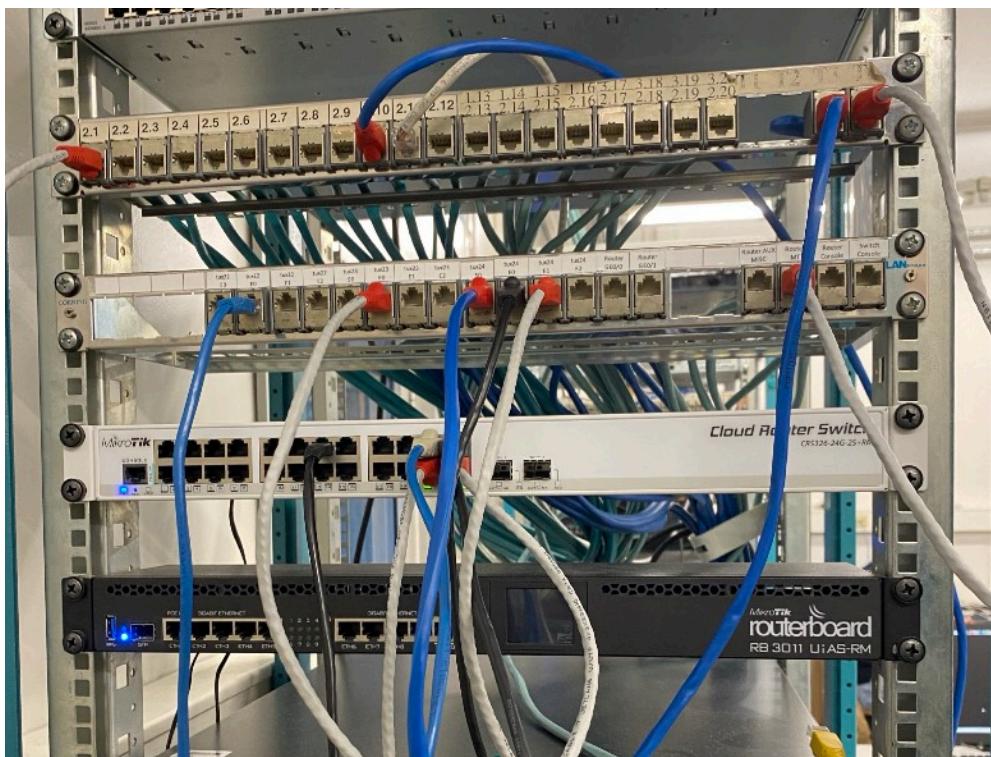




FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Relatório 2º Trabalho Laboratorial Redes de Computadores



Trabalho realizado por:
José Miguel Moreira Isidro
up202006485@fe.up.pt
Turma 12 - Grupo 8

Índice

Sumário	3
Introdução	3
Aplicação de download	3
Arquitetura da aplicação	4
<i>Estruturas de dados</i>	4
<i>Detalhes da implementação</i>	4
<i>Execução</i>	4
Resultados	5
Configuração e análise de redes	6
<i>Experiência 1 - Configuração uma rede IP</i>	6
<i>Experiência 2 - Implementação de duas bridges num switch</i>	8
<i>Experiência 3 - Configuração de um router em Linux</i>	9
<i>Experiência 4 - Configuração de um router comercial e implementação de NAT</i>	11
<i>Experiência 5 - DNS</i>	13
<i>Experiência 6 - Conexões TCP</i>	13
Conclusões	15
Referências	15
Anexos	16
<i>Aplicação download</i>	16
<i>Logs das experiências</i>	30

Sumário

Este trabalho foi realizado no âmbito da unidade curricular Redes de Computadores (RCOM) do 3º ano da Licenciatura em Engenharia Informática e Computação. O relatório incide sobre o segundo trabalho laboratorial, cujo focos foram o desenvolvimento de uma aplicação de transferência de um ficheiro e a configuração de uma rede de computadores através de um conjunto de experiências realizadas em laboratório.

Introdução

Este trabalho tem dois objetivos principais:

- Desenvolver uma aplicação para efetuar a transferência de um único ficheiro, através do protocolo FTP.
- Configurar uma rede de computadores, através de um conjunto de experiências realizadas em laboratório.

Quanto ao primeiro objetivo, a aplicação foi desenvolvida na linguagem C, sendo capaz de transferir um ficheiro do modo pretendido e detetar alguns possíveis erros durante este processo.

Em relação ao segundo objetivo, as experiências foram realizadas com sucesso, tendo as principais questões relativas a estas sido respondidas.

Aplicação de download

A primeira parte deste trabalho consistiu no desenvolvimento de uma aplicação de transferência de um ficheiro de acordo com o protocolo FTP (File Transfer Protocol) descrito no RFC959 e com ligações TCP (Transmission Control Protocol) a partir de sockets, sendo assim a aplicação capaz de fazer o download de qualquer tipo de ficheiro de um servidor FTP. A aplicação aceita como argumento um URL com a seguinte configuração:

`ftp://[<user>:<password>@]<host>/<url-path>`

Este URL está de acordo com a sintaxe descrita no RFC1738. Os parênteses retos representam um campo opcional. Foi necessário estudar com cuidado tanto o RFC959 como o RFC1738 para compreender o protocolo FTP e o tratamento de informações provenientes de URL's.

Arquitetura da aplicação

O código encontra-se distribuído por dois ficheiros de modo a ficar mais organizado e legível. No ficheiro principal, main.c, são chamadas funções auxiliares que foram implementadas no ficheiro app.c.

Estruturas de dados

```
struct FTPparameters {
    char user[MAX_LENGTH]; /* User string */
    char password[MAX_LENGTH]; /* Password string */
    char host_name[MAX_LENGTH]; /* Host name string */
    char file_path[MAX_LENGTH]; /* File path string */
    char file_name[MAX_LENGTH]; /* File name string */
};

struct FTP {
    int control_socket_fd; /* File descriptor to control socket */
    int data_socket_fd; /* File descriptor to data socket */
};
```

Detalhes da implementação

- Sempre que se envia comandos ao servidor, estes são enviados para um *control socket*;
- Para receber um ficheiro, é criado um novo socket para receber os dados, este é o *data socket*;
- Nas funções que enviam comandos ao *control socket*, estes são enviados em minúsculas, já que o protocolo também aceita este formato como indicado em RFC959.

Execução

Em primeiro lugar, é necessário fazer o parsing do url indicado pelo utilizador. Para este efeito, foi implementada uma função chamada **parseArguments** que guarda os valores resultantes do parsing (username, password, nome do host, path do ficheiro, nome do ficheiro) numa estrutura de dados do tipo **FTPparameters**.

Em seguida utiliza-se uma função chamada **getIPAddress** para a qual passamos o nome do host e que retorna o endereço IP correspondente. Com este endereço IP utilizamos a função **openAndConnectSocket** para conectar com o socket, sendo utilizada a porta 21 para o efeito.

Para enviar comandos e interpretar respostas do socket é utilizada a função **sendCommandHandleReply** que por sua vez utiliza a função **sendCommandToControlSocket** para enviar comandos para o socket e a função **readReplyFromControlSocket** para ler as respostas deste mesmo socket.

Esta função também faz o processamento da resposta baseando-se no primeiro dígito da resposta:

- Caso o primeiro dígito seja 1, 2 ou 3 a resposta é de natureza “positiva”, caso seja 4 ou 5 a resposta é de natureza “negativa”;
- Caso o primeiro dígito seja 1, então se se estiver a receber um ficheiro retorna, senão a função **readReplyFromControlSocket** é chamada novamente e a resposta volta a ser interpretada. Isto deve-se ao facto da primeira resposta ao comando *RETR* ser 150 o que indica que o ficheiro vai começar a ser transferido, pelo que neste caso a resposta é a esperada;
- Caso seja 2 ou 3, a função retorna, e, caso seja 4, tenta reenviar comando e ler a resposta;
- Caso seja 5, ocorreu um erro ao enviar o comando, pelo que o socket é fechado e a aplicação termina.

Depois de aberto o socket é agora necessário fazer o login pelo que serão enviados os comandos *USER* ‘user’ e *PASS* ‘pass’, respetivamente, o que é realizado pela função **login**.

Posteriormente, caso seja necessário mudar o diretório atual no servidor para o diretório onde se encontra o ficheiro que se pretende fazer o download, é então chamada a função **changeWorkingDirectory**, que envia o comando *CWD* ‘file path’ para o servidor.

Em seguida, é ativado o modo passivo através do envio do comando *PASV*, que retorna um endereço IP e a porta onde irá ser aberto um novo socket para receber o ficheiro. Tanto o envio do comando *PASV* bem como a abertura deste novo socket são feitos pela função **enablePassiveMode**.

Para fazer a transferência do ficheiro é preciso enviar o comando *RETR* ‘file name’ para o novo socket (*data socket*) e só depois podemos fazer o download do ficheiro. Para este efeito, foi implementada a função **retrieveFile** que envia o comando *RETR* para o servidor e depois cria um novo ficheiro onde guarda o conteúdo recebido através do data socket.

Por fim, é utilizada a função **disconnectFromSocket** que efetua o fecho da ligação com o servidor através do envio do comando *QUIT* para o control socket.

Resultados

A aplicação funcionou como esperado e cumpriu com todos os requisitos iniciais, tanto do protocolo FTP, descritos em RFC959, como os da sintaxe descrita no RFC1738. Deste modo, a aplicação funciona quer no caso do utilizador e password serem passados, quer no caso em que não o são, pelo que o login é então feito com user ‘anonymous’ e password vazia. Para além disso, foram testados vários tipos de ficheiros com tamanhos variados e também em diferentes servidores, sucedendo em todos os cenários.

(Ver Figuras 1 e 2 nos anexos)

Configuração e análise de redes

Nota: as experiências foram realizadas na bancada 2, pelo que tuxYZ corresponderá a tux2Z.

Experiência 1 - Configuração uma rede IP

O objetivo desta experiência foi a ligação de duas máquinas através de um switch, o tux23 e o tux24, sendo que foi necessária a atribuição de um endereço IP a cada uma delas. Foi também observada a transmissão de mensagens (pings) entre elas.

Quais os comandos necessários para a configuração desta experiência?

- Para a atribuição de um endereço IP a uma máquina: ifconfig <cartaDeRede> <endereçoIP>/<mascara> (ex: ifconfig eth0 172.16.30.1/24)
- Para visualizar as rotas existentes : route -n
- Para visualizar as entradas na tabela ARP: arp -a
- Para o envio de um ping: ping <destino>
- Para eliminar um registo da tabela ARP: arp -d <endereçoIP>

(Os comandos acima podem ser consultados no slide 46 do guião)

O que são os pacotes ARP e para que são usados?

O protocolo ARP (Address Resolution Protocol) tem como objetivo o mapeamento de um endereço IP de uma máquina a um endereço MAC da máquina na rede local.

Um computador, ao tentar enviar um pacote a outro, na mesma rede local (admitindo que a tabela ARP não tem entradas para o IP da máquina que recebe), irá enviar um pacote ARP, por broadcast, para todas as máquinas da rede local, perguntando qual das máquinas tem um endereço MAC que corresponde ao endereço IP do destinatário.

O destinatário irá enviar outro pacote ARP que indica à máquina, que enviou o primeiro pacote, qual o seu endereço MAC.

