

Cliente FTP e configuração de rede

Hugo Manuel Carvalho Fonseca - 2011092010

Nuno Diogo Gonçalves Martins - 201105567

Nuno Figueiredo Pires - 200907657

**Mestrado Integrado de Engenharia Electrotécnica e de
Computadores
Redes de Computadores
2014/2015**

Sumário

O relatório é constituído por uma análise detalhada quer à implementação da aplicação de download, quer às configurações da rede local. Desse modo, serão descritas quer a arquitetura da aplicação, quer as análises às configurações feitas ao longo das várias experiências, sendo, quando necessário, auxiliadas de *log files* retirados e/ou comandos efetuados.

Após a descrição dos resultados obtidos, procurar-se-á retirar conclusões acerca dos conceitos envolvidos no projeto e que foram lecionados na unidade curricular ao longo do semestre.

Introdução

Este projeto foi desenvolvido no âmbito da unidade curricular de Redes de Computadores, lecionada no 4º ano do Mestrado Integrado de Engenharia Electrotécnica e de Computadores, do ramo de Telecomunicações, Electrónica e Computadores, da Faculdade de Engenharia da Universidade do Porto, com os seguintes objetivos:

1. Desenvolvimento de uma aplicação, responsável pelo *download* de um único ficheiro em cada execução
2. Configuração e estudo de uma rede local de computadores

A aplicação efectua *download* de um ficheiro de um servidor FTP, através da implementação de um protocolo FTP auxiliado de *sockets* em C, seguindo a norma especificada no RFC959, sendo que os dados introduzidos pelo utilizador seguem a norma especificada no RFC1738.

Os *sockets* são utilizados para comunicar com o servidor, enviando uma série de comandos e recebendo as respectivas respostas por fim a descargar o ficheiro especificado pelo utilizador.

Por fim, seguem-se os passos para a configuração de uma rede local de computadores, i.e., configuração de uma rede de IPs, criação e configuração de LANs virtuais, configuração de um *router* e de um *router* comercial com implementação de NAT, e, por último, configuração de DNS.

Após estes passos, separados em várias experiências, foi testada a aplicação na rede configurada.

Em relação à estrutura do relatório, este encontra-se dividido em seis secções:

- Descrição da arquitectura da aplicação, e exposição de um *download* bem sucedido por fim a mostrar o seu bom funcionamento
- Descrição da configuração da rede local e análise dos resultados obtidos, bem como dos objectivos de cada experiência, auxiliada pelos *log files* obtidos (incluídos em anexo)
- Conclusões obtidas após a finalização do projeto, onde será feito um resumo do que foi tratado e salientados os pontos fulcrais do projeto
- Inclusão de anexos importantes, nomeadamente o código da aplicação, comandos usados nas configurações da rede e os *log files*

Cliente FTP

Para o desenvolvimento da aplicação de *download* tentou-se simplificar ao máximo a lógica do programa, seguindo-se para esse efeito a ideia de ‘automatizar’ a experiência de efectuar o mesmo *download* na aplicação *Telnet*.

Para a execução da aplicação é necessário passar-lhe como argumento o URL no formato descrito no RFC1738 que é o seguinte:

```
ftp://<username>:<password>@<host>/<path>
```

Procede-se então ao processamento da string para extração dos campos que são necessários para a ligação ao servidor, sendo para o efeito chamada a função `check_URL()` que aceita tanto o argumento com todos os campos especificados, como com *username*, *host* e *path* ou apenas *host* e *path*. Quando o *username* e *password* não são especificados a autenticação é feita no modo *anonymous*.

De seguida efectua-se a ligação ao servidor usando para o efeito as funções `getaddrinfo()`, `socket()` e `connect()`. Se a ligação for bem sucedida segue-se para o envio dos comandos necessários. Envia-se primeiro os comandos *USER* e *PASS* (função `send_command()`) para efectuar a autenticação no servidor. Ao receber as respostas (função `get_answer()`) é verificado o código recebido e se este for um código de erro (≥ 400), então a execução da aplicação é abortada.

Após uma autenticação bem sucedida, é enviado o comando *PASS* para se passar ao modo passivo e fazer o *download*. Na resposta do servidor vem o novo IP e porta a que a aplicação se deve ligar. Extraídos esses dados, cria-se a uma nova ligação.

Por fim, é enviado para o primeiro *socket* o comando *RETR* com o caminho e nome do ficheiro na forma *<path>/<file>* e o ficheiro é lido e gravado (com o nome correcto, extraído do *path*) através do segundo *socket*, chamando a função `get_file()`.

Com o download bem sucedido fecha-se a ligação e a execução da aplicação termina. Tanto o código como o registo de um *download* bem sucedido podem ser consultados no anexo C.

É também de notar que para além desta versão do código, foi demonstrada outra que usava o comando *LIST* para listar os conteúdos da directória ao utilizador e então dava a opção de este escolher qual dos ficheiros queria fazer *download*.

Além disso, é possível também alterar uma *flag* na criação da estrutura que é passada à função `getaddrinfo()` de *AF_INET* para *AF_UNSPEC*, para tornar o código independente da versão do IP, mas infelizmente esta funcionalidade não foi possível de demonstrar na rede do laboratório.

Configuração de rede IP

Esta experiência teve como objetivo a configuração de uma rede IP que, neste caso, foi estabelecida entre dois computadores através de um *switch*.

Inicialmente, foi desligado o cabo que liga o *switch* ao *router* do laboratório, algo que resultou na perda da ligação à internet em ambos os computadores. Deste modo, pretendeu-se entender o funcionamento dos protocolos ARP e ICMP quando se envia/recebe um *ping*. Feito isso, configurou-se os IPs de cada *tux* pelos valores especificados, através dos comandos especificados no anexo A.1.1

O protocolo ARP (*Address Resolution Protocol*) é usado quando um computador, conectado a uma rede, pretende enviar uma trama ethernet, sendo conhecedor do endereço IP do computador a que se destina. Desse modo, antes de enviar a trama, o computador procura saber qual é o endereço MAC correspondente ao IP que se pretende. Tal é conseguido através do envio de uma ARP *request*, em *broadcast*, obtendo essa informação. Cada computador que se encontra conectado à rede recebe essa mensagem e, caso verifique que essa mensagem lhe é destinada, retornará a informação pretendida, igualmente em *broadcast*. De seguida, o computador que solicitou as informações irá atualizar a sua ARP *table* – tabela que contém os endereços MAC para IPs já conhecidos, enviando, posteriormente, a trama ethernet com as informações que já contém.

Deste modo, cada trama ARP contém quer o IP de origem, quer o de destino, identificando assim o caminho que se pretende fazer. No entanto, é também identificado o endereço MAC do computador do qual partiu a trama pela última vez, assim como o endereço MAC para onde esta se dirige. Tal deve-se devido ao facto de, em cada momento, a trama ARP poder situar-se num qualquer sítio aleatório da rede e é necessário saber o último endereço MAC para poder enviar a resposta e o endereço MAC de destino onde continuar a procurar (inicialmente, o de *broadcast*).

Quando enviamos um comando *ping*, no caso deste ainda não ter na sua tabela ARP a identificação de qual computador possui determinado IP, envia antes uma trama ARP, por forma a obter esta informação (tal como foi anteriormente descrito). De seguida, será enviada uma trama ICMP (*echo request*) para o computador respetivo. Em consequência disso, o computador que recebeu a trama enviará de volta uma trama ICMP (*echo reply*). Os endereços MAC e IP destas duas tramas estão em conformidade com o que foi descrito no parágrafo anterior.

É também possível determinar qual o tipo de trama ethernet. Para tal, basta observar os 2 bytes correspondentes ao *header*. No caso de conter o valor 0x0800, corresponde ao protocolo ARP, ao invés do valor 0x0806, que corresponde ao protocolo IPv4. Para além destes, no caso de se tratar de um protocolo IP, existe no *header* a informação sobre o protocolo correspondente e que, no caso do ICMP, conterá o valor 1. O pacote IP contém a informação acerca do tamanho da trama, que é facilmente obtido através do *Wireshark*.

Por último, uma interface *loopback* é uma interface virtual que não está associada a qualquer componente físico num computador, sendo habitualmente usada para testes. Assim, tal como seu nome indica, qualquer mensagem transmitida para esta interface é imediatamente recebida pelo mesmo. De notar que, por definição, esta interface deve estar sempre ativa e que, quando é estabelecida uma nova vizinhança através desta, apenas haverá perda de vizinhança quando as interfaces físicas ficarem inativas.

Implementação de duas LANs virtuais através do *switch*

Esta experiência teve como objetivo a criação e implementação de duas LANs virtuais (VLAN) através do *switch*, podendo as portas ser escolhidas de forma personalizada.

Para se criar e configurar uma VLAN, foram necessários determinados comandos executados no ambiente do *GTKTerm* e que poderão ser observados em anexo (A.2.2). No entanto, foram ainda necessários outros comandos para tornar as VLANs criadas funcionais, nomeadamente a permissão a cada computador para responder a *pings broadcast*, bem como configurar quais os IPs correspondentes (o comando *echo* presente no anexo A.2.1) e as rotas que cada computador possuirá.

Nesta experiência, existem dois domínios *broadcast*, sendo tal comprovado pelos *logs* obtidos e que se encontram no anexo B.2. Os *logs* mostram o que acontece na rede após um *ping broadcast* enviado pelo *tux21*. Conclui-se que o *tux22* não recebe os pacotes ICMP, ao contrário do *tux24*. No entanto, este último apenas não responde aos pacotes pois a opção para resposta a

pings broadcast está desativada. Nos anexos são ainda incluídos os *logs* obtidos na sequência de um *ping broadcast* enviado pelo *tux22* e, tal como se pode verificar, o *tux22* é o único que recebe os pacotes do *ping*.

Configuração de um Router

Esta experiência teve por base a configuração de um computador como router (neste caso, do *tux24*), por forma a permitir a comunicação entre as duas VLANs criadas na experiência anterior.

Deste modo, é necessário que os *tux21* e *tux22* conheçam um caminho pelo qual pode enviar o pacote quando o computador destinado não se encontra na rede. Configurando o *tux24* como router – colocando cada interface deste em cada uma das VLANs criadas – e definindo-o como *gateway* dos *tux21* e *tux22* (anexo A.3.1), é obtido o pretendido.

Esta configuração é descrita pelas tabelas de re-encaminhamento em cada *tux*, a qual contém as rotas com os destinos para os quais se devem enviar pacotes, sabendo o IP de destino.

As tabelas de encaminhamento caracterizam as rotas em várias componentes: IP de destino, *gateway* – IP para o qual se devem enviar os pacotes, caso não exista nenhuma rota para o IP de destino –, máscara de rede e a interface na qual se envia o pacote (existem outros campos, mas só estes interessam para a nossa análise).

Por este motivo, é justo dizer que o *tux24* assume um papel vital na nossa rede, pois é ele que re-encaminha os pacotes para o *tux* respetivo. Além deste *tux* ser o *gateway* do *tux21* e do *tux22*, contém também na sua tabela de endereçamento a informação de que, para a rede do lado do *tux21* (i.e., a rede 172.16.20.0/24), a informação passa pela interface *eth0*, enquanto para a rede do lado oposto (i.e., a rede 172.16.21.0/24), a informação passa pela interface *eth1*.

Em relação aos *logs* observados (anexo B.3), é possível observar o que se conclui na experiência 1 sobre os endereços IP e MAC existentes nos pacotes ARP e ICMP. Tal foi derivado por um *ping* enviado do *tux21* para o *tux22*, observado a partir do *tux24*, e que se concluiu que, para enviar esse *ping* para o *tux22*, o pacote ICMP respetivo que passa na interface *eth0* contém o endereço MAC de destino do *tux24* (do lado da interface *eth0*) e não o do *tux22*. Na situação contrária, isto é, do envio da resposta ao *ping* pelo *tux22*, o pacote ICMP contém igualmente o endereço MAC de origem do *tux24* (*eth0*) e não o do *tux22*. Em relação aos pacotes ICMP que passam pela interface *eth1*, a situação é idêntica, contudo, os endereços MAC de origem e destino serão o do *tux24*, conforme seja receção ou resposta ao *ping*.

