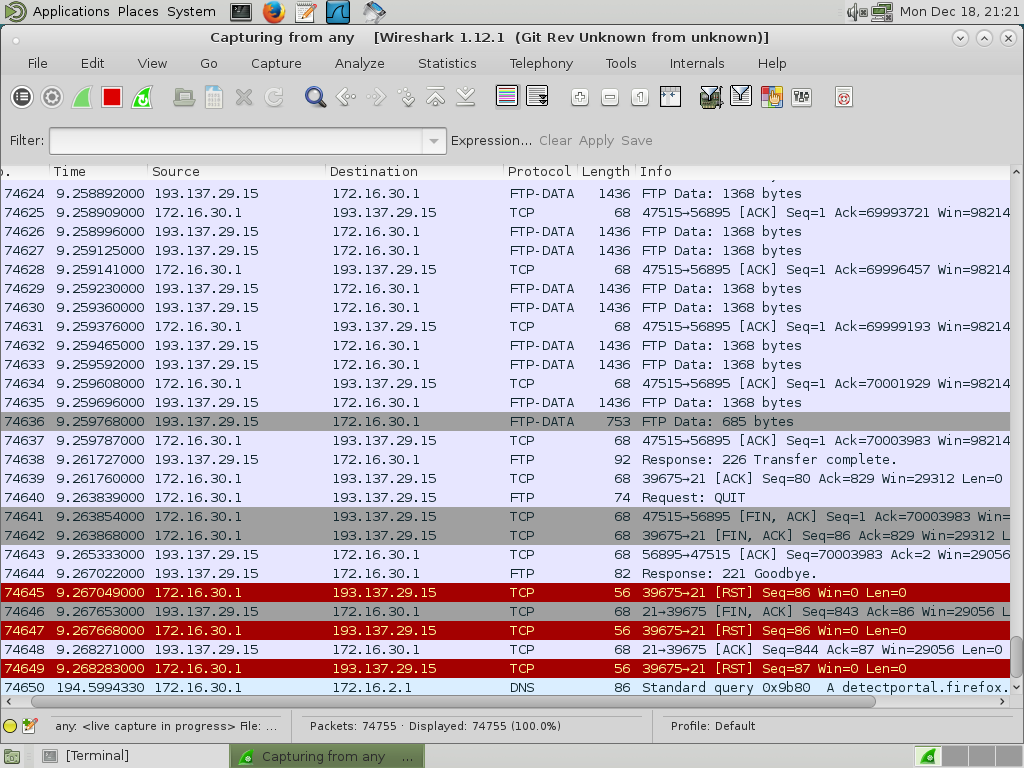


Figura 31 - Exp6: Pc1 Wireshark log download part 3

### Código da aplicação de *download*

main.c

1. #include "ip.h"
2. #include "ftp.h"
4. **int** main(**int** argc , **char** \* argv[]){
6. url url;
7. **int** parseurl = parseURL(argv[1], &url);
8. **if**(parseurl == -1)
9. **return** -1;
11. **int** ipParse = getip(&url);
12. **if**(ipParse == -1)
13. **return** -1;
15. ftp ftp;
16. **int** connectionStatus = connectftp(url.ip, url.port);
17. **if**(connectionStatus == -1)
18. **return** -1;
19. ftp.control\_socket\_fd = connectionStatus;
20. **char** response[1024];
21. **int** receiveConnectResponse;
22. **if**((receiveConnectResponse = receiveftp(&ftp, response, **sizeof**(response))) == -1)
23. **return** -1;
25. **if**(strlen(url.password) == 0){
26. **char** pass[MAX\_STRING];
27. printf("Please insert random password: ");
28. fgets(pass, MAX\_STRING + 1, stdin);
29. strcpy(url.password, pass);
30. }
32. **int** loginStatus = loginftp(&ftp, url.username, url.password);
33. **if**(loginStatus == -1)
34. **return** -1;
36. **int** changeCWDStatus = changedirectoryftp(&ftp, url.path);
37. **if**(changeCWDStatus == -1)
38. **return** -1;
40. **int** passiveModeStatus =  passiveMode(&ftp);
41. **if**(passiveModeStatus == -1)
42. **return** -1;
44. **int** retrStatus = retrftp(&ftp, url.filename);
45. **if**(retrStatus == -1)
46. **return** -1;
48. **int** downloadStatus  = downloadftp(&ftp, url.filename);
49. **if**(downloadStatus == -1)
50. **return** -1;
52. **int** disconnect = disconnectftp(&ftp);
53. **if**(disconnect == -1)
54. **return** -1;
56. **return** 0;
57. }