Deste modo, a transferência de pacotes pode ser efetuada, uma vez que os endereços MAC são agora conhecidos.

Quais são os endereços MAC e IP dos pacotes ARP, e porquê?

Quando o tux23 tenta enviar um pacote ao tux24, após apagar a entrada da tabela ARP referente ao tux24, o tux23 não sabe qual é o endereço MAC associado ao endereço IP do tux24 (172.16.20.254).

Deste modo, irá enviar um pacote ARP em broadcast (para toda a rede local), sendo que este pacote contém o endereço IP (172.16.20.1) e o endereço MAC (00:21:5a:5a:7d:12) do tux23. O endereço MAC do destinatário, como não é conhecido, tem o valor de 00:00:00:00:00:00.

De seguida, o tux24 irá enviar um pacote ARP para o tux23, com os seus endereços MAC (00:22:64:a6:a4:f1) e IP (172.16.20.254).

Portanto, pode-se concluir que cada pacote ARP contém campos para os endereços MAC e IP, da máquina que envia, e campos para os endereços MAC e IP, da máquina que recebe.

Que pacotes gera o comando ping?

O comando ping gera primeiro pacotes ARP para saber qual o endereço MAC do destinatário, tal como foi explicado anteriormente. De seguida, gera pacotes ICMP (Internet Control Message Protocol).

Quais são os endereços MAC e IP dos pacotes ping?

Quando é efetuado um ping do tux23 para o tux24, os endereços dos pacotes enviados e recebidos estão representados na tabela a seguir.

Pacote	Endereço IP (origem)	Endereço MAC (origem)	Endereço IP (destino)	Endereço MAC (destino)
Request (de tux23 para tux24)	172.16.20.1	00:21:5a:5a:7d:12	172.16.20.254	00:22:64:a6:a4:f1
Reply (de tux24 para tux23)	172.16.20.254	00:22:64:a6:a4:f1	172.16.20.1	00:21:5a:5a:7d:12

Como determinar se uma trama Ethernet é ARP, IP, ICMP?

Conseguimos determinar o tipo da trama recebida analisando o Ethernet Header da trama:

- Caso este tenha o valor 0x0800, quer dizer que a trama é do tipo IP.
- Se tiver o valor 0x0806, então a trama é do tipo ARP.

Caso a trama seja do tipo IP, podemos analisar o seu IP Header. Se este header tomar o valor 1, então o tipo de protocolo é ICMP.

Como determinar o tamanho de uma trama recebida?

O comprimento de uma trama recetora pode ser visualizado através do programa Wireshark.

O que é a interface loopback e qual a sua importância?

A interface loopback é uma interface virtual da rede que permite ao computador receber respostas de si próprio, para testar a correta configuração da carta de rede.

Experiência 2 - Implementação de duas bridges num switch

O objetivo da segunda experiência foi a configuração de duas bridges num switch, a bridge20 e bridge21, sendo que duas máquinas foram ligadas à primeira, tux23 e tux24, e uma máquina foi ligada à segunda, a tux22. Foi novamente observada a transmissão de mensagens (pings) entre elas, de modo a melhor compreender como é que estas bridges influenciam a troca de informação entre as máquinas.

Quais os comandos necessários para a configuração desta experiência?

Criação e configuração das bridges (slide 50), em mais detalhe na resposta abaixo.

Como configurar a bridgeY0?

1. Ligação ao switch:

- Efetuar a ligação pela porta série /dev/ttyS0, para poder aceder ao switch através do GTKTerm.
- Inserir as credencias de acesso para o switch, no caso da Mikrotik:
 - “admin”
 - “”

2. Criação das bridges

- /interface bridge add name=bridge20
- /interface bridge add name=bridge21
- /interface bridge print (opcional, imprimir as bridges)

Nota: as interfaces indicadas a seguir dependem de onde os cabos foram conectados no switch,. Na minha experiência, as máquinas tux23, tux24 e tux22 foram ligadas às entradas ether1, ether2 e ether3, respetivamente.

3. Remover as portas conectadas à bridge padrão

- /interface bridge port remove [find interface=ether1] (tux23)
- /interface bridge port remove [find interface=ether2] (tux24)
- /interface bridge port remove [find interface=ether3] (tux22)
- /interface port print brief (opcional, imprimir bridges e os ports)

4. Para adicionar as novas portas na bridges correspondentes

- /interface bridge port add bridge=bridge20 interface=ether1 (tux23)
- /interface bridge port add bridge=bridge20 interface=ether2 (tux24)
- /interface bridge port add bridge=bridge21 interface=ether3 (tux22)
- /interface port print brief (opcional, imprimir bridges e os ports)

5. Para dar reset à configuração do switch, no fim do procedimento

- /system reset-configuration ('y' seguido de um [ENTER])

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

Existem 2 domínios de broadcast, uma vez que ao fazer broadcast (ping -b <destino>) apenas abrange as portas pertencentes a essa bridge, tal como foi visto na experiência e reiterado pelos logs do Wireshark.

Experiência 3 - Configuração de um router em Linux

Nesta experiência, o tux24 foi configurado de modo a funcionar como um *router*, que ao estar ligado a ambas as *bridges*, permite a transmissão de informação entre máquinas em bridges diferentes (tux21 e tux22). Assim como nas experiências anteriores, foi novamente observada a transmissão de mensagens (pings) entre as diferentes máquinas.

Quais os comandos necessários para a configuração desta experiência?

- Para a atribuição de um endereço IP a uma máquina: ifconfig <cartaDeRede> <endereçoIP>/<mascara> (ex: ifconfig eth0 172.16.30.1/24)
 - Para visualizar as rotas existentes : route -n
 - Para o envio de um ping: ping <destino>
 - Para a configuração de rotas: route add -net <destino>/<mask> gw <endGateway>
 - Para ativar o IP forwarding: sysctl net.ipv4.ip_forward=1
 - Para desativar ignore broadcast: sysctl net.ipv4.icmp_echo_ignore_broadcasts=0
- (Os comandos acima podem ser consultados nos slides 46 e 47 do guião)

Que rotas existem no tuxes? Qual o seu significado?

Algumas rotas são geradas automaticamente, conectando as máquinas às bridges a que pertencem: o tux23 tem uma rota para a bridge20, o tux22 tem uma rota para a bridge21, e o tux24 tem rota para as duas. O gateway para estas rotas é 0.0.0.0.

Para além disso, no tux23 adicionou-se a rota seguinte:

```
route add -net 172.16.21.0/24 gw 172.16.20.254
```

Assim, quando o tux23 (172.16.20.1) pretender enviar um ping para a bridge21 (172.16.21.0), este vai utilizar como gateway o tux24 (172.16.20.254) que está a funcionar como um router.

No tux24 fez-se algo similar ao que se fez no tux23, só que no sentido inverso. Criou-se a rota:
route add -net 172.16.20.1/24 gw 172.16.21.253

Ou seja, quando o tux24 pretender enviar um ping para a bridge20 envia primeiro para o router (172.16.21.253).

Que informação contém uma entrada da forwarding table?

Na tabela de forwarding cada entrada possuí informação do tipo [Destination - Gateway - Interface], em que a destination é o IP do computador, o gateway é o IP do computador para o qual vai ser mandada a mensagem para fazer o routing para o destino, e a interface é a placa de rede usada para enviar a mensagem exemplo: eth0, eth1, etc.

Outras informações que também estão presentes para cada entrada na tabela são:

- Netmask - utilizado para determinar o ID da rede a partir do endereço IP do destino;
- Flags - informações sobre a rota;
- Metric - custo da rota;
- Ref - número de referências para esta rota (não usado no kernel do linux);
- Use - contador de pesquisas pela rota, dependente do uso de -F ou -C (número de falhas da cache e número de sucessos, respectivamente).

Que mensagem ARP, e endereços MAC associados, são observados, e porquê?

O tux23 envia uma mensagem ARP, na qual pede o endereço MAC de tux22, através do seu IP. Quando uma mensagem ARP é enviada, o tux que envia, neste caso, o tux23, associa o seu endereço MAC à mensagem, para que o receptor esperado da mensagem, neste caso, o tux2, saiba a que tux responder ("Who has [IP de tux22]? Tell [próprio IP]"). Esta mensagem será enviada no modo broadcast (endereço MAC do receptor tem o valor 00:00:00:00:00:00), porque o endereço MAC de tux22 ainda é desconhecido. Quando recebe este broadcast, tux22 responde com uma mensagem ARP, na qual fornece o seu endereço MAC ("[IP de tux22] is at [endereço MAC de tux22]"). Esta mensagem é enviada apenas para o endereço MAC de tux23 e não em modo broadcast.

Esta troca de mensagens ARP ocorre sempre que uma mensagem é enviada de uma máquina para outra, onde os endereços MAC não são conhecidos.

Que pacotes ICMP são observados, e porquê?

Podemos observar pacotes ICMP do tipo *Request* e *Reply*, uma vez que todas as rotas foram adicionadas, pelo que todos os tuxes reconhecem a presença uns dos outros. Se não reconhecessem, os pacotes ICMP enviados seriam do tipo "Host Unreachable".

Quais são os endereços IP e MAC associados aos pacotes ICMP, e porquê?

Os endereços IP e MAC de origem e destino associados com os pacotes ICMP são os endereços MAC e IP das máquinas/interfaces que recebem/enviam os pacotes. Por exemplo, quando um pacote ICMP é enviado do tux23 para a interface do tux24 conectada à mesma subrede (no nosso caso foi utilizada a interface eth0), os endereços MAC e IP de origem serão os do tux23 (IP: 172.16.20.1 e MAC: 00:21:5a:5a:7d:12) e os endereços IP e MAC de destino serão os associados à interface eth0 do tux24 (IP: 172.16.20.254 e MAC: 00:22:64:a6:a4:f1).

Se as duas máquinas não estiverem conectadas à mesma rede virtual, como é o caso do tux23 e tux22, não é possível efetuar o envio de informação entre as duas máquinas com apenas um pacote ICMP sem este ser modificado.

Neste caso, o tux23 irá enviar um pacote ICMP para a interface eth0 do tux24, uma vez que o tux24 está ligado a ambas as bridges e consegue interagir diretamente com o tux22. Assim, os endereços IP do pacote serão os IPs do tux23 e tux22 (origem e destino, respectivamente), e os endereços MAC serão os do tux23 e o da interface eth0 do tux24 (00:22:64:a6:a4:f1).

Ao receber este pacote, o tux24 irá redirecioná-lo para o tux22, mantendo os endereços IP do pacote, mas os endereços MAC associados a este serão diferentes: o de origem será o da interface eth1 do tux24 (00:08:54:50:3f:2c), já que esta interface está conectada à bridge onde se encontra o tux22, e o de destino será o endereço MAC do tux22.

Experiência 4 - Configuração de um router comercial e implementação de NAT

A quarta experiência teve como objetivo a configuração de um router comercial, efetuando a ligação deste à rede do laboratório (172.16.1.0/24) e também à bridge21. Por defeito, a configuração do router implementa NAT para garantir a conexão entre as máquinas rede IP e a internet. O objetivo da experiência foi compreender de que maneira é que o NAT influencia a conexão entre as máquinas e a Internet e, também, analisar o mecanismo de redirecionamento de pacotes ICMP.

Quais os comandos necessários para a configuração desta experiência?

Configuração do router e da NAT (slide 51 do guião).

Como configurar uma rota estática num router comercial?