Configuração de um Router comercial e implementação de NAT

Esta experiência teve como objetivo a configuração de um router comercial para que, posteriormente, possa ser implementado o NAT. Além disso, as restantes configurações seguem o que foi feito na experiência anterior (nomeadamente, as VLANs existentes e a atuação do *tux24* como *router*).

A configuração do *router* comercial começa por atribuir as interfaces para a rede configurada, tendo conta as ligações efetuadas no *switch*, adicionando posteriormente as rotas que possibilitem a correta comunicação entre todos os *tux*s. Os comandos relativas à configuração do router comercial pode ser observada em anexo (A.4.3), destacam-se que, para definir uma rota estática, será necessário o seguinte comando:

```
ip route <ip_destino> <mascara_subrede> <gateway>
```

De seguida, o *tux24* (*eth0*) foi definido como *default gateway* do *tux21*. Já o router comercial foi definido como *default gateway* do *tux22* e do *tux24* (*eth1*). Por forma a mostrar a eficácia destas configurações, foi efetuado um *ping* a partir do *tux21* para as interfaces do *tux24*, para o *tux22* e também para o *router*.

Como era esperado, após esta configuração, os pacotes enviados pelo *tux21* vão para o *tux24* e, a partir deste, são redirecionados para o *router* ou para o *tux22*, conforme o que se especificou. Ao remover a rota do *tux22* para o *tux21* conclui-se (pelos *logs* em anexo (B.4)) que ao tentar fazer *ping*, o pacote primeiro viaja até ao router que então informa o *tux22* que fica mais perto ir a partir do *tux24*.

Por fim, e por forma a obter resposta às tentativas de contacto à rede exterior, procedeu-se à configuração do NAT (*Network Address Translation*), para que os terminais exteriores possuam uma forma de saber a qual endereço devem responder, uma vez que todos os *tuxs* são vistos como possuindo o mesmo endereço.

Configurando o NAT, todos os endereços da rede são mapeados para endereços públicos e, desta forma, a resposta é concretizável, pois a rede exterior já sabe identificar cada *tux* separadamente. Pode-se ser observado o *ping* do *tux21* para a rede de laboratório no anexo B.4. Os comandos responsáveis pela configuração do NAT podem ser consultados nos anexos (anexo A.4.4).

Configuração de DNS

Esta experiência teve como objetivo a possibilidade de a rede criada possuir um serviço resolução de nomes, ou DNS (*Domain Name System*). Deste modo, foi permitida a utilização direta do *hostname* ao invés de apenas ser possível o uso do IP pretendido.

Para um *host* poder ter serviços de DNS basta que tenha, no mínimo, uma tabela na qual, para cada *hostname*, seja identificado o IP que lhe está associado. Assim, na experiência feita, bastou apenas identificar no computador a localização de um servidor DNS existente (neste caso, o servidor *netlab.fe.up.pt*, com IP 172.16.2.1), contendo já estas tabelas. Para o computador puder ter acesso a estas informações, bastou editar o ficheiro *resolv.conf*, existente na pasta /etc, colocando a informação seguinte:

```
search netlab.fe.up.pt  
nameserver 172.16.2.1
```

Podemos observar pelos Anexos B.5. que, quando executado um *ping* para *www.facebook.com*, é enviado um pacote DNS para o servidor DNS configurado, requerendo o IP do site “pingado”. Após este pedido, é recebido um outro pacote DNS enviado pelo servidor com a informação pretendida e, após isso, o primeiro *ping* é enviado. Posteriormente, antes de cada pacote enviado, é visualizado o mecanismo reverso, que perante um IP conhecido, o servidor DNS informa sobre qual a máquina correspondente.

Ligações TCP

Esta experiência teve como objetivo a criação de ligações TCP usando para o efeito a aplicação desenvolvida para *download* de um ficheiro de um servidor FTP. Nesta experiência, ao contrário das anteriores, foram usados os computadores da bancada 6.

Pela análise dos *logs* capturados na realização da experiência, repara-se que ao iniciar a execução da aplicação, é estabelecida uma ligação TCP com o servidor.

Esta ligação diz respeito ao primeiro *socket* aberto pela aplicação, que é usado para o envio de comandos e escuta das respostas do servidor.

Posteriormente ao envio do comando PASV, para passagem ao modo passivo, é calculada a nova porta e feita uma nova ligação através de um segundo *socket*.

Essa segunda ligação pode ser também observada no *log*, onde se repara que embora esteja a ser feita uma comunicação para o mesmo endereço IP, a porta de destino é diferente (inicialmente estava a comunicar para a porta 21 para o controlo da ligação FTP, passando de seguida a comunicar para a porta 50011 para a transferência do ficheiro).

As ligações TCP estão divididas em três fases, sendo essas a fase de estabelecimento de comunicação, que se segue por uma fase de transferência de dados e termina com a fase de encerramento da comunicação.

Com uma análise aos primeiros pacotes TCP no *log* capturado vê-se que a fase de estabelecimento de comunicação dá-se em três passos:

1. O cliente envia uma trama SYN ao servidor, que o informa que se pretende realizar uma transferência de dados.
2. O servidor responde com uma trama SYN ACK que informa o cliente que o servidor está pronto para iniciar a comunicação.
3. O cliente responde com uma trama ACK que informa o servidor que também ele está pronto

No modo de transferência de dados ocorre o envio das diversas tramas que contêm a informação solicitada pelo cliente, neste caso o ficheiro. Nesta fase existem diversos mecanismos que asseguram que a transferência ocorre com sucesso, como números de sequência que permitem ordenar as tramas enviadas e detectar possíveis perdas de informação, *checksums*, que permitem detectar erros nas tramas e temporizadores e ACKs (ARQ TCP) que possibilitam a detecção de tramas perdidas. Com uma análise cuidada dos *logs* (anexo B.6) é possível reparar na existência de muitas tramas perdidas, com erros que são depois retransmitidas, e também de tramas duplicadas.

Feita a transferência de dados, o cliente envia uma trama FIN para comunicar ao servidor que deseja terminar a ligação, e este responde com uma trama ACK e quando estiver também preparado para terminar a comunicação irá ocorrer novamente a mesma troca de tramas, mas com sentidos inversos.

O mecanismo de ARQ TCP, já mencionado anteriormente, utiliza *acknowledgements* e *timeouts* para assegurar que a transferência se dá correctamente. Este mecanismo usa uma janela que vai mudando de tamanho conforme a estabilidade da ligação. Esta janela ajuda o controlo de fluxo e de congestionamento pois aumenta/diminui o número de bytes que são transferidos por segundo.

Por fim, testando fazer o *download* no *tux61* e a meio da transferência começar um novo *download* no *tux62*, nota-se que a transferência é mais demorada, pois existem diversas falhas da ligação, com quedas abruptas na velocidade de transferência, como se pode observar pelas imagens em anexo (B.6).

Implementação de NAT em Linux

Esta experiência teve como finalidade a implementação de NAT no *tux24*.

Para implementar NAT em Linux é preciso dar uma série de comandos (anexo A.5) por forma a criar regras de NAT que digam ao sistema que conexões modificar e como o fazer. Para tal, é necessário realizar camuflagem e para isto é preciso implementar NAT na fonte, ou seja, modificar a informação que diz respeito à máquina de onde provém uma ligação que sai da rede. Executa-se então o seguinte comando:

```
iptables -t nat -A POSTROUTING -o eth1 -j MASQUERADE
```

Isto faz com que todos os pacotes têm como IP de origem o da interface eth1. De seguida é necessário implementar regras para *forwarding*, o que irá permitir a existência de tráfego. Para tal usa-se o comando:

```
iptables -A FORWARD -i eth1 -m state --state NEW,INVALID -j DROP
```

Feito isto, e gerando tráfego no *tux21*, repara-se que os pacotes que chegam são modificados de modo a apresentarem o IP da interface eth1 como IP fonte, conforme se verifica nos *logs* capturados (anexo **B.5**).

Conclusões

Após a apresentação do trabalho e da documentação do mesmo neste relatório pode-se dizer que concluímos todos os pontos propostos com sucesso.

Este trabalho, em particular as experiências, permitiu perceber melhor toda a complexidade por detrás de algo que usamos todos os dias e dá-mos como garantido, que são as redes de computadores.

O desenvolvimento da aplicação permitiu-nos também entender melhor o protocolo FTP, e também ganhar alguma experiência com programação/desenvolvimento de software com funcionalidades de rede, através do uso de *sockets*.

É também importante referir que este trabalho nos despertou uma curiosidade em tentar saber mais sobre esta área tão importante e com grande destaque nas últimas décadas, que veio abrir as portas para o mundo ligado que temos hoje e que só vai crescer no futuro.

Bibliografia

- Brian Hall, “Beej’s Guide to Network Programming” [Online] Disponível em: <http://beej.us/guide/bgnet/>
- Peter Prinz & Ulla Kirch-Prinz, “C Pocket Reference”, O’Reilly Media Inc., 1st Edition, 2002, ISBN: 978-0-596-00436-1
- Andrew S. Tanenbaum & David J. Wetherall, “Computer Networks”, Pearson Education Limited, 5th Edition, 2014, ISBN: 978-1-292-02422-6

Anexo A - Comandos para as configurações

1. Experiência 1

1. Configuração dos IPs

```
tux21 >> ifconfig eth0 172.16.20.1/24  
tux24 >> ifconfig eth0 172.16.20.254/24
```

2. Experiência 2

1. Configuração dos IPs

```
tux21 >> ifconfig eth0 172.16.20.1/24  
tux24 >> ifconfig eth0 172.16.20.254/24  
tux22 >> ifconfig eth1 172.16.21.1/24  
tux24 >> echo 0 > /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts
```

2. Criação e configuração das VLANs

```
>> configure terminal
```

```
>> vlan 20
```

```
>> vlan 21
```

```
>> interface fastethernet 0/1  
>> switchport mode access  
>> switchport access vlan 20
```

```
>> interface fastethernet 0/4  
>> switchport mode access  
>> switchport access vlan 20
```

```
>> interface fastethernet 0/2  
>> switchport mode access  
>> switchport access vlan 21
```

```
>> end
```

3. Experiência 3

1. Configuração dos IPs

```
tux21 >> ifconfig eth0 172.16.20.1/24  
tux21 >> route add default gateway 172.16.20.254/24  
tux24 >> ifconfig eth0 172.16.20.254/24  
tux24 >> ifconfig eth1 172.16.21.253/24  
tux22 >> ifconfig eth1 172.16.21.1/24  
tux22 >> route add default gateway 172.16.21.253/24  
tux24 >> echo 0 > /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts  
tux24 >> echo 1 > /proc/sys/net/ipv4/ip_forward
```

2. Criação e configuração das VLANs

```
>> configure terminal  
  
>> vlan 20  
>> vlan 21  
  
>> interface fastethernet 0/1  
>> switchport mode access  
>> switchport access vlan 20  
  
>> interface fastethernet 0/4  
>> switchport mode access  
>> switchport access vlan 20  
  
>> interface fastethernet 0/3  
>> switchport mode access  
>> switchport access vlan 21  
  
>> interface fastethernet 0/2  
>> switchport mode access  
>> switchport access vlan 21  
  
>> end
```

4. Experiência 4

1. Configuração dos IPs e predefinição do *tux24* e do *router* como *default gateways*

```
tux21 >> ifconfig eth0 172.16.20.1/24  
tux21 >> route add default gateway 172.16.20.254/24  
tux24 >> ifconfig eth0 172.16.20.254/24  
tux24 >> ifconfig eth1 172.16.21.253/24  
tux24 >> route add default gateway 172.16.21.254/24  
tux22 >> ifconfig eth1 172.16.21.1/24  
tux22 >> route add default gateway 172.16.21.253/24  
tux24 >> echo 0 > /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts  
tux24 >> echo 1 > /proc/sys/net/ipv4/ip_forward
```