url.h

1. #include <string.h>
2. #include <stdio.h>
3. #include <regex.h>
4. #include <stdlib.h>
5. #include <sys/types.h>
7. #define MAX\_STRING 256
8. #define ANONYMOUS\_SIZE 10
9. #define ANONYMOUS\_STRING "anonymous\0"


13. **typedef** **char** url\_parameter[MAX\_STRING];
14. **typedef** **struct** URL {
15. url\_parameter username;
16. url\_parameter password;
17. url\_parameter hostname;
18. url\_parameter ip;
19. url\_parameter path;
20. url\_parameter filename;
21. **int** port;
22. }url;
24. **int** parseURL(**char** \* url\_param, url\* url);
25. **int** parseUser(**char** \* url\_param, url \* url);
26. **int** parseAnonymous(**char** \* url\_param, url \* url);
27. **int** parseHost(**char** \* url\_param , url\* url);
28. **int** parsePath(**char** \* url\_param, url\* url);
29. **int** parseFilename(**char** \* url\_param, url \* url);

url.c

1. #include "url.h"
3. //ftp://[<user>:<password>@]<host>/<url-path>
4. //ftp://ftp.up.pt/pub/...
5. //ftp://<user>:<password>@ftp.up.pt/pub/...
7. **const** **char** \* regExpUser = "ftp://([A-Za-z0-9])\*:([A-Za-z0-9])\*@([A-Za-z0-9.~-])+/([[A-Za-z0-9/~.\_-])+";
9. **const** **char** \* regExpAnonymous = "ftp://([A-Za-z0-9.~-])+/([[A-Za-z0-9/~.\_-])+";
11. **int** parseFilename(**char** \* url\_param, url \* url){
12. strcpy(url->filename, url\_param);
13. **return** 0;
14. }
16. **int** parsePath(**char** \* url\_param, url\* url){
17. **char**\* path = (**char**\*) malloc(strlen(url\_param));
18. path[0] = 0;
19. **int** first = 1;
20. **while** (strchr(url\_param, '/')) {
21. **char** aux[MAX\_STRING];
22. **int** counter;
23. **for** (counter = 0; url\_param[counter] != '/'; counter++) {
24. aux[counter] = url\_param[counter];
25. }
26. aux[counter] = '/';
27. aux[counter + 1] = '\0';
28. **if**(first){
29. first = 0;
30. strcpy(path, aux);
31. }
32. **else**{
33. strcat(path, aux);
34. }
35. memmove(url\_param, url\_param + counter + 1, strlen(url\_param)); //advance path and /
36. }
37. **if**(strlen(path)!=0) strcpy(url->path, path);
39. **else** url->path[0] = 0;
41. free(path);
42. **return** 0;
43. }
45. **int** parseHost(**char** \* url\_param , url\* url){
46. **int** counter;
47. **char** hostname[MAX\_STRING];
48. **for**(counter = 0; url\_param[counter] != '/'; counter++){
49. hostname[counter] = url\_param[counter];
50. }
51. hostname[counter] = '\0';
52. memcpy(url->hostname, hostname, counter + 1);
53. memmove(url\_param, url\_param + counter + 1, strlen(url\_param)); //advance the hostname
54. **return** 0;
55. }
57. **int** parseAnonymous(**char** \* url\_param, url \* url){
58. url->password[0] = 0;
59. memcpy(url->username, ANONYMOUS\_STRING, ANONYMOUS\_SIZE);
60. memmove(url\_param, url\_param + 6, strlen(url\_param)); //advance ftp
61. **return** 0;
62. }
64. **int** parseUser(**char** \* url\_param, url \* url){
65. memmove(url\_param, url\_param + 6, strlen(url\_param)); //advance ftp
66. **int** counterUsername;
67. **int** counterPassword;
68. **char** username[MAX\_STRING];
69. **for**(counterUsername = 0; url\_param[counterUsername] != ':'; counterUsername++){
70. username[counterUsername] = url\_param[counterUsername];
71. }
72. memcpy(url->username, username, counterUsername);
73. **char** password[MAX\_STRING];
74. memmove(url\_param, url\_param + counterUsername + 1, strlen(url\_param)); //advance the username and :
75. **for**(counterPassword = 0; url\_param[counterPassword] != '@'; counterPassword++){
76. password[counterPassword] = url\_param[counterPassword];
77. }
78. password[counterPassword] = '\0';
79. memcpy(url->password, password, counterPassword + 1);
80. memmove(url\_param, url\_param + counterPassword + 1, strlen(url\_param)); //advance password and @
81. **return** 0;
82. }