1. Ligação ao router:

- Efetuar a ligação pela porta série /dev/ttyS0, para poder aceder ao router através do GTKTerm.
- Inserir as credencias de acesso para o switch, no caso da Mikrotik:
 - “admin”
 - “”

2. Configuração das interfaces

```
/ip address add address=172.16.1.29/24 interface=ether1
/ip address add address=172.16.21.254/24 interface=ether2
/ip address print (opcional, para imprimir as interface)
```

3. Configuração das rotas

```
/ip route add dst-address=0.0.0.0/0 gateway=172.16.1.254
/ip route add dst-address=172.16.20.0/24 gateway=172.16.21.253
/ip route print (opcional, para imprimir as rotas)
```

Quais são os caminhos seguidos pelos pacotes nas experiências efetuadas, e porquê?

No caso de uma rota existir, os pacotes seguem essa mesma rota. Caso contrário, os pacotes são dirigidos ao router pela rota *default*.

No passo 4 da experiência, os redirecionamentos são desativados através dos comandos a seguir (executados no tux22):

```
sysctl net.ipv4.conf.eth0.accept_redirects=0
sysctl net.ipv4.conf.all.accept_redirects=0
```

Deste modo, caso existam redirects na mesma interface de rede, o tux não guarda na sua lista de forwarding uma entrada resultante do redirect de um outro tux.

Para além disso, foi definido, tanto para o tux24, como para o tux22, como rota *default* o router Rc: “route add default gw 172.16.21.254”. No tux23, foi adicionada também como rota *default* o tux24: “route add default gw 172.16.20.254”. O tux24 é o único que tem comunicação com o tux23 através da bridge20. O tux24, o tux22 e o router estão todos ligados à bridge21. O router é o único que sabe que para chegar ao tux23 é necessário utilizar como gateway o tux24, para isso adicionou-se a seguinte route ao router ”/ip route add dst-address=172.16.20.0/24 gateway=172.16.21.253”, como mencionado acima.

Quando se envia um ping para o tux23 a partir do tux22 como este não tem conhecimento da rota para o tux23 e eles não estão na mesma rede, o tux22 vai, em primeiro lugar, enviar o ping para o router, porque este é a sua rota *default*. Depois, como este tem conhecimento da rota para o tux23, envia para o tux24 e este redireciona finalmente para o tux23 o ping.

Nos seguintes pings o caminho vai ser sempre o mesmo. Caso os redirects estejam ativos apenas no primeiro ping é que vai haver um redirecionamento do ping para o router, depois vai haver um redirect do router para o tux22 e este vai guardar na sua tabela de forwarding que para chegar ao tux23 pode utilizar o tux24 como gateway, depois do primeiro ping não existem mais redirects, uma vez que o tux22 envia logo para o tux24 o ping invés de passar pelo router.

Como configurar o NAT num router comercial?

Assumindo que a configuração anterior dos endereços IP e rotas foi feita, podemos então configurar o NAT. No caso do route da Mikrotik, este já vem implementado por defeito. Contudo, podemos escolher desativá-lo com o comando seguinte:

```
/ip firewall nat disable 0  
(para o reativar, o comando é: /ip firewall nat enable 0)
```

E podemos adicionar regras específicas, por exemplo:

```
/ip firewall nat add chain=srcnat action=masquerade out-interface=ether1
```

O que faz o NAT?

O NAT (Network Address Translation) é um protocolo cuja função é a associação e transformação de um endereço IP noutro endereço IP, de forma a “mascarar” o remetente/destinatário de pacotes enviados.

Tem várias finalidades, como assegurar a privacidade e segurança de máquinas numa subrede privada local, que estão a comunicar com máquinas fora dessa subrede, conservando os seus endereços IPs. Permite também que essa comunicação seja possível, já que possibilita que redes IP privadas, que usem endereços não registados, se conectem e comuniquem com a Internet ou outras redes públicas. As máquinas dessa rede, pelas máquinas exteriores, são reconhecidas através de um endereço IP único, que representa todos os dispositivos da mesma.

O NAT opera num router, como o que foi configurado e utilizado na experiência 4. Esse router estabelece o ponto de ligação entre a rede local e a Internet/rede pública. No caso de uma máquina da rede local, por exemplo o tux23, querer enviar um pacote para um endereço numa rede pública (172.16.1.254, como foi feito na experiência), o pacote é enviado para o router e este modifica o endereço de origem do pacote para o seu endereço exterior (172.16.1.29), “mascarando” assim o verdadeiro remetente do pacote (tux23) e assegurando a sua privacidade e segurança.

O pacote é enviado para router do laboratório (172.16.1.254), que responde com um pacote que tem como destinatário 172.16.1.29. O router, ao receber esse pacote, reenvia-o para tux23, alterando o destinatário do pacote para o seu endereço, possibilitando então a comunicação entre a rede privada local e a rede pública.

Caso o router não estivesse configurado com o protocolo NAT, a máquina 172.16.1.254, ao receber o pacote, que iria ter como origem o verdadeiro remetente (tux23), não saberia como responder a esse endereço IP e, por isso, é que podemos observar uma ausência de resposta por parte do router do laboratório (172.16.1.254) após desativar o NAT nos passos 6 e 7 da experiência .

Experiência 5 - DNS

O objetivo desta experiência foi a conexão das máquinas da rede IP a um servidor DNS, que efetua a tradução de hostnames para endereços IP, e verificar como isso altera o modo como as máquinas se conectam à Internet.

Quais os comandos necessários para a configuração desta experiência?

Configuração do DNS (slide 47 do guião).

Como configurar o serviço DNS num anfitrião?

O serviço DNS é configurado no ficheiro `resolv.conf`, localizado no diretório `/etc/` do anfitrião (foi necessário fazer esta configuração para os 3 tuxes). A configuração é feita através do comando `nameserver`, que representa o endereço IP do servidor DNS:

- nameserver 172.16.1.1 (endereço IP de services.netlab.fe.up.pt)

Para motivos de teste, uma vez que o servidor acima estava com problemas, foi usado o seguinte servidor DNS:

- nameserver 193.136.28.9

Que pacotes são trocados por DNS, e que informação é transportada?

O anfitrião primeiro envia para o servidor um pacote com o `hostname` e espera que seja retornado o seu endereço IP. O servidor responde com um pacote que contém o endereço IP do hostname em causa.

Experiência 6 - Conexões TCP

Na última experiência, o objetivo foi a observação do funcionamento e comportamento do protocolo TCP, sendo para isso utilizada a aplicação de download desenvolvida anteriormente.

Quais os comandos necessários para a configuração desta experiência?

Compilação e execução da aplicação download.

Quantas conexões TCP são abertas pela aplicação FTP?

São abertas duas conexões TCP pela aplicação, uma quando se entra em contacto com o servidor (control socket), pela qual se envia e recebe comandos para preparar a transferência do ficheiro, e outra para fazer a transferência do ficheiro em si (data socket).

Em qual conexão é transportado o controlo de informação?

O controlo de informação é transportado na primeira conexão TCP, na qual se trata do envio e receção de comandos.

Quais são as fases de uma conexão TCP?

Numa conexão TCP, primeiro estabelece-se a conexão, depois ocorre a troca de dados e por fim a conexão é encerrada, havendo assim três fases.

Como funciona o mecanismo ARQ TCP? Quais são os campos TCP relevantes? Que informação relevante pode ser observada nos logs?

O mecanismo ARQ TCP funciona através do método da janela deslizante, que consiste no controlo de erros na transmissão de dados. Para este efeito, são utilizados *acknowledgement numbers*, que indicam o correto recebimento da trama, *window size*, que indica a gama de pacotes recebidos, e *sequence number*, que é o número do pacote a ser enviado.

Como funciona o mecanismo de congestionamento de controlo TCP? Quais são os campos relevantes? Como é que a taxa de transmissão da conexão de dados evolui ao longo do tempo? Está de acordo com o mecanismo de congestionamento de controlo TCP?

O mecanismo de controlo de congestão do TCP tem como base os ACKs recebidos na transmissão de pacotes. Estes são o source clock da transmissão. É utilizada uma nova variável/valor por conexão, denominada CongestionWindow, de modo a regular o tamanho da janela deslizante de transmissão de pacotes, tendo em conta a congestão da conexão. Este valor é regulado, incrementando se a congestão da rede diminui, ou decrementando se a congestão da rede aumenta. Isto é normalmente feito incrementando CongestionWindow por 1, a cada RTT (“round trip time”). Quando se deteta que um pacote é perdido (normalmente isso é detetado quando ocorre um timeout, ou quando são recebidos 3 ACKs duplicados), o valor de CongestionWindow diminui para metade. O bitrate da conexão irá ser aproximadamente igual a CongestionWindow a dividir por RTT.

No início da conexão, pode também haver uma fase de slow start, que serve para delimitar um threshold que é utilizado numa fase posterior de congestion avoidance.

Foi registado um rápido aumento na taxa de transmissão quando começou o primeiro download, no tux23, tendo depois atingido a um pico alguns segundos depois. Após o início do segundo download, no tux22, a taxa de transmissão no tux23 diminuiu rapidamente e a do tux22 aumentou também rapidamente, e passado alguns “altos e baixos”, a taxa de transmissão estabilizou num nível mais baixo do que era antes do segundo download ter começado.

Estas alterações fazem sentido e estão de acordo com o mecanismo de controlo de congestão do TCP, uma vez que quando circulavam na rede menos pacotes (apenas um download a ser feito), o bitrate para a conexão do tux23 é mais alto do que quando os dois downloads estavam a ser feitos ao mesmo tempo, o que aumenta o congestionamento da rede.

A taxa de transmissão de uma conexão de dados TCP é perturbada pelo aparecimento de uma segunda conexão TCP? Como?

Sim. A taxa de transmissão de pacotes da conexão TCP que já estava iniciada diminui, uma vez que à outra conexão foi atribuída uma capacidade para a transmissão de pacotes na mesma, de modo a que a taxa de transferência seja distribuída de igual forma para cada ligação.

Conclusões

Este segundo trabalho de RCOM teve como objetivo o desenvolvimento de uma aplicação que permitisse o download de um ficheiro através de conexões utilizando os protocolos FTP e TCP, e também a configuração, em vários passos, de uma rede IP, de modo a melhor compreender o funcionamento de várias máquinas e dispositivos, como o switch e o router, e também de modo a reconhecer as diversas técnicas, (NAT, DNS, etc), protocolos (ICMP, ARP, etc) e estruturas de dados (forwarding tables, tabelas ARP, etc) utilizadas na comunicação entre essas máquinas.

A meu ver, todos os objetivos deste trabalho foram cumpridos, sendo que o meu conhecimento sobre os temas abordados no mesmo foi desenvolvido e aprofundado.

Referências

Este trabalho foi desenvolvido com recurso à consulta dos slides teóricos da unidade curricular e ao guia fornecido. Para além disso, deixo abaixo algumas referências a recursos úteis para a configuração do switch e router da Mikrotik, assim como os registos dos RFCs consultados para o desenvolvimento da aplicação.