2. Criação e configuração das VLANs

```
>> configure terminal  
  
>> vlan 20  
>> vlan 21  
  
>> interface fastethernet 0/1  
>> switchport mode access  
>> switchport access vlan 20
```

```
>> interface fastethernet 0/4
>> switchport mode access
>> switchport access vlan 20

>> interface fastethernet 0/3
>> switchport mode access
>> switchport access vlan 21

>> interface fastethernet 0/2
>> switchport mode access
>> switchport access vlan 21

>> interface fastethernet 0/5
>> switchport mode access
>> switchport access vlan 21

>> end
```

3. Configuração do *router*

```
>> configure terminal

>> interface fastethernet 0/0
>> ip address 172.16.21.254
>> no shutdown
>> exit

>> interface fastethernet 0/1
>> ip address 172.16.2.29
>> no shutdown
>> exit

>> ip route 0.0.0.0 0.0.0.0 172.16.2.254
>> ip route 172.16.20.0 255.255.255.0 172.16.21.253
>> end
```

4. Implementação de NAT no *router*

```
>> configure terminal

>> interface fastethernet 0/0
>> ip address 172.16.21.254 255.255.255.0
>> no shutdown
>> ip nat inside
>> exit

>> interface fastethernet 0/1
>> ip address 172.16.2.29 255.255.255.0
>> no shutdown
```

```
>> ip nat outside
>> exit

>> ip nat pool ovrlid 172.16.2.29 172.16.2.29 prefix 24
>> ip nat inside source list 1 pool ovrlid overload

>> access-list 1 permit 172.16.20.0 0.0.0.7
>> access-list 1 permit 172.16.21.0 0.0.0.7

>> ip route 0.0.0.0 0.0.0.0 172.16.2.254
>> ip route 172.16.20.0 255.255.255.0 172.16.21.253
>> end
```

5. Experiência 7

1. Implementação de NAT no *tux24*

```
tux64 >> iptables -t nat -A POSTROUTING -o eth1 -j MASQUERADE
tux64 >> iptables -A FORWARD -i eth1 -m state --state NEW,INVALID -j DROP
tux64 >> iptables -L
tux64 >> iptables -t nat -L
```

Anexo B - Logs capturados

1. Experiência 1

No.	Time	Source	Destination	Protocol	Length	Info
30	44.523803000	G-ProCom_8b:e4:c5	Broadcast	ARP	42	who has 172.16.20.254? Tell 172.16.20.1
31	44.524161000	Hewlett-_a7:26:a2	G-ProCom_8b:e4:c5	ARP	60	172.16.20.254 is at 00:22:64:a7:26:a2
32	44.524179000	172.16.20.1	172.16.20.254	ICMP	98	Echo (ping) request id=0x69d8, seq=1/256, ttl=64 (reply in 33)
33	44.524438000	172.16.20.254	172.16.20.1	ICMP	98	Echo (ping) reply id=0x69d8, seq=1/256, ttl=64 (request in 32)
34	45.522794000	172.16.20.1	172.16.20.254	ICMP	98	Echo (ping) request id=0x69d8, seq=2/512, ttl=64 (reply in 35)
35	45.523007000	172.16.20.254	172.16.20.1	ICMP	98	Echo (ping) reply id=0x69d8, seq=2/512, ttl=64 (request in 34)
36	46.111709000	Cisco_7c:9c:81	Spanning-tree-(for-STP)	60	Conf. Root = 32768/1/00:1e:14:7c:9c:80 Cost = 0 Port = 0x8001	
37	46.522200000	172.16.20.1	172.16.20.254	ICMP	98	Echo (ping) request id=0x69d8, seq=3/768, ttl=64 (reply in 38)
38	46.522566000	172.16.20.254	172.16.20.1	ICMP	98	Echo (ping) reply id=0x69d8, seq=3/768, ttl=64 (request in 37)
39	47.522196000	172.16.20.1	172.16.20.254	ICMP	98	Echo (ping) request id=0x69d8, seq=4/1024, ttl=64 (reply in 40)
40	47.522432000	172.16.20.254	172.16.20.1	ICMP	98	Echo (ping) reply id=0x69d8, seq=4/1024, ttl=64 (request in 39)
41	48.116499000	Cisco_7c:9c:81	Spanning-tree-(for-STP)	60	Conf. Root = 32768/1/00:1e:14:7c:9c:80 Cost = 0 Port = 0x8001	
42	48.522196000	172.16.20.1	172.16.20.254	ICMP	98	Echo (ping) request id=0x69d8, seq=5/1280, ttl=64 (reply in 43)
43	48.522547000	172.16.20.254	172.16.20.1	ICMP	98	Echo (ping) reply id=0x69d8, seq=5/1280, ttl=64 (request in 42)
44	49.522231000	172.16.20.1	172.16.20.254	ICMP	98	Echo (ping) request id=0x69d8, seq=6/1536, ttl=64 (reply in 45)
45	49.522467000	172.16.20.254	172.16.20.1	ICMP	98	Echo (ping) reply id=0x69d8, seq=6/1536, ttl=64 (request in 44)
46	49.528452000	Hewlett-_a7:26:a2	G-ProCom_8b:e4:c5	ARP	60	who has 172.16.20.1? Tell 172.16.20.254

```

Frame 32: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface 0
Ethernet II, Src: G-ProCom_8b:e4:c5 (00:0f:fe:8b:e4:c5), Dst: Hewlett-_a7:26:a2 (00:22:64:a7:26:a2)
Internet Protocol Version 4, Src: 172.16.20.1 (172.16.20.1), Dst: 172.16.20.254 (172.16.20.254)
Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0x20f8 [correct]
    Identifier (BE): 27096 (0x69d8)
    Identifier (LE): 55401 (0xd869)
    Sequence number (BE): 1 (0x0001)
    Sequence number (LE): 256 (0x0100)
    [Response frame: 33]
Timestamp from icmp data: Nov 20, 2014 16:37:43.966190000 Hora padrão de GMT

```

1 - Log resultante do ping do *tux21* para o *tux24*. note-se os pacotes ARP, responsáveis pela informação a inserir na tabela ARP, bem como, neste caso, do tipo de trama ICMP (igual a 8 para *Echo request*)

No.	Time	Source	Destination	Protocol	Length	Info
27	40.517991000	G-ProCom_8b:e4:c5	Broadcast	ARP	42	who has 172.16.20.254? Tell 172.16.20.1
28	40.518346000	Hewlett-_a7:26:a2	G-ProCom_8b:e4:c5	ARP	60	172.16.20.254 is at 00:22:64:a7:26:a2
29	40.518363000	172.16.20.1	172.16.20.254	ICMP	98	Echo (ping) request id=0x32f1, seq=1/256, ttl=64 (reply in 30)
30	40.518621000	172.16.20.254	172.16.20.1	ICMP	98	Echo (ping) reply id=0x32f1, seq=1/256, ttl=64 (request in 29)
31	41.516983000	172.16.20.1	172.16.20.254	ICMP	98	Echo (ping) request id=0x32f1, seq=2/512, ttl=64 (reply in 32)
32	41.517219000	172.16.20.254	172.16.20.1	ICMP	98	Echo (ping) reply id=0x32f1, seq=2/512, ttl=64 (request in 31)
33	42.097821000	Cisco_7c:9c:81	Spanning-tree-(for-STP)	60	Conf. Root = 32768/20/00:1e:14:7c:9c:80 Cost = 0 Port = 0x8001	
34	42.516649000	172.16.20.1	172.16.20.254	ICMP	98	Echo (ping) request id=0x32f1, seq=3/768, ttl=64 (reply in 35)
35	42.517011000	172.16.20.254	172.16.20.1	ICMP	98	Echo (ping) reply id=0x32f1, seq=3/768, ttl=64 (request in 34)
36	43.516632000	172.16.20.1	172.16.20.254	ICMP	98	Echo (ping) request id=0x32f1, seq=4/1024, ttl=64 (reply in 37)
37	43.516846000	172.16.20.254	172.16.20.1	ICMP	98	Echo (ping) reply id=0x32f1, seq=4/1024, ttl=64 (request in 36)
38	44.102695000	Cisco_7c:9c:81	Spanning-tree-(for-STP)	60	Conf. Root = 32768/20/00:1e:14:7c:9c:80 Cost = 0 Port = 0x8001	
39	44.111245000	Cisco_7c:9c:81	Cisco_7c:9c:81	LOOP	60	Reply
40	44.516645000	172.16.20.1	172.16.20.254	ICMP	98	Echo (ping) request id=0x32f1, seq=5/1280, ttl=64 (reply in 41)
41	44.516989000	172.16.20.254	172.16.20.1	ICMP	98	Echo (ping) reply id=0x32f1, seq=5/1280, ttl=64 (request in 40)
42	45.516645000	172.16.20.1	172.16.20.254	ICMP	98	Echo (ping) request id=0x32f1, seq=6/1536, ttl=64 (reply in 43)
43	45.516881000	172.16.20.254	172.16.20.1	ICMP	98	Echo (ping) reply id=0x32f1, seq=6/1536, ttl=64 (request in 42)

```

Frame 29: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface 0
Ethernet II, Src: G-ProCom_8b:e4:c5 (00:0f:fe:8b:e4:c5), Dst: Hewlett-_a7:26:a2 (00:22:64:a7:26:a2)
Internet Protocol Version 4, Src: 172.16.20.1 (172.16.20.1), Dst: 172.16.20.254 (172.16.20.254)
Internet Control Message Protocol

```

2 - Log resultante do ping pelo *tux21*, visto deste. O log resultante, visto do *tux24*, obtém resultado idêntico

2. Experiência 2

No.	Time	Source	Destination	Protocol	Length	Info
87	84.132870000	Cisco_7c:9c:81	Cisco_7c:9c:81	LOOP	60	Reply
88	84.199978000	Cisco_7c:9c:81	Spanning-tree-(for- STP)	60 Conf.	Root = 32768/20/00:1e:14:7c:9c:80 Cost = 0 Port = 0x8001	
89	86.208947000	Cisco_7c:9c:81	Spanning-tree-(for- STP)	60 Conf.	Root = 32768/20/00:1e:14:7c:9c:80 Cost = 0 Port = 0x8001	
90	88.209679000	Cisco_7c:9c:81	Spanning-tree-(for- STP)	60 Conf.	Root = 32768/20/00:1e:14:7c:9c:80 Cost = 0 Port = 0x8001	
91	90.214540000	Cisco_7c:9c:81	Spanning-tree-(for- STP)	60 Conf.	Root = 32768/20/00:1e:14:7c:9c:80 Cost = 0 Port = 0x8001	
92	92.186214000	Cisco_7c:9c:81	CDP/VTP/DTP/PagP/UDCDP	604 Device ID: gnu-sw2 Port ID: FastEthernet0/1	60 Conf.	Root = 32768/20/00:1e:14:7c:9c:80 Cost = 0 Port = 0x8001
93	92.219397000	Cisco_7c:9c:81	Spanning-tree-(for- STP)	60 Conf.	Root = 32768/20/00:1e:14:7c:9c:80 Cost = 0 Port = 0x8001	
94	94.136151000	Cisco_7c:9c:81	Cisco_7c:9c:81	LOOP	60	Reply
95	94.224230000	Cisco_7c:9c:81	Spanning-tree-(for- STP)	60 Conf.	Root = 32768/20/00:1e:14:7c:9c:80 Cost = 0 Port = 0x8001	
96	96.233187000	Cisco_7c:9c:81	Spanning-tree-(for- STP)	60 Conf.	Root = 32768/20/00:1e:14:7c:9c:80 Cost = 0 Port = 0x8001	
97	98.233949000	Cisco_7c:9c:81	Spanning-tree-(for- STP)	60 Conf.	Root = 32768/20/00:1e:14:7c:9c:80 Cost = 0 Port = 0x8001	
98	100.238931000	Cisco_7c:9c:81	Spanning-tree-(for- STP)	60 Conf.	Root = 32768/20/00:1e:14:7c:9c:80 Cost = 0 Port = 0x8001	
99	102.243697000	Cisco_7c:9c:81	Spanning-tree-(for- STP)	60 Conf.	Root = 32768/20/00:1e:14:7c:9c:80 Cost = 0 Port = 0x8001	
100	104.139509000	Cisco_7c:9c:81	Cisco_7c:9c:81	LOOP	60	Reply
101	104.248536000	Cisco_7c:9c:81	Spanning-tree-(for- STP)	60 Conf.	Root = 32768/20/00:1e:14:7c:9c:80 Cost = 0 Port = 0x8001	
102	106.257392000	Cisco_7c:9c:81	Spanning-tree-(for- STP)	60 Conf.	Root = 32768/20/00:1e:14:7c:9c:80 Cost = 0 Port = 0x8001	
103	108.258252000	Cisco_7c:9c:81	Spanning-tree-(for- STP)	60 Conf.	Root = 32768/20/00:1e:14:7c:9c:80 Cost = 0 Port = 0x8001	