85. **int** parseURL(**char** \* url\_param, url\* url){
86. **int** reti;
87. **size\_t** nmatch = strlen(url\_param);
88. regmatch\_t pmatch[nmatch];
89. **char** \* tempURL = (**char** \* ) malloc(strlen(url\_param));
90. memcpy(tempURL, url\_param, strlen(url\_param));
91. regex\_t \* regexUser = (regex\_t\*) malloc(**sizeof**(regex\_t));
92. regex\_t  \* regexAnonymous = (regex\_t\*) malloc(**sizeof**(regex\_t));
93. **if** ((reti = regcomp(regexUser, regExpUser, REG\_EXTENDED)) != 0) {
94. printf("URL format is wrong: regExpUser");
95. **return** -1;
96. }
97. **if** ((reti = regcomp(regexAnonymous, regExpAnonymous, REG\_EXTENDED)) != 0) {
98. printf("URL format is wrong: regExpAnonymous");
99. **return** -1;
100. }
101. **if** ((reti = regexec(regexUser, tempURL, nmatch, pmatch, REG\_EXTENDED)) == 0) {
102. printf("URL is of type User\n");
103. parseUser(tempURL, url);
104. }
105. **else** **if** ((reti = regexec(regexAnonymous, tempURL, nmatch, pmatch, REG\_EXTENDED)) == 0) {
106. printf("URL is of type anonymous\n");
107. parseAnonymous(tempURL, url);
108. }
109. **else**{
110. printf("Error invalid URL!\n");
111. **return** -1;
112. }
114. free(regexUser);
115. free(regexAnonymous);
117. parseHost(tempURL, url);
118. parsePath(tempURL, url);
119. parseFilename(tempURL, url);
121. free(tempURL);
122. **return** 0;
123. }

ftp.h

1. #include <string.h>
2. #include <netdb.h>
3. #include <stdio.h>
4. #include <stdlib.h>
5. #include <regex.h>
6. #include <errno.h>
7. #include <unistd.h>
9. #include <sys/types.h>
10. #include <sys/socket.h>
12. #include <arpa/inet.h>
14. #include <netinet/in.h>
16. **typedef** **struct** FTP
17. {
18. **int** control\_socket\_fd; // file descriptor to control socket
19. **int** data\_socket\_fd; // file descriptor to data socket
20. } ftp;
22. **int** connectftp(**const** **char** \* ip, **int** port);
23. **int** loginftp(ftp \* ftp, **const** **char** \* username, **const** **char** \* password);
24. **int** sendftp(ftp \* ftp, **const** **char** \* str);
25. **int** receiveftp(ftp \* ftp, **char** \* resultStr, **size\_t** size);
26. **int** changedirectoryftp(ftp \* ftp, **const** **char** \* path);
27. **int** passiveMode(ftp \* ftp);
28. **int** retrftp(ftp\* ftp, **const** **char**\* filename);
29. **int** downloadftp(ftp\* ftp, **const** **char** \* filename);
30. **int** disconnectftp(ftp \* ftp);