<https://wiki.mikrotik.com/wiki/Manual:Console>

https://wiki.mikrotik.com/wiki/Manual:First_time_startup#CLI

https://wiki.mikrotik.com/wiki/Manual:Interface/Bridge#Bridge_Interface_Setup

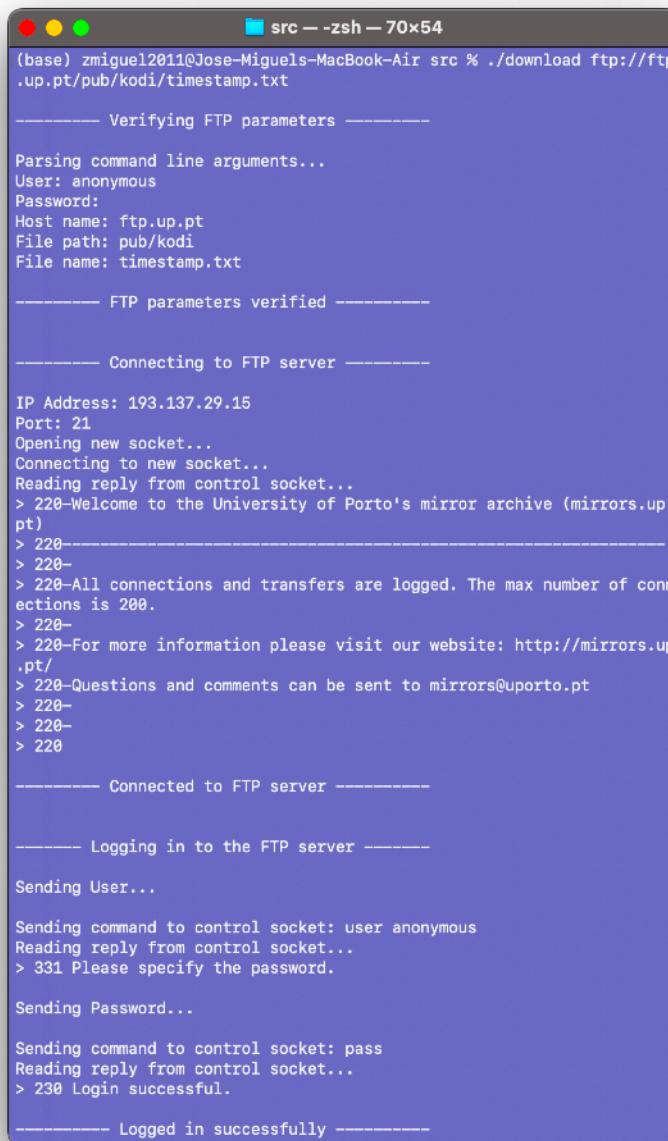
<https://wiki.mikrotik.com/wiki/Manual:IP/Firewall/NAT>

<https://datatracker.ietf.org/doc/html/rfc959>

<https://datatracker.ietf.org/doc/html/rfc1738>

Anexos

Aplicação download



The screenshot shows a terminal window titled "src -- zsh -- 70x54". The command entered is "../download ftp://ftp.up.pt/pub/kodi/timestamp.txt". The application performs the following steps:

- Verifies FTP parameters:
 - Parsing command line arguments...
 - User: anonymous
 - Password:
 - Host name: ftp.up.pt
 - File path: pub/kodi
 - File name: timestamp.txt
- Connects to the FTP server:
 - IP Address: 193.137.29.15
 - Port: 21
 - Opening new socket...
 - Connecting to new socket...
 - Reading reply from control socket...
 - > 220-Welcome to the University of Porto's mirror archive (mirrors.up.pt)
 - > 220-
 - > 220-
 - > 220-All connections and transfers are logged. The max number of connections is 200.
 - > 220-
 - > 220-For more information please visit our website: http://mirrors.up.pt/
 - > 220-Questions and comments can be sent to mirrors@porto.pt
 - > 220-
 - > 220-
 - > 220
- Logs in to the FTP server:
 - Connected to FTP server
 - Logging in to the FTP server
 - Sending User...
 - Sending command to control socket: user anonymous
 - Reading reply from control socket...
 - > 331 Please specify the password.
 - Sending Password...
 - Sending command to control socket: pass
 - Reading reply from control socket...
 - > 230 Login successful.
- Logs in successfully

Figura 1: Execução da aplicação - pt1

```

> 230 Login successful.
----- Logged in successfully -----

----- Changing working directory ----

Sending command to control socket: cwd pub/kodi
Reading reply from control socket...
> 250 Directory successfully changed.

----- Directory changed successfully ----

----- Enabling Passive Mode ----

Sending command to control socket: pasv
Reading reply from control socket...
> 227 Entering Passive Mode (193,137,29,15,225,244).

Connecting to new data socket...

IP Address: 193.137.29.15
Port: 57844
Opening new socket...
Connecting to new socket...

----- Passive Mode was enabled ----

----- Starting file transfer ----

Sending command to control socket: retr timestamp.txt
Reading reply from control socket...
> 150 Opening BINARY mode data connection for timestamp.txt (11 bytes)
.

Started dowloading file timestamp.txt
Finished dowloading file!
Reading reply from control socket...
> 226 Transfer complete.

----- File transfer is complete ----

----- Disconnecting from FTP server ----

Sending command to control socket: quit
Reading reply from control socket...
> 221 Goodbye.

----- Disconnected from FTP server ----

(base) zmiguel2011@Jose-Miguels-MacBook-Air src %

```

Figura 2: Execução da aplicação - pt2

```

// FTP
#define MAX_LENGTH 200 // max length for a given string (reply message)
#define FTP_PORT 21 // the designated port for the FTP

// MISC
#define FALSE 0
#define TRUE 1

```

Figura 3: Macros utilizadas

```

char* reply = (char *) malloc(MAX_LENGTH); // buffer to read the initial reply
/* receive confirmation from server (welcome message) */
readReplyFromControlSocket(&ftp, reply);
if (reply[0] != '2') {
    printf("> Error in connection...\n\n");
    return -1;
}
free(reply);

printf("\n----- Connected to FTP server -----\\n\\n");

printf("\n----- Logging in to the FTP server -----\\n\\n");
// login in the server
if (login(&ftp, params.user, params.password)<0) {
    printf("> Login failed...\\n\\n");
    return -1;
}

printf("\n----- Logged in successfully -----\\n\\n");

/* change working directory in server, if needed */
if (strlen(params.file_path) > 0) {
    printf("\n----- Changing working directory -----\\n\\n");
    if (changeWorkingDirectory(&ftp, params.file_path) < 0)
    {
        printf("> Error while changing working directory\\n");
        return -1;
    }
    printf("\n---- Directory changed successfully ----\\n\\n");
}

printf("\n----- Enabling Passive Mode -----\\n\\n");
/* send pasv command to get ip address and port
   and create the data socket for the file transfer */
if (enablePassiveMode(&ftp) < 0){
    printf("> Error while enabling passive mode\\n");
    return -1;
}

printf("\n----- Passive Mode was enabled -----\\n\\n");

```

Figura 5: Código fonte da função **main** - pt2

```

printf("\n----- Starting file transfer -----\\n\\n");
/* retrieve file through data socket */
if(retrieveFile(&ftp, params.file_name) < 0){
    printf("> Error while retrieving file\\n");
    return -1;
}

printf("\n----- File transfer is complete -----\\n\\n");

printf("\n----- Disconnecting from FTP server -----\\n\\n");
if (disconnectFromSocket(&ftp) < 0){
    printf("> Error while disconnecting from socket\\n");
    return -1;
}

printf("\n----- Disconnected from FTP server -----\\n\\n");

return 0;
}

```

Figura 6: Código fonte da função **main** - pt3

```

int getIPAddress(char *ipAddress, char *hostName) {

    struct hostent *h;
    if ((h = gethostbyname(hostName)) == NULL) {
        perror("gethostbyname");
        return -1;
    }
    strcpy(ipAddress, inet_ntoa(*((struct in_addr *)h->h_addr)));
    return 0;
}

```

Figura 7: Código fonte da função **getIPAddress**

```

int openAndConnectSocket(char* address, int port) {

    int sockfd;
    struct sockaddr_in server_addr;

    printf("IP Address: %s\n", address);
    printf("Port: %d\n", port);

    printf("Opening new socket...\n");

    /* server address handling */
    bzero((char *) &server_addr, sizeof(server_addr));
    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = inet_addr(address);      /*32 bit Internet address network byte ordered*/
    server_addr.sin_port = htons(port);          /*server TCP port must be network byte ordered */

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

    printf("Connecting to new socket...\n");

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

```

Figura 8: Código fonte da função **openAndConnectSocket**

```

int parseArguments(struct FTPparameters* params, char* commandLineArg) {

    printf("Parsing command line arguments...\n");

    /* verifying FTP protocol */
    char *token = strtok(commandLineArg, ":");
    if ((token == NULL) || (strcmp(token, "ftp") != 0)) {
        printf("> Error in the protocol name (should be 'ftp')\n");
        return -1;
    }

    token = strtok(NULL, "\0");
    char* string = (char *) malloc(MAX_LENGTH);
    strcpy(string, token);

    char* aux = (char *) malloc(MAX_LENGTH);
    strcpy(aux, string);
    token = strtok(aux, ":");

    /* verifying user and password */
    if (token == NULL || (strlen(token) < 3) || (token[0] != '/') || (token[1] != '/')) {
        printf("> Error while parsing the user name\n");
        return -1;
    }
    else if (strcmp(token, string) == 0) {
        // if user and password are not set, set them to anonymous
        memset(params->user, 0, sizeof(params->user));
        strcpy(params->user, "anonymous");
        memset(params->password, 0, sizeof(params->password));
        strcpy(params->password, "");

        char* aux2 = (char *) malloc(MAX_LENGTH);
        strcpy(aux2, &string[2]);
        strcpy(string, aux2);
        free(aux2);
    }
}

```

Figura 9: Código fonte da função **parseArguments** - pt1

```
else {
    /* parsing user name */
    memset(params->user, 0, sizeof(params->user));
    strcpy(params->user, &token[2]);
    /* parsing password */
    token = strtok(NULL, "@");
    if (token == NULL || (strlen(token) == 0)) {
        printf("> Error while parsing the password\n");
        return -1;
    }
    memset(params->password, 0, sizeof(params->password));
    strcpy(params->password, token);

    token = strtok(NULL, "\0");
    strcpy(string, token);
}

free(aux);

/* parsing host name */
token = strtok(string, "/");
if (token == NULL) {
    printf("> Error parsing the hostname\n");
    return -1;
}
memset(params->host_name, 0, sizeof(params->host_name));
strcpy(params->host_name, token);

/* parsing file path and file name */
token = strtok(NULL, "\0");
if (token == NULL) {
    printf("> Error while parsing the file path\n");
    return -1;
}
```

Figura 10: Código fonte da função **parseArguments** - pt2

```

char* last = strrchr(token, '/');
if (last != NULL) { // path is set
    memset(params->file_path, 0, sizeof(params->file_path));
    strncpy(params->file_path, token, last - token);
    memset(params->file_name, 0, sizeof(params->file_name));
    strcpy(params->file_name, last + 1);
}
else { // path is not set
    memset(params->file_path, 0, sizeof(params->file_path));
    strcpy(params->file_path, "");
    memset(params->file_name, 0, sizeof(params->file_name));
    strcpy(params->file_name, token);
}

free(string);

return 0;
}