3 - Log resultante do ping pelo tux21, visto no tux22

No.	Time	Source	Destination	Protocol	Length	Info
12	17.526046000	172.16.21.1	172.16.21.255	ICMP	98	Echo (ping) request id=0x4bf7, seq=1/256, ttl=64 (no response found!)
13	18.043649000	Cisco_7c:9c:82	Spanning-tree-(for- STP)	60 Conf.	Root = 32768/21/00:1e:14:7c:9c:80 Cost = 0 Port = 0x8002	
14	18.534453000	172.16.21.1	172.16.21.255	ICMP	98	Echo (ping) request id=0x4bf7, seq=2/512, ttl=64 (no response found!)
15	19.542444000	172.16.21.1	172.16.21.255	ICMP	98	Echo (ping) request id=0x4bf7, seq=3/768, ttl=64 (no response found!)
16	20.048521000	Cisco_7c:9c:82	Spanning-tree-(for- STP)	60 Conf.	Root = 32768/21/00:1e:14:7c:9c:80 Cost = 0 Port = 0x8002	
17	20.550477000	172.16.21.1	172.16.21.255	ICMP	98	Echo (ping) request id=0x4bf7, seq=4/1024, ttl=64 (no response found!)
18	21.558452000	172.16.21.1	172.16.21.255	ICMP	98	Echo (ping) request id=0x4bf7, seq=5/1280, ttl=64 (no response found!)
19	22.053338000	Cisco_7c:9c:82	Spanning-tree-(for- STP)	60 Conf.	Root = 32768/21/00:1e:14:7c:9c:80 Cost = 0 Port = 0x8002	
20	22.566425000	172.16.21.1	172.16.21.255	ICMP	98	Echo (ping) request id=0x4bf7, seq=6/1536, ttl=64 (no response found!)
21	23.574441000	172.16.21.1	172.16.21.255	ICMP	98	Echo (ping) request id=0x4bf7, seq=7/1792, ttl=64 (no response found!)
22	24.058237000	Cisco_7c:9c:82	Spanning-tree-(for- STP)	60 Conf.	Root = 32768/21/00:1e:14:7c:9c:80 Cost = 0 Port = 0x8002	
23	24.582429000	172.16.21.1	172.16.21.255	ICMP	98	Echo (ping) request id=0x4bf7, seq=8/2048, ttl=64 (no response found!)
24	25.590433000	172.16.21.1	172.16.21.255	ICMP	98	Echo (ping) request id=0x4bf7, seq=9/2304, ttl=64 (no response found!)
25	26.067211000	Cisco_7c:9c:82	Spanning-tree-(for- STP)	60 Conf.	Root = 32768/21/00:1e:14:7c:9c:80 Cost = 0 Port = 0x8002	
26	26.598487000	172.16.21.1	172.16.21.255	ICMP	98	Echo (ping) request id=0x4bf7, seq=10/2560, ttl=64 (no response found!)
27	26.801210000	Cisco_7c:9c:82	Cisco_7c:9c:82	LOOP	60	Reply
28	27.606432000	172.16.21.1	172.16.21.255	ICMP	98	Echo (ping) request id=0x4bf7, seq=11/2816, ttl=64 (no response found!)

4 - Log resultante do ping pelo tux22 visto no tux22

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	Cisco_7c:9c:81	Spanning-tree-(for- STP)	60 Conf.	Root = 32768/20/00:1e:14:7c:9c:80 Cost = 0 Port = 0x8001	
2	2.008944000	Cisco_7c:9c:81	Spanning-tree-(for- STP)	60 Conf.	Root = 32768/20/00:1e:14:7c:9c:80 Cost = 0 Port = 0x8001	
3	2.793171000	Cisco_7c:9c:81	Cisco_7c:9c:81	LOOP	60	Reply
4	4.009686000	Cisco_7c:9c:81	Spanning-tree-(for- STP)	60 Conf.	Root = 32768/20/00:1e:14:7c:9c:80 Cost = 0 Port = 0x8001	
5	6.014475000	Cisco_7c:9c:81	Spanning-tree-(for- STP)	60 Conf.	Root = 32768/20/00:1e:14:7c:9c:80 Cost = 0 Port = 0x8001	
6	7.943951000	Cisco_7c:9c:81	CDP/VTP/DTP/PagP/UDCDP	604 Device ID: gnu-sw2 Port ID: FastEthernet0/1	60 Conf.	Root = 32768/20/00:1e:14:7c:9c:80 Cost = 0 Port = 0x8001
7	8.019481000	Cisco_7c:9c:81	Spanning-tree-(for- STP)	60 Conf.	Root = 32768/20/00:1e:14:7c:9c:80 Cost = 0 Port = 0x8001	
8	10.024391000	Cisco_7c:9c:81	Spanning-tree-(for- STP)	60 Conf.	Root = 32768/20/00:1e:14:7c:9c:80 Cost = 0 Port = 0x8001	
9	12.033349000	Cisco_7c:9c:81	Spanning-tree-(for- STP)	60 Conf.	Root = 32768/20/00:1e:14:7c:9c:80 Cost = 0 Port = 0x8001	
10	12.800693000	Cisco_7c:9c:81	Cisco_7c:9c:81	LOOP	60	Reply
11	14.033969000	Cisco_7c:9c:81	Spanning-tree-(for- STP)	60 Conf.	Root = 32768/20/00:1e:14:7c:9c:80 Cost = 0 Port = 0x8001	
12	16.038791000	Cisco_7c:9c:81	Spanning-tree-(for- STP)	60 Conf.	Root = 32768/20/00:1e:14:7c:9c:80 Cost = 0 Port = 0x8001	
13	18.043620000	Cisco_7c:9c:81	Spanning-tree-(for- STP)	60 Conf.	Root = 32768/20/00:1e:14:7c:9c:80 Cost = 0 Port = 0x8001	
14	20.048688000	Cisco_7c:9c:81	Spanning-tree-(for- STP)	60 Conf.	Root = 32768/20/00:1e:14:7c:9c:80 Cost = 0 Port = 0x8001	
15	22.057587000	Cisco_7c:9c:81	Spanning-tree-(for- STP)	60 Conf.	Root = 32768/20/00:1e:14:7c:9c:80 Cost = 0 Port = 0x8001	
16	22.812322000	Cisco_7c:9c:81	Cisco_7c:9c:81	LOOP	60	Reply
17	24.058388000	Cisco_7c:9c:81	Spanning-tree-(for- STP)	60 Conf.	Root = 32768/20/00:1e:14:7c:9c:80 Cost = 0 Port = 0x8001	

5 - Log resultante do ping pelo tux22, visto no tux24. O mesmo se observa no tux21

3. Experiência 3

No.	Time	Source	Destination	Protocol	Length	Info
435	350.854415000	CISCO/_C:9C:85	Spanning-tree-(for-STP)	60	Conf.	Root = 32/68/21/00:1e:14:/C:9C:80 Cost = 0 Port = 0x8005
436	350.956494000	G-ProCom_8b:e4:c5	Broadcast	ARP	60	who has 172.16.20.254? Tell 172.16.20.1
437	350.956509000	Hewlett-_a7:26:a2	G-ProCom_8b:e4:c5	ARP	42	172.16.20.254 is at 00:22:64:a7:26:a2
438	350.956870000	Netronix_c8:7c:55	Broadcast	ARP	42	who has 172.16.21.1? Tell 172.16.21.253
439	350.956844000	172.16.20.1	172.16.21.1	ICMP	98	Echo (ping) request id=0x2d4d, seq=1/256, ttl=64 (no response found!)
440	350.957004000	Hewlett-_3c:aa:a1	Netronix_c8:7c:55	ARP	60	172.16.21.1 is at 00:11:0a:3c:aa:a1
441	350.957111000	172.16.20.1	172.16.21.1	ICMP	98	Echo (ping) request id=0x2d4d, seq=1/256, ttl=63 (reply in 442)
442	350.957127000	172.16.21.1	172.16.20.1	ICMP	98	Echo (ping) reply id=0x2d4d, seq=1/256, ttl=63 (request in 441)
443	350.957718000	172.16.20.1	172.16.20.1	ICMP	98	Echo (ping) reply id=0x2d4d, seq=1/256, ttl=64
444	351.957734000	172.16.20.1	172.16.21.1	ICMP	98	Echo (ping) request id=0x2d4d, seq=2/512, ttl=63 (no response found!)
445	352.957728000	172.16.20.1	172.16.21.1	ICMP	98	Echo (ping) request id=0x2d4d, seq=2/512, ttl=64 (reply in 446)
446	351.957899000	172.16.21.1	172.16.20.1	ICMP	98	Echo (ping) reply id=0x2d4d, seq=2/512, ttl=63 (request in 445)
447	351.957885000	172.16.21.1	172.16.20.1	ICMP	98	Echo (ping) reply id=0x2d4d, seq=2/512, ttl=64
448	352.855169000	Cisco_7c:9c:84	Spanning-tree-(for-STP)	60	Conf.	Root = 32/68/20/00:1e:14:/C:9C:80 Cost = 0 Port = 0x8004
449	352.855203000	Cisco_7c:9c:85	Spanning-tree-(for-STP)	60	Conf.	Root = 32/68/21/00:1e:14:/C:9C:80 Cost = 0 Port = 0x8005
450	352.958166000	172.16.20.1	172.16.21.1	ICMP	98	Echo (ping) request id=0x2d4d, seq=3/768, ttl=64 (no response found!)
451	352.958179000	172.16.20.1	172.16.21.1	ICMP	98	Echo (ping) request id=0x2d4d, seq=3/768, ttl=63 (reply in 452)
452	352.958226000	172.16.20.1	172.16.20.1	ICMP	98	Echo (ping) reply id=0x2d4d, seq=3/768, ttl=63 (request in 451)

□ Frame 439: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface 0

Interface id: 0 (eth0)
 Encapsulation type: Ethernet (1)
 Arrival Time: Nov 27, 2014 16:09:33.200031000 Hora padrão de GMT
 [Time shift for this packet: 0.000000000 seconds]
 Epoch Time: 1417104573.200031000 seconds
 [Time delta from previous captured frame: -0.000026000 seconds]
 [Time delta from previous displayed frame: -0.000026000 seconds]
 [Time since reference or first frame: 350.956844000 seconds]
 Frame Number: 439
 Frame Length: 98 bytes (784 bits)
 Capture Length: 98 bytes (784 bits)
 Frame is marked: False