ftp.c

1. #include "ftp.h"
3. **int** connectftp(**const** **char** \* ip, **int** port){
4. **struct**  sockaddr\_in server\_addr;
5. **int** sock;
6. /\*server address handling\*/
7. bzero((**char**\*)&server\_addr,**sizeof**(server\_addr));
8. server\_addr.sin\_family = AF\_INET;
9. server\_addr.sin\_addr.s\_addr = inet\_addr(ip);    /\*32 bit Internet address network byte ordered\*/
10. server\_addr.sin\_port = htons(port);     /\*server TCP port must be network byte ordered \*/
12. /\*open an TCP socket\*/
13. **if** ((sock= socket(AF\_INET,SOCK\_STREAM,0)) < 0) {
14. perror("socket() ERROR");
15. **return** -1;
16. }
18. /\*connect to the server\*/
19. **if**(connect(sock,(**struct** sockaddr \*)&server\_addr, **sizeof**(server\_addr)) < 0){
20. perror("connect() ERROR");
21. **return** -1;
22. }
24. printf("Connected!\n");
25. **return** sock;
26. }

29. **int** loginftp(ftp \* ftp, **const** **char** \* username, **const** **char** \* password){
30. **int** sendLoginUser;
31. **char** response[1024];
32. sprintf(response, "USER %s\r\n", username);
34. **if**((sendLoginUser = sendftp(ftp, response)) == -1)
35. **return** -1;
36. **else** printf("Username was sent\n");
37. memset(response, 0 ,**sizeof**(response));
38. **int** receiveUserResponse;
39. **if**((receiveUserResponse = receiveftp(ftp, response, **sizeof**(response))) == -1)
40. **return** -1;
42. memset(response, 0 ,**sizeof**(response));
43. **int** sendLoginPass;
44. sprintf(response, "PASS %s\r\n", password);
46. **if**((sendLoginPass = sendftp(ftp, response)) == -1)
47. **return** -1;
48. **else** printf("Password was sent\n");
50. memset(response, 0 ,**sizeof**(response));
51. **int** receivePassResponse;
52. **if**((receivePassResponse = receiveftp(ftp, response, **sizeof**(response))) == -1)
53. **return** -1;
55. printf("Logged in!\n");
56. **return** 0;
57. }
59. **int** changedirectoryftp(ftp \* ftp, **const** **char** \* path){
60. **int** sendPath;
61. **char** cwd[1024];
62. sprintf(cwd, "CWD %s\r\n", path);
64. **if**((sendPath = sendftp(ftp, cwd)) == -1)
65. **return** -1;
67. **else** printf("Path was sent\n");
69. memset(cwd, 0 ,**sizeof**(cwd));
70. **int** receiveCWDResponse;
71. **if**((receiveCWDResponse = receiveftp(ftp, cwd, **sizeof**(cwd))) == -1)
72. **return** -1;
74. printf("Changed directory!\n");
75. **return** 0;
76. }
78. **int** passiveMode(ftp \* ftp){
79. **char** pasv[1024] = "PASV\r\n";
80. **int** sendPassive = sendftp(ftp, pasv);
81. **if** (sendPassive == -1)
82. **return** -1;
84. sendPassive = receiveftp(ftp, pasv, **sizeof**(pasv));
85. **if** (sendPassive == -1)
86. **return** -1;
88. **int** ip1, ip2, ip3, ip4, p1, p2;
89. **if** ((sscanf(pasv, "227 Entering Passive Mode (%d,%d,%d,%d,%d,%d)", &ip1,
90. &ip2, &ip3, &ip4, &p1, &p2)) < 0) {
91. printf("ERROR parsing information from Passive Mode response.\n");
92. **return** -1;
93. }
95. memset(pasv, 0, **sizeof**(pasv));
97. **if** ((sprintf(pasv, "%d.%d.%d.%d", ip1, ip2, ip3, ip4))
98. < 0) {
99. printf("ERROR forming ip address\n");
100. **return** -1;
101. }
103. **int** portResult = p1 \* 256 + p2;
105. // printf("IP: %s\n", pasv);
106. // printf("port1:%d\n", p1);
107. // printf("port2:%d\n", p2);
108. // printf("PORT: %d\n", portResult);
110. ftp->data\_socket\_fd = connectftp(pasv, portResult);
111. **if** (ftp->data\_socket\_fd == -1)
112. **return** -1;
114. printf("Entered passive mode!\n");
116. **return** 0;
117. }
119. **int** retrftp(ftp\* ftp, **const** **char**\* filename) {
120. **char** retr[1024];
121. sprintf(retr, "RETR %s\r\n", filename);
122. **if** (sendftp(ftp, retr))
123. **return** -1;
125. **if** (receiveftp(ftp, retr, **sizeof**(retr)))
126. **return** -1;
128. printf("Preparing download of %s\n", filename);
129. **return** 0;
130. }
132. **int** downloadftp(ftp\* ftp, **const** **char** \* filename){
133. **FILE** \* file = fopen(filename, "w");
134. **if**(file == NULL)
135. {
136. printf("Could not create local file: %s\n", filename);
137. **return** -1;
138. }
140. **char** buffer[1024];
141. **int** bytes = 1;
142. printf("Downloading file: ");
143. **while**(bytes > 0){
144. printf("...");
145. bytes = recv(ftp->data\_socket\_fd, buffer, 1024, 0);
146. **int** fwriteStatus = fwrite(buffer, bytes, 1, file);
147. **if**(fwriteStatus < 0)
148. {
149. printf("Error on fwrite!\n");
150. **return** -1;
151. }
152. }
153. printf("\n");
154. fclose(file);
156. **char** response[1024];
157. **int** receiveUserResponse;
158. **if**((receiveUserResponse = receiveftp(ftp, response, **sizeof**(response))) == -1)
159. **return** -1;
160. **return** 0;
161. }
163. **int** disconnectftp(ftp \* ftp){
164. **int** sendQuit;
165. **char** response[1024];
166. sprintf(response, "QUIT\r\n");
168. **if**((sendQuit = sendftp(ftp, response)) == -1)
169. **return** -1;
171. close(ftp->data\_socket\_fd);
172. close(ftp->control\_socket\_fd);
173. printf("QUIT\n");
174. **return** 0;
175. }
177. **int** sendftp(ftp \* ftp, **const** **char** \* str){
178. **int** bytesWritten;
179. **if** ((bytesWritten = write(ftp->control\_socket\_fd, str, strlen(str))) <= 0) {
180. printf("ERROR: Nothing was sent. \n");
181. **return** -1;
182. }
183. //printf("Bytes sent: %d\nInfo: %s\n", bytesWritten, str);
184. **return** 0;
185. }