```

Figura 11: Código fonte da função **parseArguments** - pt3

```

int sendCommandToControlSocket(struct FTP *ftp, char *command, char *argument) {

printf("Sending command to control socket: %s %s\n", command, argument);

/* sends command's code */
int bytes = write(ftp->control_socket_fd, command, strlen(command));
if (bytes != strlen(command))
    return -1;
/* sends a space to separate the command's code and argument */
bytes = write(ftp->control_socket_fd, " ", 1);
if (bytes != 1)
    return -1;
/* sends command's argument */
bytes = write(ftp->control_socket_fd, argument, strlen(argument));
if (bytes != strlen(argument))
    return -1;
/* sends end of line code to signal the end of the command */
bytes = write(ftp->control_socket_fd, "\n", 1);
if (bytes != 1)
    return -1;

return 0;
}

```

Figura 12: Código fonte da função **sendCommandToControlSocket**

```

int readReplyFromControlSocket(struct FTP *ftp, char *buffer) {

    printf("Reading reply from control socket... \n");

    FILE *fp = fdopen(ftp->control_socket_fd, "r");
    do {
        /* reset the memory of the buffer in each iteration */
        memset(buffer, 0, MAX_LENGTH); // reads until:
        buffer = fgets(buffer, MAX_LENGTH, fp); // - reaches end of reply
        printf("> %s", buffer); // - reply code is not correct (fail safe)
    } while (buffer[3] != ' ' || !('1' <= buffer[0] && buffer[0] <= '5'));

    return 0;
}

```

Figura 13: Código fonte da função **readReplyFromControlSocket**

```

int sendCommandHandleReply(struct FTP *ftp, char *command, char *argument, char *reply, int dowloadingFile) {

    if (sendCommandToControlSocket(ftp, command, argument) < 0) {
        printf("> Error while sending command %s %s\n", command, argument);
        return -1;
    }

    int code;
    while (1) {
        readReplyFromControlSocket(ftp, reply);
        code = reply[0] - '0'; // first digit of reply code
        switch (code) {
            case 1:
                // expecting another reply
                if (dowloadingFile) return 1; // if downloading a file, reply code should be 150
                else break; // else expect another reply
            case 2:
                // requested action successful
                return 2;
            case 3:
                // needs aditional information
                return 3;
            case 4:
                // try resending the command
                if (sendCommandToControlSocket(ftp, command, argument) < 0) {
                    printf("> Error while sending command %s %s\n", command, argument);
                    return -1;
                }
                break;
            case 5:
                // Command was not accepted, close control socket & exit application
                printf("> Command was not accepted \n");
                close(ftp->control_socket_fd);
                exit(-1);
                break;
            default:
                break;
        }
    }
}

```

Figura 14: Código fonte da função **sendCommandHandleReply**

```

int login(struct FTP *ftp, char *user, char *password) {

    char* reply = (char *) malloc(MAX_LENGTH);

    printf("Sending User...\n\n");
    /* ret is the return value corresponding to the first digit of the reply code */
    int ret = sendCommandHandleReply(ftp, "user", user, reply, FALSE);
    if (ret != 3) {
        printf("> Error while sending user...\n\n");
        return -1;
    }

    printf("\nSending Password...\n\n");
    /* ret is the return value corresponding to the first digit of the reply code */
    ret = sendCommandHandleReply(ftp, "pass", password, reply, FALSE);
    if (ret != 2) {
        printf("> Error while sending password...\n\n");
        return -1;
    }

    free(reply);

    return 0;
}

```

Figura 15: Código fonte da função **login**

```

int changeWorkingDirectory(struct FTP *ftp, char *path) {

    char* reply = (char *) malloc(MAX_LENGTH);

    /* ret is the return value corresponding to the first digit of the reply code */
    int ret = sendCommandHandleReply(ftp, "cwd", path, reply, FALSE);
    if (ret != 2) {
        printf("> Error while sending command cwd\n\n");
        return -1;
    }

    free(reply);

    return 0;
}

```

Figura 16: Código fonte da função **changeWorkingDirectory**

```

int enablePassiveMode(struct FTP *ftp) {

    char* reply = (char *) malloc(MAX_LENGTH);

    int ret = sendCommandHandleReply(ftp, "pasv", "", reply, FALSE);
    int ipPart1, ipPart2, ipPart3, ipPart4;
    int portPart1, portPart2;
    if (ret == 2) {
        // process ip and port informaiton
        if ((sscanf(reply, "227 Entering Passive Mode (%d,%d,%d,%d,%d,%d)",
                    &ipPart1, &ipPart2, &ipPart3, &ipPart4, &portPart1, &portPart2) < 0) {
            printf("> Error while calculating port.\n");
            return -1;
        }
    }
    else {
        printf("> Error while sending command pasv \n\n");
        return -1;
    }

    char* ipAddress = (char *) malloc(MAX_LENGTH);
    sprintf(ipAddress, "%d.%d.%d.%d", ipPart1, ipPart2, ipPart3, ipPart4);
    int port = portPart1 * 256 + portPart2;

    printf("\nConnecting to new data socket...\n\n");

    if ((ftp->data_socket_fd = openAndConnectSocket(ipAddress, port)) < 0) {
        printf("> Error while creating new data socket\n");
        return -1;
    }

    free(reply);
    free(ipAddress);

    return 0;
}

```

Figura 17: Código fonte da função **enablePassiveMode**

```

int retrieveFile(struct FTP *ftp, char *fileName) {

    char* reply = (char *) malloc(MAX_LENGTH);

    /* ret is the return value corresponding to the first digit of the reply code */
    int ret = sendCommandHandleReply(ftp, "retr", fileName, reply, TRUE);
    if (ret != 1) { // reply code should be 150
        printf("> Error while sending command retr\n\n");
        return -1;
    }

    FILE* fp;
    fp = fopen(fileName, "w");
    if (fp == NULL) {
        perror(fileName);
        return -1;
    }

    char* buffer = (char *) malloc(1024);
    int bytes;
    printf("\nStarted dowloading file %s\n", fileName);
    while((bytes = read(ftp->data_socket_fd, buffer, sizeof(buffer)))){

        if(bytes < 0){
            printf("> Error while reading from data socket\n");
            return -1;
        }
        if((bytes = fwrite(buffer, bytes, 1, fp)) < 0){
            printf("> Error while writing data to file\n");
            return -1;
        }
    }

    free(buffer);
}

```

Figura 18: Código fonte da função **retrieveFile** - pt1

```

printf("Finished dowloading file!\n");
if(fclose(fp) < 0){
    printf("> Error while closing file\n");
    return -1;
}

close(ftp->data_socket_fd);

readReplyFromControlSocket(ftp, reply);
if (reply[0] != '2')
    return -1;

free(reply);

return 0;
}

```

Figura 19: Código fonte da função **retrieveFile** - pt2

```

int disconnectFromSocket(struct FTP *ftp) {

    char* reply = (char *) malloc(MAX_LENGTH);

    /* ret is the return value corresponding to the first digit of the reply code */
    int ret = sendCommandHandleReply(ftp, "quit", "", reply, FALSE);
    if (ret != 2) {
        printf("> Error while sending command quit\n\n");
        return -1;
    }

    free(reply);

    return 0;
}

```

Figura 20: Código fonte da função **disconnectFromSocket**

```

/**
 * Function that, given a host name, retrieves its corresponding IP address
 *
 * @param idAddress Variable that is going to point to the IP Address
 * @param hostName The host's name
 * @return 0 if success; -1 otherwise
 */
int getIPAddress(char *ipAddress, char *hostName);

```

Figura 21: Código do cabeçalho da função **getIPAddress**

```

/**
 * Function that opens a new TCP socket, and connects it to the address and port specified
 *
 * @param address The IP address of the server
 * @param port The number of the port to be used
 * @return Socket descriptor if success; -1 otherwise
 */
int openAndConnectSocket(char *address, int port);

```

Figura 22: Código do cabeçalho da função **openAndConnectSocket**

```

/**
 * Function that parses the command line arguments, retrieving a struct with all the individual fields
 *
 * @param params Pointer to the structure that is going to have the individual fields
 * @param commandLineArg Argument from the command line, that is going to be parsed
 * @return 0 if sucess; -1 otherwise
 */
int parseArguments(struct FTPparameters *params, char *commandLineArg);

```

```
/***
 * Function that sends a command through the control socket
 *
 * @param ftp Struct containing the socket descriptors
 * @param command Command to be sent
 * @param argument Argument of the command to be sent
 * @return 0 if success; -1 otherwise
 */
int sendCommandToControlSocket(struct FTP *ftp, char *command, char *argument);
```

Figura 24: Código do cabeçalho da função **sendCommandToControlSocket**

```
/***
 * Function that reads a reply from a control socket, according to FTP
 *
 * @param ftp Struct containing the socket descriptors
 * @param buffer Buffer containing the reply code and message received from the server
 * @return 0 if success; -1 otherwise
 */
int readReplyFromControlSocket(struct FTP *ftp, char *buffer);
```

Figura 25: Código do cabeçalho da função **readReplyFromControlSocket**

```
/***
 * Function that sends a command to the control socket and handles the reply received
 *
 * @param ftp Struct containing the socket descriptors
 * @param command Command to be sent
 * @param argument Argument of the command to be sent
 * @param reply Buffer that is going to store the reply code and message received from the server
 * @param dowloadingFile Indicates if the file is about to be transferred from the data socket
 * @return Positive if success (the first digit of the reply code); -1 otherwise
 */
int sendCommandHandleReply(struct FTP *ftp, char *command, char *argument, char *reply, int dowloadingFile);
```

Figura 26: Código do cabeçalho da função **sendCommandHandleReply**

```
/***
 * Function that sends the login information to the socket for authentication
 *
 * @param ftp Struct containing the socket descriptors
 * @param user User to be sent to the socket
 * @param password Password to be sent to the socket
 * @return 0 if success; -1 otherwise
 */
int login(struct FTP *ftp, char *user, char *password);
```

Figura 27: Código do cabeçalho da função **login**

```
/***
 * Function to change the working directory of the FTP server, using the CWD command
 *
 * @param ftp Struct containing the socket descriptors
 * @param path Path of the new working directory to be changed to
 * @return 0 if successful; -1 otherwise
 */
int changeWorkingDirectory(struct FTP *ftp, char *path);
```

Figura 28: Código do cabeçalho da função **changeWorkingDirectory**

```
/***
 * Function that instructs the FTP server to enter a Passive session rather than Active
 * and then obtains a server IP address and Port for the data socket used in the transfer of a file
 *
 * @param ftp Struct containing the socket descriptors
 * @return 0 if success; -1 otherwise
 */
int enablePassiveMode(struct FTP *ftp);
```

Figura 29: Código do cabeçalho da função **enablePassiveMode**

```
/***
 * Function that sends the RETR command to the control socket,
 * so the file can be transferred in the data socket
 * Then proceeds to download the file sent from the server
 * through the data socket, and saves it in a local file
 *
 * @param ftp Struct containing the socket descriptors
 * @param fileName The name of the file to be transferred
 * @return 0 if successful; -1 otherwise
 */
int retrieveFile(struct FTP *ftp, char *fileName);
```

Figura 30: Código do cabeçalho da função **retrieveFile**

```
/***
 * Function to disconnect from the socket on the FTP server, using the QUIT command
 *
 * @param ftp Struct containing the socket descriptors
 * @return 0 if successful; -1 otherwise
 */
int disconnectFromSocket(struct FTP *ftp);
```