6 - Log resultante do ping pelo tux21, visto na interface eth0 do tux24

No.	Time	Source	Destination	Protocol	Length	Info
435	350.854415000	CISCO/_C:9C:85	Spanning-tree-(for-STP)	60	Conf.	Root = 32/68/21/00:1e:14:/C:9C:80 Cost = 0 Port = 0x8005
436	350.956494000	G-ProCom_8b:e4:c5	Broadcast	ARP	60	who has 172.16.20.254? Tell 172.16.20.1
437	350.956509000	Hewlett-_a7:26:a2	G-ProCom_8b:e4:c5	ARP	42	172.16.20.254 is at 00:22:64:a7:26:a2
438	350.956870000	Netronix_c8:7c:55	Broadcast	ARP	42	who has 172.16.21.1? Tell 172.16.21.253
439	350.956844000	172.16.20.1	172.16.21.1	ICMP	98	Echo (ping) request id=0x2d4d, seq=1/256, ttl=64 (no response found!)
440	350.957004000	Hewlett-_3c:aa:a1	Netronix_c8:7c:55	ARP	60	172.16.21.1 is at 00:11:0a:3c:aa:a1
441	350.957101000	172.16.20.1	172.16.21.1	ICMP	98	Echo (ping) request id=0x2d4d, seq=1/256, ttl=63 (reply in 442)
442	350.957127000	172.16.21.1	172.16.20.1	ICMP	98	Echo (ping) reply id=0x2d4d, seq=1/256, ttl=63 (request in 441)
443	350.957118000	172.16.21.1	172.16.20.1	ICMP	98	Echo (ping) reply id=0x2d4d, seq=1/256, ttl=64
444	351.957734000	172.16.20.1	172.16.21.1	ICMP	98	Echo (ping) request id=0x2d4d, seq=2/512, ttl=63 (no response found!)
445	352.957728000	172.16.20.1	172.16.21.1	ICMP	98	Echo (ping) request id=0x2d4d, seq=2/512, ttl=64 (reply in 446)
446	351.957899000	172.16.21.1	172.16.20.1	ICMP	98	Echo (ping) reply id=0x2d4d, seq=2/512, ttl=63 (request in 445)
447	351.957885000	172.16.21.1	172.16.20.1	ICMP	98	Echo (ping) reply id=0x2d4d, seq=2/512, ttl=64
448	352.855169000	Cisco_7c:9c:84	Spanning-tree-(for-STP)	60	Conf.	Root = 32/68/20/00:1e:14:/C:9C:80 Cost = 0 Port = 0x8004
449	352.855203000	Cisco_7c:9c:85	Spanning-tree-(for-STP)	60	Conf.	Root = 32/68/21/00:1e:14:/C:9C:80 Cost = 0 Port = 0x8005
450	352.958166000	172.16.20.1	172.16.21.1	ICMP	98	Echo (ping) request id=0x2d4d, seq=3/768, ttl=64 (no response found!)
451	352.958179000	172.16.20.1	172.16.21.1	ICMP	98	Echo (ping) request id=0x2d4d, seq=3/768, ttl=63 (reply in 452)
452	352.958226000	172.16.20.1	172.16.20.1	ICMP	98	Echo (ping) reply id=0x2d4d, seq=3/768, ttl=63 (request in 451)

□ Frame 442: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface 0

Interface id: 0 (eth0)
 Encapsulation type: Ethernet (1)
 Arrival Time: Nov 27, 2014 16:09:33.200314000 Hora padrão de GMT
 [Time shift for this packet: 0.000000000 seconds]
 Epoch Time: 1417104573.200314000 seconds
 [Time delta from previous captured frame: 0.000016000 seconds]
 [Time delta from previous displayed frame: 0.000016000 seconds]
 [Time since reference or first frame: 350.957127000 seconds]
 Frame Number: 442
 Frame Length: 98 bytes (784 bits)
 Capture Length: 98 bytes (784 bits)
 Frame is marked: False

7 - Log resultante do ping pelo tux21, visto na interface eth1 do tux24

4. Experiência 4

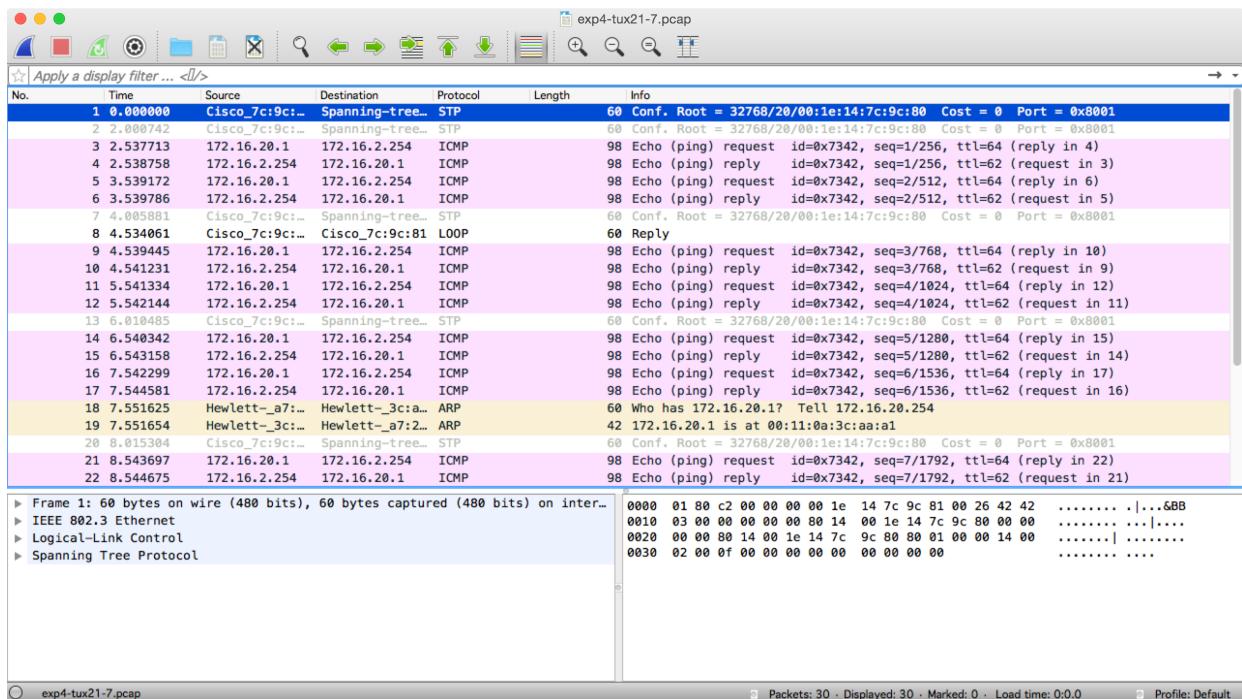
No.	Time	Source	Destination	Protocol	Length	Info
22	27.059248	172.16.20.254	172.16.20.1	ICMP	60	98 Echo (ping) reply id=0x6ff0, seq=3/768, ttl=64 (request in 21)
23	28.067938	Cisco_7c:9c:81	Spanning-tree-...	STP	60	60 Conf. Root = 32768/20/00:le:14:7c:9c:80 Cost = 0 Port = 0x8001
24	28.965499	Cisco_7c:9c:81	COP/FTP/DTP/PAg...	DTP	60	60 Dynamic Trunk Protocol
25	28.965593	Cisco_7c:9c:81	COP/FTP/DTP/PAg...	DTP	60	98 Dynamic Trunk Protocol
26	29.527516	Cisco_7c:9c:81	Cisco_7c:9c:81	LOOP	60	60 Reply
27	30.065722	Hewlett_a7:2...	Hewlett_3c:a...	ARP	60	60 Who has 172.16.20.1? Tell 172.16.20.254
28	30.065734	Hewlett_3c:a...	Hewlett_a7:26:...	ARP	42	42 172.16.20.1 is at 00:11:0a:3c:aa:a1
29	30.072755	Cisco_7c:9c:81	Spanning-tree-...	STP	60	60 Conf. Root = 32768/20/00:le:14:7c:9c:80 Cost = 0 Port = 0x8001
30	31.691814	172.16.20.1	172.16.21.253	ICMP	98	98 Echo (ping) request id=0x66f7, seq=1/256, ttl=64 (reply in 31)
31	31.691981	172.16.21.253	172.16.20.1	ICMP	98	98 Echo (ping) reply id=0x66f7, seq=1/256, ttl=64 (request in 30)
32	32.077611	Cisco_7c:9c:81	Spanning-tree-...	STP	60	60 Conf. Root = 32768/20/00:le:14:7c:9c:80 Cost = 0 Port = 0x8001
33	32.691019	172.16.20.1	172.16.21.253	ICMP	98	98 Echo (ping) request id=0x66f7, seq=2/512, ttl=64 (reply in 34)
34	32.691173	172.16.21.253	172.16.20.1	ICMP	98	98 Echo (ping) reply id=0x66f7, seq=2/512, ttl=64 (request in 33)
35	33.691016	172.16.20.1	172.16.21.253	ICMP	98	98 Echo (ping) request id=0x66f7, seq=3/768, ttl=64 (reply in 36)
36	33.691158	172.16.21.253	172.16.20.1	ICMP	98	98 Echo (ping) reply id=0x66f7, seq=3/768, ttl=64 (request in 35)
37	34.088655	Cisco_7c:9c:81	Spanning-tree-...	STP	60	60 Conf. Root = 32768/20/00:le:14:7c:9c:80 Cost = 0 Port = 0x8001
38	34.225412	Cisco_7c:9c:81	COP/FTP/DTP/PAg...	COP	604	604 Device ID: gnu-sw2 Port ID: FastEthernet0/1
39	36.087396	Cisco_7c:9c:81	Spanning-tree-...	STP	60	60 Conf. Root = 32768/20/00:le:14:7c:9c:80 Cost = 0 Port = 0x8001
40	38.092232	Cisco_7c:9c:81	Spanning-tree-...	STP	60	60 Conf. Root = 32768/20/00:le:14:7c:9c:80 Cost = 0 Port = 0x8001
41	38.522670	172.16.20.1	172.16.21.1	ICMP	98	98 Echo (ping) request id=0x66fb, seq=1/256, ttl=64 (reply in 42)
42	38.522991	172.16.21.1	172.16.20.1	ICMP	98	98 Echo (ping) reply id=0x66fb, seq=1/256, ttl=63 (request in 41)
43	39.523082	172.16.20.1	172.16.21.1	ICMP	98	98 Echo (ping) request id=0x66fb, seq=2/512, ttl=64 (reply in 44)
44	39.523298	172.16.21.1	172.16.20.1	ICMP	98	98 Echo (ping) reply id=0x66fb, seq=2/512, ttl=63 (request in 43)
45	39.535023	Cisco_7c:9c:81	Cisco_7c:9c:81	LOOP	60	60 Reply
46	40.097042	Cisco_7c:9c:81	Spanning-tree-...	STP	60	60 Conf. Root = 32768/20/00:le:14:7c:9c:80 Cost = 0 Port = 0x8001
47	40.523008	172.16.20.1	172.16.21.1	ICMP	98	98 Echo (ping) request id=0x66fb, seq=3/768, ttl=64 (reply in 48)
48	40.523263	172.16.21.1	172.16.20.1	ICMP	98	98 Echo (ping) reply id=0x66fb, seq=3/768, ttl=63 (request in 47)
49	42.105956	Cisco_7c:9c:81	Spanning-tree-...	STP	60	60 Conf. Root = 32768/20/00:le:14:7c:9c:80 Cost = 0 Port = 0x8001
50	44.106732	Cisco_7c:9c:81	Spanning-tree-...	STP	60	60 Conf. Root = 32768/20/00:le:14:7c:9c:80 Cost = 0 Port = 0x8001
51	44.665566	172.16.20.1	172.16.21.254	ICMP	98	98 Echo (ping) request id=0x66ff, seq=1/256, ttl=64 (reply in 52)
52	44.666622	172.16.21.254	172.16.20.1	ICMP	98	98 Echo (ping) reply id=0x66ff, seq=1/256, ttl=254 (request in 51)
53	45.666770	172.16.20.1	172.16.21.254	ICMP	98	98 Echo (ping) request id=0x66ff, seq=2/512, ttl=64 (reply in 54)
54	45.667679	172.16.21.254	172.16.20.1	ICMP	98	98 Echo (ping) reply id=0x66ff, seq=2/512, ttl=254 (request in 53)