190. **int** receiveftp(ftp \* ftp, **char** \* resultStr, **size\_t** size){
191. **FILE** \* fd = fdopen(ftp->control\_socket\_fd, "r");
192. **if**(fd == NULL)
193. {
194. printf("fdopen() error!");
195. **return** -1;
196. }
197. **do** {
198. memset(resultStr, 0, size);
199. resultStr = fgets(resultStr, size, fd);
200. printf("%s", resultStr);
201. } **while** (!('1' <= resultStr[0] && resultStr[0] <= '5') || resultStr[3] != ' ');
203. **return** 0;
204. }

ip.h

1. #include <stdio.h>
2. #include <netdb.h>
3. #include <sys/types.h>
4. #include <netinet/in.h>
5. #include <arpa/inet.h>
6. #include "url.h"
8. #define PORT 21
10. **int** getip(url \* url);

ip.c

1. #include "ip.h"
3. **int** getip(url \* url){
4. **struct** hostent \*h;
6. **if** ((h=gethostbyname(url->hostname)) == NULL) {
7. herror("gethostbyname ERROR");
8. **return** -1 ;
9. }
11. strcpy(url->ip, inet\_ntoa(\*((**struct** in\_addr \*)h->h\_addr)));
12. url->port = PORT;
13. **return** 0;
14. }