Figura 31: Código do cabeçalho da função **disconnectFromSocket**

Logs das experiências

Experiência 1

No.	Time	Source	Destination	Protocol	Length	Info
10	18.020103762	Routerbo_1c:9f:51	Spanning-tree-(for... STP	60	RST. Root = 32768/0:c4:ad:34:1c:9f:51 Cost =	
11	20.022415993	Routerbo_1c:9f:51	Spanning-tree-(for... STP	60	RST. Root = 32768/0:c4:ad:34:1c:9f:51 Cost =	
12	21.485410517	0.0.0.0	255.255.255.255	MNDP	182	5678 - 5678 Len=140
13	21.485450956	Routerbo_1c:9f:51	CDP/VTP/DTP/PAGP/UD...	CDP	116	Device ID: MikroTik Port ID: bridgeY0/ether4
14	21.485543797	Routerbo_1c:9f:51	LLDP_Multicast	LLDP	133	TTL = 120 System Name = MikroTik System Descri
15	22.024706154	Routerbo_1c:9f:51	Spanning-tree-(for... STP	60	RST. Root = 32768/0:c4:ad:34:1c:9f:51 Cost =	
16	24.027002810	Routerbo_1c:9f:51	Spanning-tree-(for... STP	60	RST. Root = 32768/0:c4:ad:34:1c:9f:51 Cost =	
17	26.029017931	Routerbo_1c:9f:51	Spanning-tree-(for... STP	60	RST. Root = 32768/0:c4:ad:34:1c:9f:51 Cost =	
18	28.031312492	Routerbo_1c:9f:51	Spanning-tree-(for... STP	60	RST. Root = 32768/0:c4:ad:34:1c:9f:51 Cost =	
19	28.113665485	172.16.20.1	172.16.20.254	ICMP	98	Echo (ping) request id=0x28d8, seq=1/256, ttl
20	28.113872638	172.16.20.254	172.16.20.1	ICMP	98	Echo (ping) reply id=0x28d8, seq=1/256, ttl
21	29.138778275	172.16.20.1	172.16.20.254	ICMP	98	Echo (ping) request id=0x28d8, seq=2/512, ttl
22	29.138950786	172.16.20.254	172.16.20.1	ICMP	98	Echo (ping) reply id=0x28d8, seq=2/512, ttl
23	30.033626399	Routerbo_1c:9f:51	Spanning-tree-(for... STP	60	RST. Root = 32768/0:c4:ad:34:1c:9f:51 Cost =	
24	30.162793486	172.16.20.1	172.16.20.254	ICMP	98	Echo (ping) request id=0x28d8, seq=3/768, ttl
25	30.162991630	172.16.20.254	172.16.20.1	ICMP	98	Echo (ping) reply id=0x28d8, seq=3/768, ttl
26	31.186788374	172.16.20.1	172.16.20.254	ICMP	98	Echo (ping) request id=0x28d8, seq=4/1024, tt
27	31.186968568	172.16.20.254	172.16.20.1	ICMP	98	Echo (ping) reply id=0x28d8, seq=4/1024, tt
28	32.035917189	Routerbo_1c:9f:51	Spanning-tree-(for... STP	60	RST. Root = 32768/0:c4:ad:34:1c:9f:51 Cost =	
29	32.210799675	172.16.20.1	172.16.20.254	ICMP	98	Echo (ping) request id=0x28d8, seq=5/1280, tt
30	32.210972396	172.16.20.254	172.16.20.1	ICMP	98	Echo (ping) reply id=0x28d8, seq=5/1280, tt
31	33.163977628	HewlettP_a6:a4:f1	HewlettP_5a:7d:12	ARP	60	Who has 172.16.20.1? Tell 172.16.20.254
32	33.163997463	HewlettP_5a:7d:12	HewlettP_a6:a4:f1	ARP	42	172.16.20.1 is at 00:21:5a:5a:7d:12
33	33.170754255	HewlettP_5a:7d:12	HewlettP_a6:a4:f1	ARP	42	Who has 172.16.20.254? Tell 172.16.20.1
34	33.170882836	HewlettP_a6:a4:f1	HewlettP_5a:7d:12	ARP	60	172.16.20.254 is at 00:22:64:a6:a4:f1
35	34.037900950	Routerbo_1c:9f:51	Spanning-tree-(for... STP	60	RST. Root = 32768/0:c4:ad:34:1c:9f:51 Cost =	
36	36.040190342	Routerbo_1c:9f:51	Spanning-tree-(for... STP	60	RST. Root = 32768/0:c4:ad:34:1c:9f:51 Cost =	
37	38.042516752	Routerbo_1c:9f:51	Spanning-tree-(for... STP	60	RST. Root = 32768/0:c4:ad:34:1c:9f:51 Cost =	
38	40.044796785	Routerbo_1c:9f:51	Spanning-tree-(for... STP	60	RST. Root = 32768/0:c4:ad:34:1c:9f:51 Cost =	
39	42.047094350	Routerbo_1c:9f:51	Spanning-tree-(for... STP	60	RST. Root = 32768/0:c4:ad:34:1c:9f:51 Cost =	
40	44.044110308	Routerbo_1c:9f:51	Spanning-tree-(for... STP	60	RST. Root = 32768/0:c4:ad:34:1c:9f:51 Cost =	

Frame 1: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0
 IEEE 802.3 Ethernet
 Logical-Link Control

0000	01	80	c2	00	00	00	c4	ad	34	1c	9f	51	00	27	42	42	4..Q.'BB	
0010	00	00	00	02	02	3c	80	00	c4	ad	34	1c	9f	51	00	00	14	00<.. 4..Q..
0020	00	00	00	00	c4	ad	34	1c	9f	51	00	01	00	00	14	004..Q.....		

exp2_ping_tux23.pcapng

Packets: 49 · Displayed: 49 (100.0%) · Profile: Default

Figura 32: Mensagens entre o tux23 e o tux24 - ARP e ICMP

Experiência 2

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	Routerbo_1c:9f:4e	Spanning-tree-(for... STP	60	RST. Root = 32768/0:c4:ad:34:1c	
2	1.812319150	HewlettP_5a:7d:12	Broadcast	ARP	42	Who has 172.16.20.254? Tell 172
3	1.812443119	HewlettP_a6:a4:f1	HewlettP_5a:7d:12	ARP	60	172.16.20.254 is at 00:22:64:a6
4	1.812460858	172.16.20.1	172.16.20.254	ICMP	98	Echo (ping) request id=0x467c,
5	1.812575607	172.16.20.254	172.16.20.1	ICMP	98	Echo (ping) reply id=0x467c,
6	2.002303296	Routerbo_1c:9f:4e	Spanning-tree-(for... STP	60	RST. Root = 32768/0:c4:ad:34:1c	
7	2.834432507	172.16.20.1	172.16.20.254	ICMP	98	Echo (ping) request id=0x467c,
8	2.834578196	172.16.20.254	172.16.20.1	ICMP	98	Echo (ping) reply id=0x467c,
9	3.858431229	172.16.20.1	172.16.20.254	ICMP	98	Echo (ping) request id=0x467c,
10	3.858550029	172.16.20.254	172.16.20.1	ICMP	98	Echo (ping) reply id=0x467c,
11	4.004572579	Routerbo_1c:9f:4e	Spanning-tree-(for... STP	60	RST. Root = 32768/0:c4:ad:34:1c	
12	4.882434282	172.16.20.1	172.16.20.254	ICMP	98	Echo (ping) request id=0x467c,
13	4.882553082	172.16.20.254	172.16.20.1	ICMP	98	Echo (ping) reply id=0x467c,
14	5.906435309	172.16.20.1	172.16.20.254	ICMP	98	Echo (ping) request id=0x467c,
15	5.906550058	172.16.20.254	172.16.20.1	ICMP	98	Echo (ping) reply id=0x467c,
16	6.006846122	Routerbo_1c:9f:4e	Spanning-tree-(for... STP	60	RST. Root = 32768/0:c4:ad:34:1c	
17	6.910858706	HewlettP_a6:a4:f1	HewlettP_5a:7d:12	ARP	60	Who has 172.16.20.1? Tell 172.1
18	6.910879589	HewlettP_5a:7d:12	HewlettP_a6:a4:f1	ARP	42	172.16.20.1 is at 00:21:5a:5a:7
19	6.930424463	172.16.20.1	172.16.20.254	ICMP	98	Echo (ping) request id=0x467c,
20	6.930514628	172.16.20.254	172.16.20.1	ICMP	98	Echo (ping) reply id=0x467c,
21	7.954438201	172.16.20.1	172.16.20.254	ICMP	98	Echo (ping) request id=0x467c,
22	7.954580747	172.16.20.254	172.16.20.1	ICMP	98	Echo (ping) reply id=0x467c,
23	8.008725062	Routerbo_1c:9f:4e	Spanning-tree-(for... STP	60	RST. Root = 32768/0:c4:ad:34:1c	
24	8.978430009	172.16.20.1	172.16.20.254	ICMP	98	Echo (ping) request id=0x467c,
25	8.978547551	172.16.20.254	172.16.20.1	ICMP	98	Echo (ping) reply id=0x467c,
26	10.002433899	172.16.20.1	172.16.20.254	ICMP	98	Echo (ping) request id=0x467c,
27	10.002550270	172.16.20.254	172.16.20.1	ICMP	98	Echo (ping) reply id=0x467c,
28	10.011000072	Routerbo_1c:9f:4e	Spanning-tree-(for... STP	60	RST. Root = 32768/0:c4:ad:34:1c	
29	11.026410105	172.16.20.1	172.16.20.254	TCMP	98	Echo (ping) request id=0x467c,

Frame 1: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0
 IEEE 802.3 Ethernet
 Logical-Link Control

0000	01	80	c2	00	00	00	c4	ad	34	1c	9f	4e	00	27	42	42	4..N.'BB
------	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	-------	----------