8 - Log resultante de *pings* para as diversas interfaces na rede

No.	Time	Source	Destination	Protocol	Length	Info
8	10.024452	Cisco_7c:9c:82	Spanning-t...	STP	60	60 Conf. TC + Root = 32768/21/00:le:14:7c:9c:80 Cost = 0 Port = 0x8002
9	10.913623	Cisco_7c:9c:82	Cisco_7c:9...	LOOP	60	60 Reply
10	12.029197	Cisco_7c:9c:82	Spanning-t...	STP	60	60 Conf. TC + Root = 32768/21/00:le:14:7c:9c:80 Cost = 0 Port = 0x8002
11	14.034052	Cisco_7c:9c:82	Spanning-t...	STP	60	60 Conf. TC + Root = 32768/21/00:le:14:7c:9c:80 Cost = 0 Port = 0x8002
12	15.379763	Hewlett_a5:7...	Broadcast	ARP	42	42 Who has 172.16.21.254? Tell 172.16.21.1
13	15.380362	Cisco_9c:84:b2	Hewlett_5...	ARP	60	60 172.16.21.254 is at 00:1e:7a:9c:84:b2
14	15.380376	172.16.21.1	172.16.20.1	ICMP	98	98 Echo (ping) request id=0x6d96, seq=1/256, ttl=64 (no response found!)
15	15.381141	172.16.21.254	172.16.21.1	ICMP	70	70 Redirect (Redirect for host)
16	15.381163	Hewlett_a5:7...	Broadcast	ARP	42	42 Who has 172.16.21.253? Tell 172.16.21.1
17	15.381274	Netronix_c8:7...	Hewlett_5...	ARP	60	60 172.16.21.253 is at 00:e0:7d:c8:7c:55
18	15.381413	172.16.21.1	172.16.20.1	ICMP	98	98 Echo (ping) request id=0x6d96, seq=1/256, ttl=63 (reply in 19)
19	15.381638	172.16.20.1	172.16.21.1	ICMP	98	98 Echo (ping) reply id=0x6d96, seq=1/256, ttl=63 (request in 18)
20	16.042983	Cisco_7c:9c:82	Spanning-t...	STP	60	60 Conf. TC + Root = 32768/21/00:le:14:7c:9c:80 Cost = 0 Port = 0x8002
21	16.381708	172.16.21.1	172.16.20.1	ICMP	98	98 Echo (ping) request id=0x6d96, seq=2/512, ttl=64 (reply in 22)
22	16.381984	172.16.20.1	172.16.21.1	ICMP	98	98 Echo (ping) reply id=0x6d96, seq=2/512, ttl=63 (request in 21)
23	17.380708	172.16.21.1	172.16.20.1	ICMP	98	98 Echo (ping) request id=0x6d96, seq=3/768, ttl=64 (reply in 24)
24	17.380967	172.16.20.1	172.16.21.1	ICMP	98	98 Echo (ping) reply id=0x6d96, seq=3/768, ttl=63 (request in 23)
25	18.043807	Cisco_7c:9c:82	Spanning-t...	STP	60	60 Conf. TC + Root = 32768/21/00:le:14:7c:9c:80 Cost = 0 Port = 0x8002
26	18.380344	172.16.21.1	172.16.20.1	ICMP	98	98 Echo (ping) request id=0x6d96, seq=4/1024, ttl=64 (reply in 27)
27	18.380610	172.16.20.1	172.16.21.1	ICMP	98	98 Echo (ping) reply id=0x6d96, seq=4/1024, ttl=63 (request in 26)
28	19.380339	172.16.21.1	172.16.20.1	ICMP	98	98 Echo (ping) request id=0x6d96, seq=5/1280, ttl=64 (reply in 29)
29	19.380626	172.16.20.1	172.16.21.1	ICMP	98	98 Echo (ping) reply id=0x6d96, seq=5/1280, ttl=63 (request in 28)

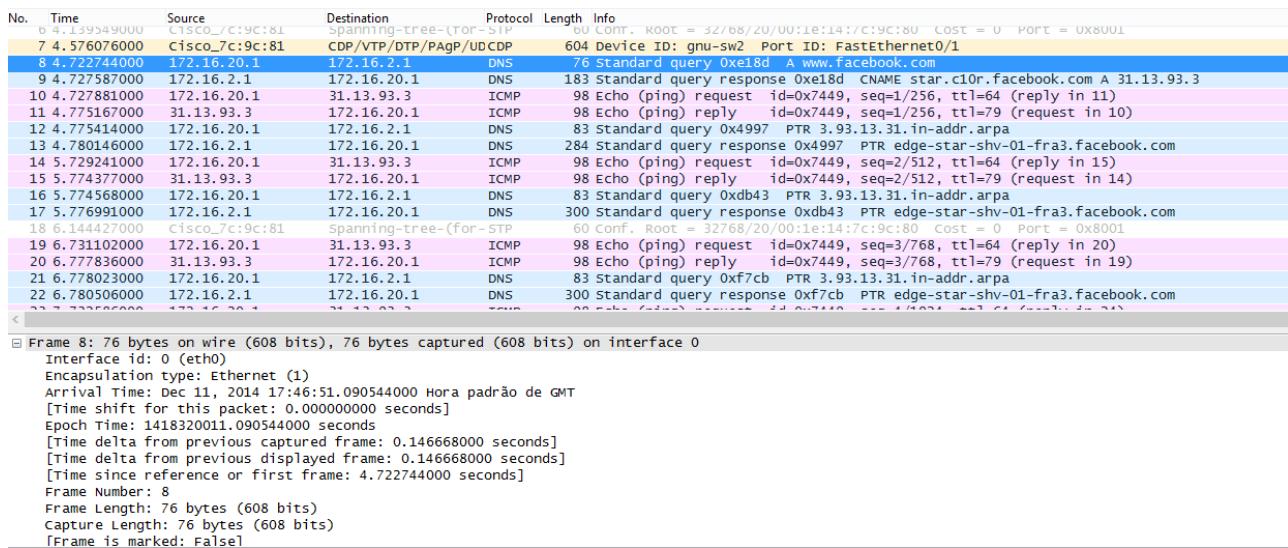
```
0000 01 80 c2 00 00 00 00 1e 14 7c 9c 82 00 26 42 42 .....|...&B
0010 03 00 00 00 01 80 15 00 1e 14 7c 9c 80 00 00 .....|.....
0020 00 00 15 00 1e 14 7c 9c 80 00 00 14 00 .....|.....
0030 02 00 0f 00 00 00 00 00 00 00 00 00 00 00 00 .....|.....
```

9 - Log resultante de *ping* do *tux22* para o *tux21* após se ter removido a rota respectiva



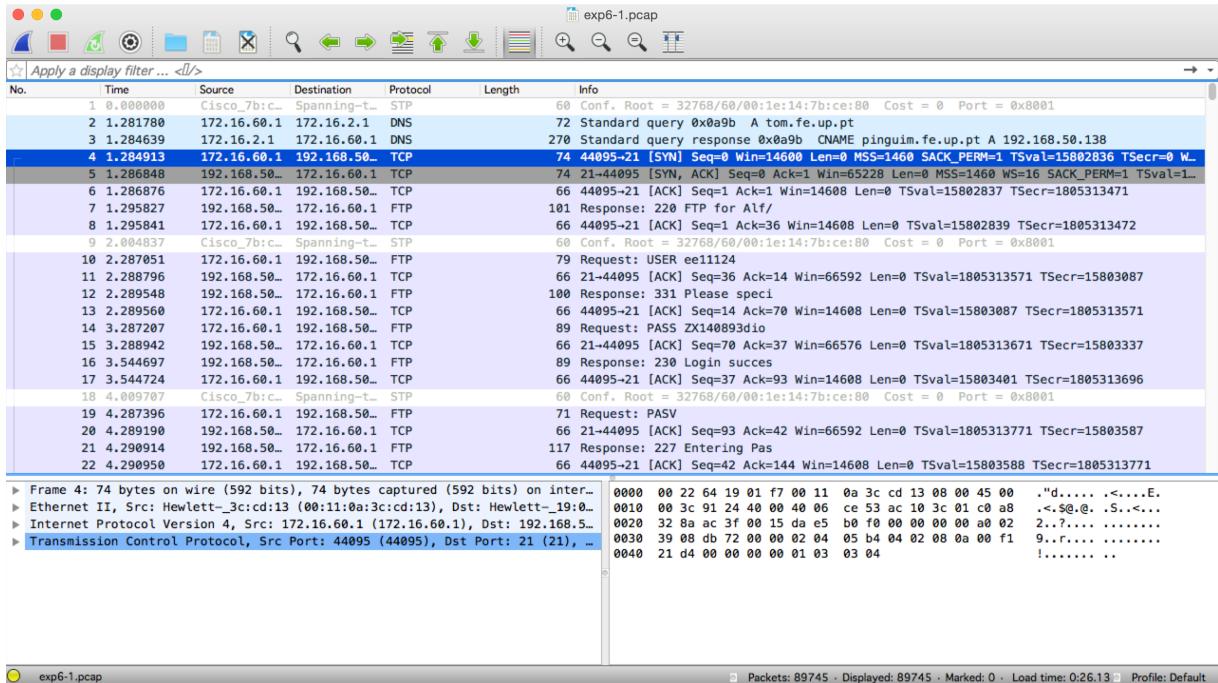
10 - Log resultante de ping do tux21 para a rede do laboratório