clientTCP.c

1. /\*      (C)2000 FEUP  \*/
3. #include <stdio.h>
4. #include <sys/types.h>
5. #include <sys/socket.h>
6. #include <netinet/in.h>
7. #include <arpa/inet.h>
8. #include <stdlib.h>
9. #include <unistd.h>
10. #include <signal.h>
11. #include <netdb.h>
12. #include <strings.h>
13. #include <string.h>
15. #define SERVER\_PORT 21
16. #define SERVER\_ADDR "193.137.29.15"
18. **int** main(**int** argc, **char**\*\* argv){
20. **int** sockfd;
21. **struct**  sockaddr\_in server\_addr;
22. **char**    buf[] = "Mensagem de teste na travessia da pilha TCP/IP\n";
23. **int** bytes;
25. /\*server address handling\*/
26. bzero((**char**\*)&server\_addr,**sizeof**(server\_addr));
27. server\_addr.sin\_family = AF\_INET;
28. server\_addr.sin\_addr.s\_addr = inet\_addr(SERVER\_ADDR);   /\*32 bit Internet address network byte ordered\*/
29. server\_addr.sin\_port = htons(SERVER\_PORT);      /\*server TCP port must be network byte ordered \*/
31. /\*open an TCP socket\*/
32. **if** ((sockfd = socket(AF\_INET,SOCK\_STREAM,0)) < 0) {
33. perror("socket()");
34. exit(0);
35. }
36. /\*connect to the server\*/
37. **if**(connect(sockfd,
38. (**struct** sockaddr \*)&server\_addr,
39. **sizeof**(server\_addr)) < 0){
40. perror("connect()");
41. exit(0);
42. }
43. /\*send a string to the server\*/
44. bytes = write(sockfd, buf, strlen(buf));
45. printf("Bytes escritos %d\n", bytes);
46. **char** \* resultStr = (**char**\*) malloc(1024);
47. **FILE** \* fd = fdopen(sockfd, "r");
48. **if**(fd == NULL)
49. {
50. printf("fdopen() error!");
51. **return** -1;
52. }
53. **do** {
54. memset(resultStr, 0, **sizeof**(resultStr));
55. resultStr = fgets(resultStr, **sizeof**(resultStr), fd);
56. printf("%s", resultStr);
57. } **while** (!('1' <= resultStr[0] && resultStr[0] <= '5') || resultStr[3] != ' ');
59. close(sockfd);
60. free(resultStr);
61. exit(0);
62. }

getip.c

1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <errno.h>
4. #include <netdb.h>
5. #include <sys/types.h>
6. #include <netinet/in.h>
7. #include<arpa/inet.h>
9. **int** main(**int** argc, **char** \*argv[])
10. {
11. **struct** hostent \*h;
13. **if** (argc != 2) {
14. fprintf(stderr,"usage: getip address\n");
15. exit(1);
16. }

19. /\*
20. struct hostent {
21. char    \*h\_name;    Official name of the host.
22. char    \*\*h\_aliases;    A NULL-terminated array of alternate names for the host.
23. int     h\_addrtype; The type of address being returned; usually AF\_INET.
24. int     h\_length;   The length of the address in bytes.
25. char    \*\*h\_addr\_list;  A zero-terminated array of network addresses for the host.
26. Host addresses are in Network Byte Order.
27. };
29. #define h\_addr h\_addr\_list[0]   The first address in h\_addr\_list.
30. \*/
31. **if** ((h=gethostbyname(argv[1])) == NULL) {
32. herror("gethostbyname");
33. exit(1);
34. }
36. printf("Host name  : %s\n", h->h\_name);
37. printf("IP Address : %s\n",inet\_ntoa(\*((**struct** in\_addr \*)h->h\_addr)));
39. **return** 0;
40. }

Makefile

1. all: download
3. download: url.c ip.c url.h ip.h ftp.h ftp.c main.c
4. gcc ftp.h ftp.c ip.c ip.h url.c url.h main.c -o download -Wall
6. clean:
7. -rm -rf download