exp1_ping.pcapng

Packets: 35 · Displayed: 35 (100.0%) · Profile: Default

Figura 33: Log 1 da Exp. 2 - Ping de tux23 para tux24

Relatório do 2º Trabalho - RCOM 2023/2024

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	0.0.0.0	255.255.255.255	MNDP	183	5678 → 5678 Len=141
2	0.000027798	Routerbo_1c:9f:5b	CDP/FTP/DTP/PAgP/UD...	CDP	117	Device ID: MikroTik Port ID: bridge20/ether14
3	0.000126276	Routerbo_1c:9f:5b	LLDP_Multicast	LLDP	134	TTL = 120 System Name = MikroTik System Description
4	0.819535578	Routerbo_1c:9f:5b	Spanning-tree-(for...	STP	60	RST. Root = 32768/0/c4:ad:34:ic:9f:5b Cost = 0 Port = 0x0001
5	1.852798999	172.16.20.1	172.16.20.255	ICMP	98	Echo (ping) request id=0x22bc, seq=1/256, ttl=64 (no response found)
6	1.852966132	172.16.20.254	172.16.20.1	ICMP	98	Echo (ping) reply id=0x22bc, seq=1/256, ttl=64
7	2.821876662	Routerbo_1c:9f:5b	Spanning-tree-(for...	STP	60	RST. Root = 32768/0/c4:ad:34:ic:9f:5b Cost = 0 Port = 0x0001
8	2.877572486	172.16.20.1	172.16.20.255	ICMP	98	Echo (ping) request id=0x22bc, seq=2/512, ttl=64 (no response found)
9	2.877700647	172.16.20.254	172.16.20.1	ICMP	98	Echo (ping) reply id=0x22bc, seq=2/512, ttl=64
10	3.901559814	172.16.20.1	172.16.20.255	ICMP	98	Echo (ping) request id=0x22bc, seq=3/768, ttl=64 (no response found)
11	3.901674994	172.16.20.254	172.16.20.1	ICMP	98	Echo (ping) reply id=0x22bc, seq=3/768, ttl=64
12	4.824221797	Routerbo_1c:9f:5b	Spanning-tree-(for...	STP	60	RST. Root = 32768/0/c4:ad:34:ic:9f:5b Cost = 0 Port = 0x0001
13	4.925543669	172.16.20.1	172.16.20.255	ICMP	98	Echo (ping) request id=0x22bc, seq=4/1024, ttl=64 (no response found)
14	4.925669246	172.16.20.254	172.16.20.1	ICMP	98	Echo (ping) reply id=0x22bc, seq=4/1024, ttl=64
15	5.949566976	172.16.20.1	172.16.20.255	ICMP	98	Echo (ping) request id=0x22bc, seq=5/1280, ttl=64 (no response found)
16	5.949727614	172.16.20.254	172.16.20.1	ICMP	98	Echo (ping) reply id=0x22bc, seq=5/1280, ttl=64
17	6.826072726	Routerbo_1c:9f:5b	Spanning-tree-(for...	STP	60	RST. Root = 32768/0/c4:ad:34:ic:9f:5b Cost = 0 Port = 0x0001
18	6.973566047	172.16.20.1	172.16.20.255	ICMP	98	Echo (ping) request id=0x22bc, seq=6/1536, ttl=64 (no response found)
19	6.973693929	172.16.20.254	172.16.20.1	ICMP	98	Echo (ping) reply id=0x22bc, seq=6/1536, ttl=64
20	7.020702373	HewlettP_a6:a4:f1	HewlettP_5a:7d:12	ARP	60	Who has 172.16.20.1? Tell 172.16.20.254
21	7.020710335	HewlettP_5a:7d:12	HewlettP_a6:a4:f1	ARP	42	172.16.20.1 is at 00:21:5a:5a:7d:12
22	7.997571544	172.16.20.1	172.16.20.255	ICMP	98	Echo (ping) request id=0x22bc, seq=7/1792, ttl=64 (no response found)
23	7.9977084315	172.16.20.254	172.16.20.1	ICMP	98	Echo (ping) reply id=0x22bc, seq=7/1792, ttl=64
24	8.028425055	Routerbo_1c:9f:5b	Spanning-tree-(for...	STP	60	RST. Root = 32768/0/c4:ad:34:ic:9f:5b Cost = 0 Port = 0x0001
25	9.021573269	172.16.20.1	172.16.20.255	ICMP	98	Echo (ping) request id=0x22bc, seq=8/2048, ttl=64 (no response found)
26	9.021701989	172.16.20.254	172.16.20.1	ICMP	98	Echo (ping) reply id=0x22bc, seq=8/2048, ttl=64
27	10.045578626	172.16.20.1	172.16.20.255	ICMP	98	Echo (ping) request id=0x22bc, seq=9/2304, ttl=64 (no response found)
28	10.045710976	172.16.20.254	172.16.20.1	ICMP	98	Echo (ping) reply id=0x22bc, seq=9/2304, ttl=64
29	10.830978312	Routerbo_1c:9f:5b	Spanning-tree-(for...	STP	60	RST. Root = 32768/0/c4:ad:34:ic:9f:5b Cost = 0 Port = 0x0001
30	11.069571063	172.16.20.1	172.16.20.255	ICMP	98	Echo (ping) request id=0x22bc, seq=10/2560, ttl=64 (no response found)
31	11.069727021	172.16.20.254	172.16.20.1	ICMP	98	Echo (ping) reply id=0x22bc, seq=10/2560, ttl=64

Frame 1: 183 bytes on wire (1464 bits), 183 bytes captured (1464 bits) on interface 0
Ethernet II, Src: Routerbo_1c:9f:5b (c4:ad:34:ic:9f:5b), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
Internet Protocol Version 4, Src: 0.0.0.0, Dst: 255.255.255.255

```
0000 ff ff ff ff ff c4 ad 34 1c 9f 5b 08 00 45 00 ..... 4 .. [E]
0010 00 a9 00 00 00 00 40 11 7a 45 00 00 00 ff ff ..... @ zE ....
0020 ff ff 16 2e 16 2e 00 95 a4 23 07 00 00 00 00 01 ..... # ....
```



Figura 34: Log 2 da Exp. 2 - Ping Broadcast a partir de tux23

No.	Time	Source	Destination	Protocol	Len	Info
36	48...	172.16.21...	172.16.21.255	ICMP	...	Echo (ping) request id=0x7fff8, seq=2/512, ttl=64 (no response found)
31	49...	172.16.21...	172.16.21.255	ICMP	...	Echo (ping) request id=0x7fff8, seq=3/768, ttl=64 (no response found)
32	50...	Routerbo_...	Spanning-tree-(f...	STP	...	RST. Root = 32768/0/c4:ad:34:ic:9f:5a Cost = 0 Port = 0x0001
33	50...	172.16.21...	172.16.21.255	ICMP	...	Echo (ping) request id=0x7fff8, seq=4/1024, ttl=64 (no response found)
34	51...	172.16.21...	172.16.21.255	ICMP	...	Echo (ping) request id=0x7fff8, seq=5/1280, ttl=64 (no response found)
35	52...	Routerbo_...	Spanning-tree-(f...	STP	...	RST. Root = 32768/0/c4:ad:34:ic:9f:5a Cost = 0 Port = 0x0001
36	52...	172.16.21...	172.16.21.255	ICMP	...	Echo (ping) request id=0x7fff8, seq=6/1536, ttl=64 (no response found)
37	53...	172.16.21...	172.16.21.255	ICMP	...	Echo (ping) request id=0x7fff8, seq=7/1792, ttl=64 (no response found)
38	54...	Routerbo_...	Spanning-tree-(f...	STP	...	RST. Root = 32768/0/c4:ad:34:ic:9f:5a Cost = 0 Port = 0x0001
39	54...	172.16.21...	172.16.21.255	ICMP	...	Echo (ping) request id=0x7fff8, seq=8/2048, ttl=64 (no response found)
40	55...	172.16.21...	172.16.21.255	ICMP	...	Echo (ping) request id=0x7fff8, seq=9/2304, ttl=64 (no response found)
41	56...	Routerbo_...	Spanning-tree-(f...	STP	...	RST. Root = 32768/0/c4:ad:34:ic:9f:5a Cost = 0 Port = 0x0001
42	56...	172.16.21...	172.16.21.255	ICMP	...	Echo (ping) request id=0x7fff8, seq=10/2560, ttl=64 (no response found)
43	57...	172.16.21...	172.16.21.255	ICMP	...	Echo (ping) request id=0x7fff8, seq=11/2816, ttl=64 (no response found)
44	58...	Routerbo_...	Spanning-tree-(f...	STP	...	RST. Root = 32768/0/c4:ad:34:ic:9f:5a Cost = 0 Port = 0x0001
45	59...	172.16.21...	172.16.21.255	ICMP	...	Echo (ping) request id=0x7fff8, seq=12/3072, ttl=64 (no response found)
46	60...	172.16.21...	172.16.21.255	ICMP	...	Echo (ping) request id=0x7fff8, seq=13/3228, ttl=64 (no response found)
47	61...	Routerbo_...	Spanning-tree-(f...	STP	...	RST. Root = 32768/0/c4:ad:34:ic:9f:5a Cost = 0 Port = 0x0001
48	61...	172.16.21...	172.16.21.255	ICMP	...	Echo (ping) request id=0x7fff8, seq=14/3584, ttl=64 (no response found)
49	62...	Routerbo_...	Spanning-tree-(f...	STP	...	RST. Root = 32768/0/c4:ad:34:ic:9f:5a Cost = 0 Port = 0x0001
50	62...	172.16.21...	172.16.21.255	ICMP	...	Echo (ping) request id=0x7fff8, seq=15/3840, ttl=64 (no response found)
51	63...	172.16.21...	172.16.21.255	ICMP	...	Echo (ping) request id=0x7fff8, seq=16/4096, ttl=64 (no response found)
52	64...	Routerbo_...	Spanning-tree-(f...	STP	...	RST. Root = 32768/0/c4:ad:34:ic:9f:5a Cost = 0 Port = 0x0001
53	64...	172.16.21...	172.16.21.255	ICMP	...	Echo (ping) request id=0x7fff8, seq=17/4352, ttl=64 (no response found)
54	65...	172.16.21...	172.16.21.255	ICMP	...	Echo (ping) request id=0x7fff8, seq=18/4608, ttl=64 (no response found)
55	65...	0.0.0.0	255.255.255.255	MNDP	5678	→ 5678 Len=141
56	65...	Routerbo_...	CDP/FTP/DTP/PAgP...	CDP	...	Device ID: MikroTik Port ID: bridge21/ether13
57	65...	Routerbo_...	LLDP_Multicast	LLDP	...	TTL = 120 System Name = MikroTik System Description = MikroTik Route...
58	66...	Routerbo_...	Spanning-tree-(f...	STP	...	RST. Root = 32768/0/c4:ad:34:ic:9f:5a Cost = 0 Port = 0x0001
59	66...	172.16.21...	172.16.21.255	ICMP	...	Echo (ping) request id=0x7fff8, seq=19/4864, ttl=64 (no response found)
60	67...	172.16.21...	172.16.21.255	ICMP	...	Echo (ping) request id=0x7fff8, seq=20/5120, ttl=64 (no response found)
61	68...	Routerbo_...	Spanning-tree-(f...	STP	...	RST. Root = 32768/0/c4:ad:34:ic:9f:5a Cost = 0 Port = 0x0001
62	68...	172.16.21...	172.16.21.255	ICMP	...	Echo (ping) request id=0x7fff8, seq=21/5376, ttl=64 (no response found)
63	69...	172.16.21...	172.16.21.255	ICMP	...	Echo (ping) request id=0x7fff8, seq=22/5632, ttl=64 (no response found)
64	70...	Routerbo_...	Spanning-tree-(f...	STP	...	RST. Root = 32768/0/c4:ad:34:ic:9f:5a Cost = 0 Port = 0x0001
65	72...	Routerbo_...	Spanning-tree-(f...	STP	...	RST. Root = 32768/0/c4:ad:34:ic:9f:5a Cost = 0 Port = 0x0001

0000 01 80 c2 00 00 00 c4 ad 34 1c 9f 5a 00 27 42 42 4 .. Z 'BB'