5. Experiência 5

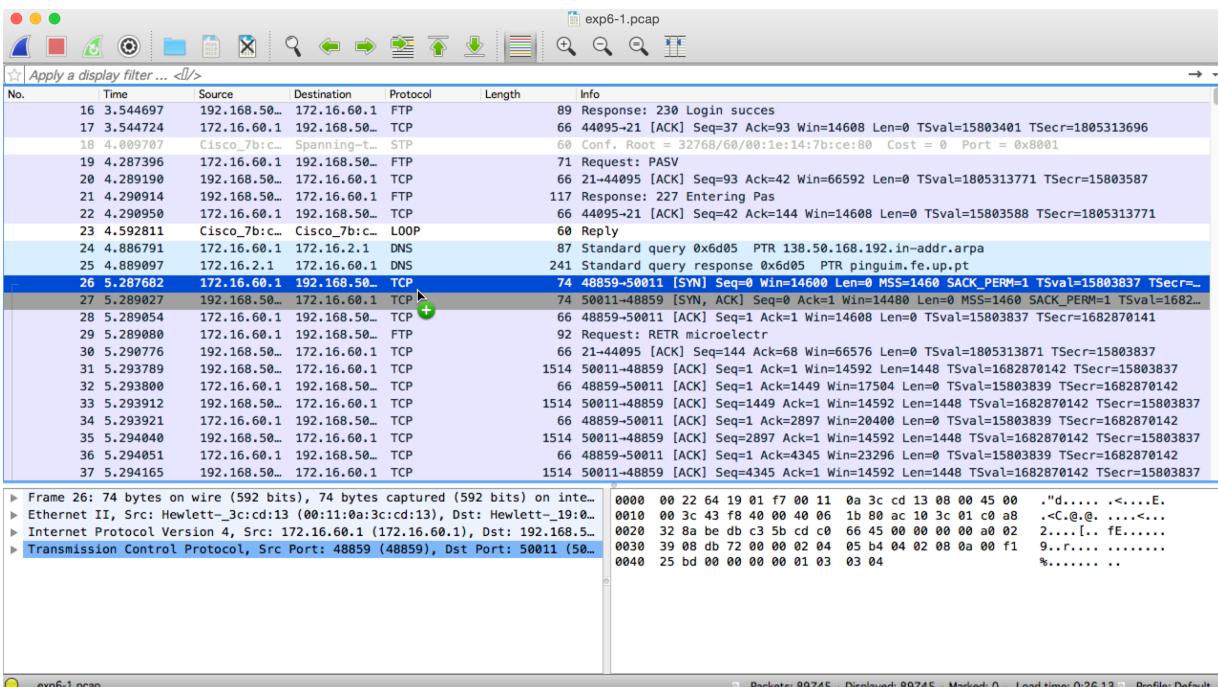


11 - Log resultante de ping a www.facebook.com

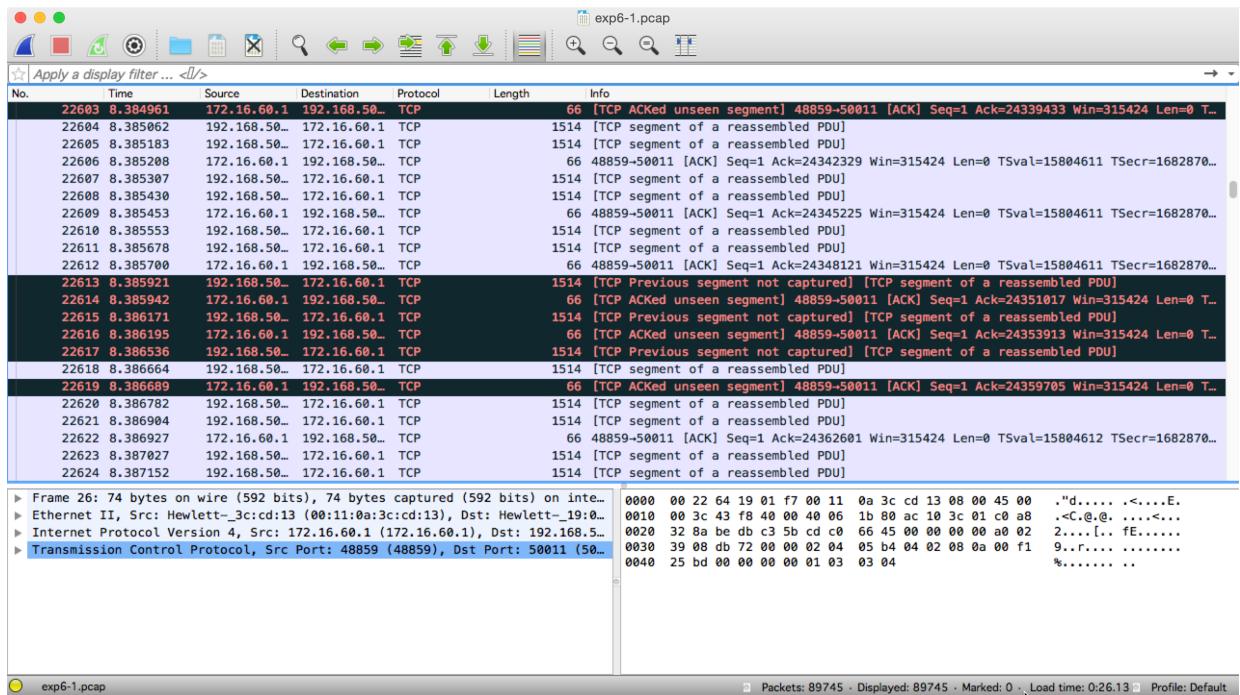
6. Experiência 6



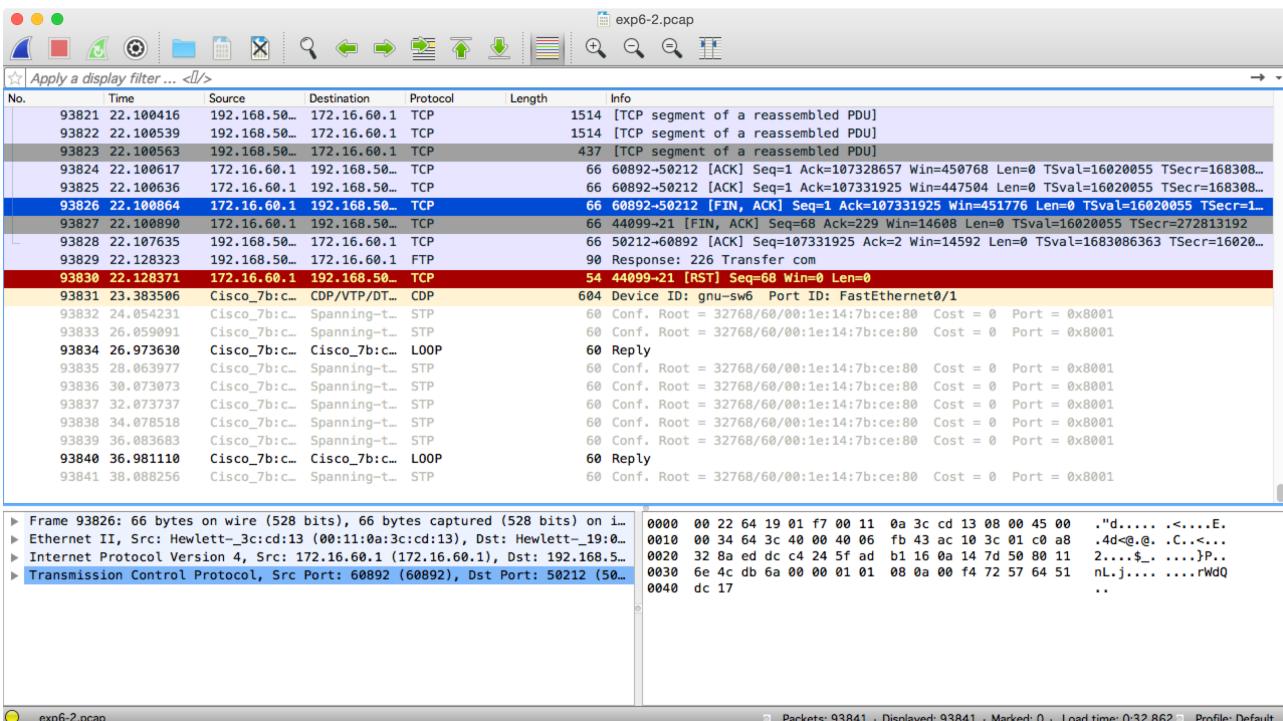
12 - Log resultante da transferência no tux21. Aqui reparamos nas tramas de estabelecimento de uma ligação TCP (neste caso a primeira ligação)



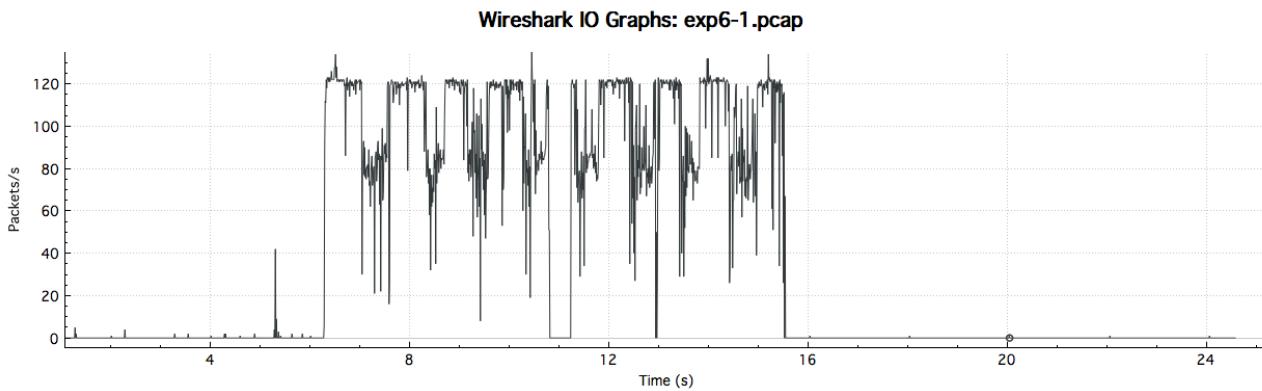
13 - Log resultante da transferência no tux21. Aqui reparamos no estabelecimento da segunda ligação para transferência de dados



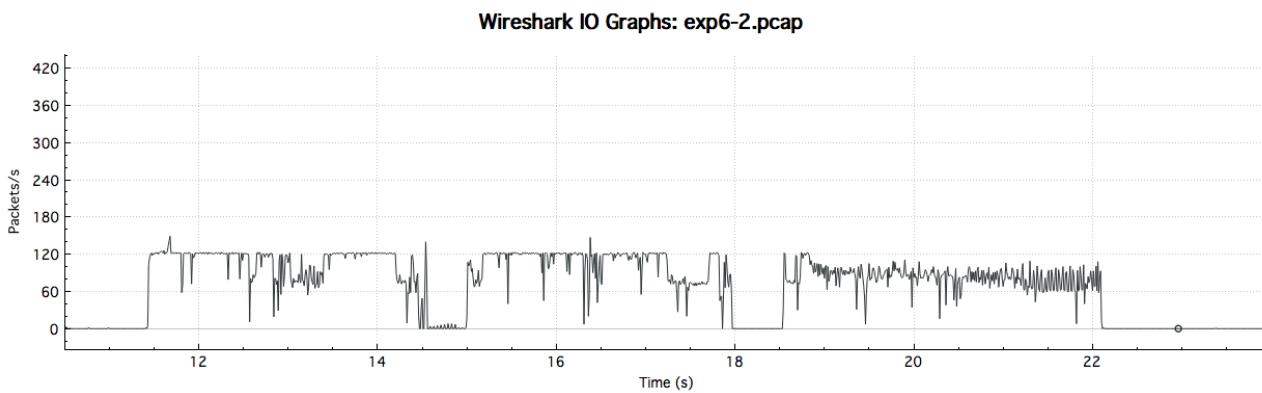
13 - Log resultante da transferência no *tux21*. Aqui pode-se ver a transferência de dados com falhas no envio de várias tramas



14 - Log resultante da transferência no *tux21*. Aqui podemos reparar nas tramas FIN para encerramento da ligação

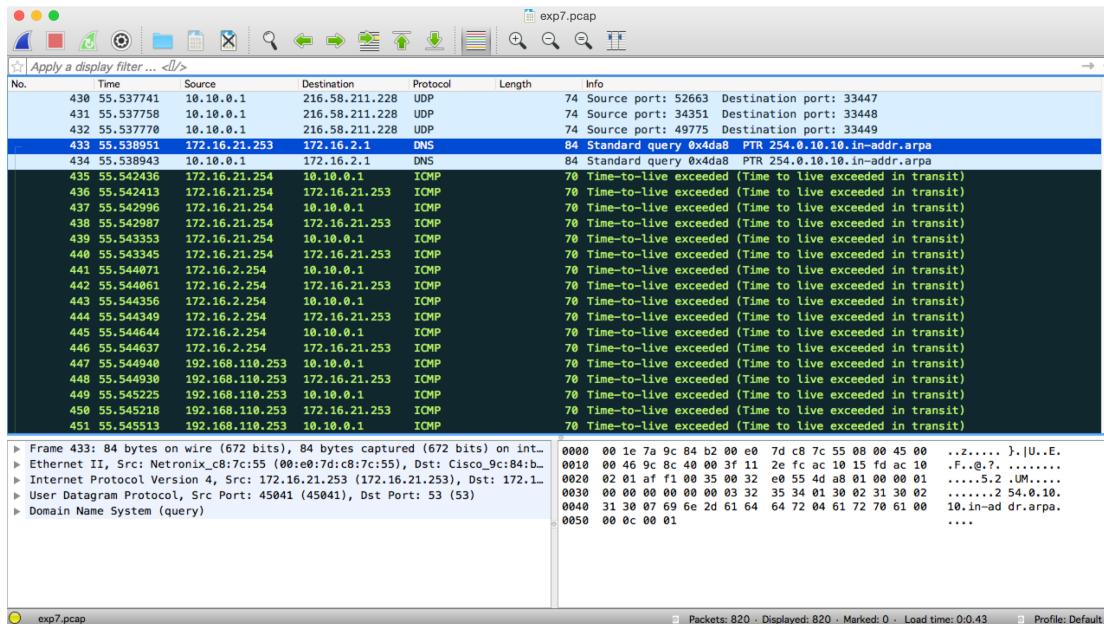


15 - Variação na velocidade de transmissão no *tux21* (sem transferência no *tux22*)



16 - Variação na velocidade de transmissão no *tux21* (com transferência no *tux22*)

7. Experiência 7



17 - Log resultante da criação de tráfego pelo *tux21*

Anexo C - Código da aplicação

```
/* FTP Client – Computer Networks Assignment 2
 * Developed by:
 * Diogo Martins
 * Hugo Fonseca
 * Nuno Pires
 * 2014
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <arpa/inet.h>

#define BUF_SIZE 1024
#define C_PORT "21"

void check_URL(char *url, char *user, char *password, char *host, char *path);
void send_command(char * cmd, char *arg, int sd);
void get_answer(int sd, char * buf);
void get_file(int sd, char * f_name);

int main(int argc, char *argv[])
{
    int sd, sd2, res, ip[4], port[2], i, j;
    char buf[BUF_SIZE];
    char user[64], password[64], host[128], path[512], new_port[10], new_ip[256],
    f_name[128], ip_str[INET6_ADDRSTRLEN];
    struct addrinfo hints, *serv_info, *serv, *aux_p;

    printf("FTP Client\n");
    printf("FEUP – Computer Networks – 2014/2015\n");
    printf("Developed by:\n* Diogo Martins\n* Hugo Fonseca\n* Nuno Pires\n");
    printf("Last compiled on %s at %s\n\n", __DATE__, __TIME__);

    if (argc < 2) {
        printf("Error! URL not specified!\n");
        exit(1);
    }

    check_URL(argv[1], user, password, host, path);
    printf("User: %s\nPass: %s\nHost: %s\nPath: %s\n", user, password, host, path);

    /* Fill up hints struct to pass it to getaddrinfo() */

    memset(&hints, 0, sizeof(hints));
    hints.ai_family = AF_INET;
    hints.ai_socktype = SOCK_STREAM;
    hints.ai_protocol = 0;

    res = getaddrinfo(host, C_PORT, &hints, &serv_info);
    if (res != 0) {
        printf("Error! Couldn't get host information!\n");
        exit(1);
    }

    printf("\nList of IP addresses for %s:\n", host);
    for (aux_p = serv_info; aux_p != NULL; aux_p = aux_p->ai_next) {
        void *addr;
        char *ip_v;

        if (aux_p->ai_family == AF_INET) {
            struct sockaddr_in *ipv4 = (struct sockaddr_in *) aux_p->ai_addr;
            addr = &(ipv4->sin_addr);
        }
    }
}
```

```

        ip_v = "IPv4";
    } else {
        struct sockaddr_in6 *ipv6 = (struct sockaddr_in6 *) aux_p->ai_addr;
        addr = &(ipv6->sin6_addr);
        ip_v = "IPv6";
    }

    inet_ntop(aux_p->ai_family, addr, ip_str, sizeof(ip_str));
    printf("%s: %s\n", ip_v, ip_str);
}

printf("\n");

sd = socket(serv_info->ai_family, serv_info->ai_socktype, serv_info->ai_protocol);
if (sd == -1) {
    printf("Error! Couldn't create socket!\n");
    exit(1);
}

res = connect(sd, serv_info->ai_addr, serv_info->ai_addrlen);
if (res == -1) {
    printf("Error! Couldn't connect to host!\n");
}

/* Server response */

get_answer(sd, buf);

/* USER command */

send_command("USER ", user, sd);
get_answer(sd, buf);

/* PASS command */

send_command("PASS ", password, sd);
get_answer(sd, buf);

/* PASV command */

send_command("PASV", NULL, sd);
get_answer(sd, buf);

if (sscanf(buf, "%*[^()(%d,%d,%d,%d,%d,%d).", &ip[0], &ip[1], &ip[2], &ip[3],
&port[0], &port[1]) != 6) {
    printf("Error! Couldn't read answer!\n");
    exit(1);
}

memset(new_ip, 0, strlen(new_ip));
sprintf(new_ip, "%d.%d.%d.%d", ip[0], ip[1], ip[2], ip[3]);
printf("\nNew IP: %s\n", new_ip);
port[0] = 256*port[0]+port[1];
memset(new_port, 0, strlen(new_port));
sprintf(new_port, "%d", port[0]);
printf("New PORT: %s\n", new_port);

res = getaddrinfo(new_ip, new_port, &hints, &serv);
if (res != 0) {
    printf("Error! Couldn't get host information!\n");
    exit(1);
}

sd2 = socket(serv->ai_family, serv->ai_socktype, serv->ai_protocol);
if (sd2 == -1) {
    printf("Error! Couldn't create socket!\n");
    exit(1);
}

res = connect(sd2, serv->ai_addr, serv->ai_addrlen);
if (res == -1) {

```

```

        printf("Error! Couldn't connect to host!\n");
        exit(1);
    }

/* RETR command */

send_command("RETR ", path, sd);
get_answer(sd, buf);

i = strlen(path) - 1;
while (i > 0) {
    if (path[i] == '/') {
        i++;
        break;
    }

    i--;
}
for (j = 0; i < strlen(path); i++, j++) {
    f_name[j] = path[i];
}
f_name[j] = '\0';

get_file(sd2, f_name);

/* Close connections */

close(sd2);
close(sd);
freeaddrinfo(serv);
freeaddrinfo(serv_info);

printf("Bye!\n");

return 0;
}

/* Using this function the application can cover several cases of valid URL's */

void check_URL(char *url, char *user, char *password, char *host, char *path) {
    if (sscanf(url, "ftp://[%^:]:%[^@]@%[^/]/%[^\\n]", user, password, host, path) == 4)
    {
        return;
    } else if (sscanf(url, "ftp://[%^:@]@%[^/]/%[^\\n]", user, host, path) == 3) {
        strcpy(password, "empty");
        return;
    } else if (sscanf(url, "ftp://[%^:@]/%[^\\n]", host, path) == 2) {
        strcpy(password, "empty");
        strcpy(user, "anonymous");
        return;
    } else {
        printf("Error! URL should be in the following format: ftp://<user>:<password>@<host>/<url-path>\\n");
        exit(1);
    }
}

/* This functions builds the command and sends it through the socket */

void send_command(char *cmd, char *arg, int sd) {
    char command[517];

    if (strcmp(cmd, "PASV") == 0) {
        strcpy(command, cmd);
        strcat(command, "\\n");
        send(sd, command, strlen(command), 0);
        printf("%s", command);
    } else {
        strcpy(command, cmd);
        strcat(command, arg);
        strcat(command, "\\n");
        send(sd, command, strlen(command), 0);
    }
}

```

```

        if (strcmp(cmd, "PASS ") == 0) {
            printf("PASS *****\n");
        } else {
            printf("%s", command);
        }
    }

    memset(command, 0, sizeof(command));
    return;
}

/* This function reads from socket and prints to STDOUT */
/* If the answer code is >= 500 (i.e. error) the app aborts */

void get_answer(int sd, char * buf) {
    int code;

    sleep(1);
    memset(buf, 0, BUF_SIZE);
    recv(sd, buf, BUF_SIZE, 0);
    printf("%s", buf);

    if (sscanf(buf, "%d %*s", &code) == 1) {
        if (code >= 400) {
            printf("Error! Command failed!\n");
            exit(1);
        } else {
            return;
        }
    }
}

/* This functions opens the file, reads from socket and writes the file */

void get_file(int sd, char *f_name) {
    char buf[BUF_SIZE];
    int bytes_written = 1, bytes_read;
    FILE *fd;

    fd = fopen(f_name, "w");
    if (fd == NULL) {
        printf("Error! Couldn't create file!\n");
        exit(1);
    }

    while (bytes_written != 0) {
        bytes_read = recv(sd, buf, BUF_SIZE, 0);
        bytes_written = fwrite(buf, sizeof(char), bytes_read, fd);
    }
    if (fclose(fd) == EOF) {
        printf("Error! Couldn't save file!\n");
        exit(1);
    }

    printf("File saved sucessfully!\n");
    return;
}

```

Excerto de código da versão com comando LIST:

```

(...)

/* PASS command */
send_command("PASS ", password, sd);
get_answer(sd, buf);

/* CWD command */
sleep(1);
if (strcmp(path, "Empty") != 0) {

```

```

    send_command("CWD ", path, sd);
    get_answer(sd, buf);
}

/* PASV command */

send_command("PASV", NULL, sd);
get_answer(sd, buf);
get_new_args(new_ip, new_port, buf);

res = getaddrinfo(new_ip, new_port, &hints, &serv);
if ( res != 0) {
    printf("Error! Couldn't get host information!\n");
    exit(1);
}

sd2 = socket(serv->ai_family, serv->ai_socktype, serv->ai_protocol);
if (sd == -1) {
    printf("Error! Couldn't create socket!\n");
    exit(1);
}

res = connect(sd2, serv->ai_addr, serv->ai_addrlen);
if (res == -1) {
    printf("Error! Couldn't connect to host!\n");
    exit(1);
}

/* LIST Command - Prints directory contents */

send_command("LIST", buf, sd);
get_answer(sd, buf);
get_answer(sd2, buf);

close(sd2);

/* PASV command */

send_command("PASV", NULL, sd);
get_answer(sd, buf);
get_new_args(new_ip, new_port, buf);

res = getaddrinfo(new_ip, new_port, &hints, &serv);
if ( res != 0) {
    printf("Error! Couldn't get host information!\n");
    exit(1);
}

sd2 = socket(serv->ai_family, serv->ai_socktype, serv->ai_protocol);
if (sd == -1) {
    printf("Error! Couldn't create socket!\n");
    exit(1);
}

res = connect(sd2, serv->ai_addr, serv->ai_addrlen);
if (res == -1) {
    printf("Error! Couldn't connect to host!\n");
    exit(1);
}

printf("\nPlease input file to retrieve: ");
fgets(f_name, sizeof(f_name), stdin);
if (f_name[strlen(f_name) - 1] == '\n')
    f_name[strlen(f_name) - 1] = '\0';
send_command("RETR ", f_name, sd);
get_answer(sd, buf);
get_file(sd2, f_name);

(...)
```

```

/* This function gets the IP and Port for the passive connection */

void get_new_args(char *new_ip, char *new_port, char *buf) {
    int ip[4], port[2];

    if (sscanf(buf, "%*d %*s %*s %*s (%d,%d,%d,%d,%d,%d).", &ip[0], &ip[1], &ip[2],
&ip[3], &port[0], &port[1]) != 6) {
        printf("Error! Couldn't read answer!\n");
        exit(1);
    }

    memset(new_ip, 0, strlen(new_ip));
    sprintf(new_ip, "%d.%d.%d.%d", ip[0], ip[1], ip[2], ip[3]);
    printf("\nNew IP: %s\n", new_ip);
    port[0] = 256*port[0]+port[1];
    memset(new_port, 0, strlen(new_port));
    sprintf(new_port, "%d", port[0]);
    printf("New PORT: %s\n", new_port);

    return;
}

(...)
```

```

FTP Client
FEUP - Computer Networks - 2014/2015
Developed by:
* Diogo Martins
* Hugo Fonseca
* Nuno Pires
Last compiled on Dec 20 2014 at 01:53:40

User: anonymous
Pass: empty
Host: ftp.up.pt
Path: pub/openoffice/packages/packages.txt

List of IP addresses for ftp.up.pt:
IPv4: 193.136.37.8

220 Bem-vindo à Universidade do Porto
USER anonymous
331 Please specify the password.
PASS *****
230 Login successful.
PASV
227 Entering Passive Mode (193,136,37,8,67,61)

New IP: 193.136.37.8
New PORT: 17213
RETR pub/openoffice/packages/packages.txt
150 Opening BINARY mode data connection for pub/openoffice/packages/packages.txt
(2380 bytes).
226 File send OK.
File saved sucessfully!
Bye!
diogom93:rcom-tp2 Diogo$
```

1 - Download bem sucedido