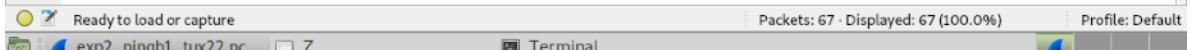


Figura 35: Log 3 da Exp. 2 - Ping Broadcast a partir de tux22

Experiência 3

12	15.076837655	172.16.20.1	172.16.20.254	ICMP	98 Echo (ping) request	id=0x2b9b, seq=1/256, ttl=64
13	15.077000179	172.16.20.254	172.16.20.1	ICMP	98 Echo (ping) reply	id=0x2b9b, seq=1/256, ttl=64
14	16.017786467	Routerbo_ic:9f:5b	Spanning-tree-(for.. STP)		60 RST. Root = 32768/0/c4:ad:34:1c:9f:5b Cost = 0 Pd	
15	16.091222607	172.16.20.1	172.16.20.254	ICMP	98 Echo (ping) request	id=0x2b9b, seq=2/512, ttl=64
16	16.091353911	172.16.20.254	172.16.20.1	ICMP	98 Echo (ping) reply	id=0x2b9b, seq=2/512, ttl=64
17	17.115213579	172.16.20.1	172.16.20.254	ICMP	98 Echo (ping) request	id=0x2b9b, seq=3/768, ttl=64
18	17.115342089	172.16.20.254	172.16.20.1	ICMP	98 Echo (ping) reply	id=0x2b9b, seq=3/768, ttl=64
19	18.019895329	Routerbo_ic:9f:5b	Spanning-tree-(for.. STP)		60 RST. Root = 32768/0/c4:ad:34:1c:9f:5b Cost = 0 Pd	
20	18.139234444	172.16.20.1	172.16.20.254	ICMP	98 Echo (ping) request	id=0x2b9b, seq=4/1024, ttl=64
21	18.139363303	172.16.20.254	172.16.20.1	ICMP	98 Echo (ping) reply	id=0x2b9b, seq=4/1024, ttl=64
22	19.163216126	172.16.20.1	172.16.20.254	ICMP	98 Echo (ping) request	id=0x2b9b, seq=5/1280, ttl=64
23	19.163346313	172.16.20.254	172.16.20.1	ICMP	98 Echo (ping) reply	id=0x2b9b, seq=5/1280, ttl=64
24	20.022189775	Routerbo_ic:9f:5b	Spanning-tree-(for.. STP)		60 RST. Root = 32768/0/c4:ad:34:1c:9f:5b Cost = 0 Pd	
25	20.187221346	172.16.20.1	172.16.20.254	ICMP	98 Echo (ping) request	id=0x2b9b, seq=6/1536, ttl=64
26	20.187370321	172.16.20.254	172.16.20.1	ICMP	98 Echo (ping) reply	id=0x2b9b, seq=6/1536, ttl=64
27	20.232323435	HewlettP_a6:a4:f1	HewlettP_5a:7d:12	ARP	60 Who has 172.16.20.1? Tell 172.16.20.254	
28	20.232333771	HewlettP_5a:7d:12	HewlettP_a6:a4:f1	ARP	42 172.16.20.1 is at 00:21:5a:5a:7d:12	
29	20.283190546	HewlettP_5a:7d:12	HewlettP_a6:a4:f1	ARP	42 Who has 172.16.20.254? Tell 172.16.20.1	
30	20.283279107	HewlettP_a6:a4:f1	HewlettP_5a:7d:12	ARP	60 172.16.20.254 is at 00:22:64:a6:a4:f1	
31	21.211229360	172.16.20.1	172.16.20.254	ICMP	98 Echo (ping) request	id=0x2b9b, seq=7/1792, ttl=64
32	21.211367090	172.16.20.254	172.16.20.1	ICMP	98 Echo (ping) reply	id=0x2b9b, seq=7/1792, ttl=64
33	22.014476582	Routerbo_ic:9f:5b	Spanning-tree-(for.. STP)		60 RST. Root = 32768/0/c4:ad:34:1c:9f:5b Cost = 0 Pd	
34	22.235244248	172.16.20.1	172.16.20.254	ICMP	98 Echo (ping) request	id=0x2b9b, seq=8/2048, ttl=64

Figura 36: Log 1 da Exp. 3 - Ping de tux23 para tux24.eth0

60	64.669827067	172.16.20.1	172.16.21.253	ICMP	98 Echo (ping) request	id=0xbcf, seq=1/256, ttl=64
61	64.669989590	172.16.21.253	172.16.20.1	ICMP	98 Echo (ping) reply	id=0xbcf, seq=1/256, ttl=64
62	65.691234209	172.16.20.1	172.16.21.253	ICMP	98 Echo (ping) request	id=0xbcf, seq=2/512, ttl=64
63	65.691398549	172.16.21.253	172.16.20.1	ICMP	98 Echo (ping) reply	id=0xbcf, seq=2/512, ttl=64
64	66.063751239	Routerbo_ic:9f:5b	Spanning-tree-(for.. STP)		60 RST. Root = 32768/0/c4:ad:34:1c:9f:5b Cost = 0 Pd	
65	66.715218127	172.16.20.1	172.16.21.253	ICMP	98 Echo (ping) request	id=0xbcf, seq=3/768, ttl=64
66	66.715346847	172.16.21.253	172.16.20.1	ICMP	98 Echo (ping) reply	id=0xbcf, seq=3/768, ttl=64
67	67.739237246	172.16.20.1	172.16.21.253	ICMP	98 Echo (ping) request	id=0xbcf, seq=4/1024, ttl=64
68	67.739371763	172.16.21.253	172.16.20.1	ICMP	98 Echo (ping) reply	id=0xbcf, seq=4/1024, ttl=64
69	68.066051889	Routerbo_ic:9f:5b	Spanning-tree-(for.. STP)		60 RST. Root = 32768/0/c4:ad:34:1c:9f:5b Cost = 0 Pd	
70	68.763237567	172.16.20.1	172.16.21.253	ICMP	98 Echo (ping) request	id=0xbcf, seq=5/1280, ttl=64
71	68.763383897	172.16.21.253	172.16.20.1	ICMP	98 Echo (ping) reply	id=0xbcf, seq=5/1280, ttl=64
72	69.691197741	HewlettP_5a:7d:12	HewlettP_a6:a4:f1	ARP	42 Who has 172.16.20.254? Tell 172.16.20.1	
73	69.691314734	HewlettP_a6:a4:f1	HewlettP_5a:7d:12	ARP	60 172.16.20.254 is at 00:22:64:a6:a4:f1	
74	69.787226872	172.16.20.1	172.16.21.253	ICMP	98 Echo (ping) request	id=0xbcf, seq=6/1536, ttl=64
75	69.787358805	172.16.21.253	172.16.20.1	ICMP	98 Echo (ping) reply	id=0xbcf, seq=6/1536, ttl=64
76	69.896288639	HewlettP_a6:a4:f1	HewlettP_5a:7d:12	ARP	60 Who has 172.16.20.1? Tell 172.16.20.254	
77	69.896306448	HewlettP_5a:7d:12	HewlettP_a6:a4:f1	ARP	42 172.16.20.1 is at 00:21:5a:5a:7d:12	
78	70.068946210	Routerbo_ic:9f:5b	Spanning-tree-(for.. STP)		60 RST. Root = 32768/0/c4:ad:34:1c:9f:5b Cost = 0 Pd	
79	70.811230626	172.16.20.1	172.16.21.253	ICMP	98 Echo (ping) request	id=0xbcf, seq=7/1792, ttl=64
80	70.811395035	172.16.21.253	172.16.20.1	ICMP	98 Echo (ping) reply	id=0xbcf, seq=7/1792, ttl=64
81	71.835226417	172.16.20.1	172.16.21.253	ICMP	98 Echo (ping) request	id=0xbcf, seq=8/2048, ttl=64
82	71.835359607	172.16.21.253	172.16.20.1	ICMP	98 Echo (ping) reply	id=0xbcf, seq=8/2048, ttl=64

Figura 37: Log 2 da Exp. 3 - Ping de tux23 para tux24.eth1

No.	Time	Source	Destination		
107	186.248792368	172.16.21.1	172.16.20.1		
108	187.263338163	172.16.20.1	172.16.21.1		
109	187.263553625	172.16.21.1	172.16.20.1		
110	188.209877753	Routerbo_ic:9f:5b	Spanning-tree-(for.. STP)		
111	188.287336663	172.16.20.1	172.16.21.1		
112	188.287529218	172.16.21.1	172.16.20.1		
113	189.311338237	172.16.20.1	172.16.21.1		
114	189.311527579	172.16.21.1	172.16.20.1		
115	190.212181695	Routerbo_ic:9f:5b	Spanning-tree-(for.. STP)		
116	190.335336878	172.16.20.1	172.16.21.1		
117	190.335529572	172.16.21.1	172.16.20.1		
118	191.359372953	172.16.20.1	172.16.21.1		
119	191.359546371	172.16.21.1	172.16.20.1		
120	191.500020851	HewlettP_a6:a4:f1	HewlettP_5a:7d		
121	191.500134554	HewlettP_5a:7d:12	HewlettP_a6:a4:f1		
122	192.214491435	Routerbo_ic:9f:5b	Spanning-tree-(for.. STP)		
123	192.383342051	172.16.20.1	172.16.21.1		
124	192.383555069	172.16.21.1	172.16.20.1		
125	193.407341389	172.16.20.1	172.16.21.1		
126	193.407540300	172.16.21.1	172.16.20.1		
127	194.216506789	Routerbo_ic:9f:5b	Spanning-tree-(for.. STP)		
128	194.431342195	172.16.20.1	172.16.21.1		
129	194.431540477	172.16.21.1	172.16.20.1		
130	195.455344048	172.16.20.1	172.16.21.1		
131	195.455531644	172.16.21.1	172.16.20.1		
132	196.218802490	Routerbo_ic:9f:5b	Spanning-tree-(for.. STP)		
133	196.218802490	172.16.20.1	172.16.21.1		
134	197.106.170272240	172.16.20.1	172.16.21.1		
135	198.209904781	Routerbo_ic:9f:5b	Spanning-tree-(for.. STP)		
136	198.447667733	172.16.20.1	172.16.21.1		
137	198.447817405	172.16.21.1	172.16.20.1		
138	199.471696266	172.16.20.1	172.16.21.1		
139	199.471849360	172.16.21.1	172.16.20.1		
140	199.471849360	172.16.20.1	172.16.20.1		

Figura 38: Log 3 da Exp. 3 - Ping de tux23 para tux22

Experiência 4

(À exceção do último, os logs desta experiência foram capturados na bancada 3 devido a problemas com a bancada 2)

4 0.000000000 172.16.30.1	172.16.30.254	ICMP	98 Echo (ping) request id=0xb2b2e, seq=1/256, ttl=64 (request in 0)
6 6.0147296725 Routerboardc_1c:9f:50	Spanning-tree-(for-brdg-)	STP	60 RST. Root = 32768/0/7/fcf1fb:fb:3a:fd:80 Cost = 0 Port = 0x8003
7 6.505309684 172.16.30.1	172.16.30.254	ICMP	98 Echo (ping) request id=0xb2b2e, seq=2/512, ttl=64 (reply in 8)
8 6.505560169 172.16.30.254	172.16.30.1	ICMP	98 Echo (ping) reply id=0xb2b2e, seq=2/512, ttl=64 (request in 7)
9 7.504436100 172.16.30.1	172.16.30.254	ICMP	98 Echo (ping) request id=0xb2b2e, seq=3/768, ttl=64 (reply in 10)
10 7.504454537 172.16.30.254	172.16.30.1	ICMP	98 Echo (ping) reply id=0xb2b2e, seq=3/768, ttl=64 (request in 9)
11 7.868693146 Routerboardc_1c:9f:50	Spanning-tree-(for-brdg-)	STP	60 RST. Root = 32768/0/7
12 8.019447451 Routerboardc_1c:9f:50	Spanning-tree-(for-brdg-)	STP	60 RST. Root = 32768/0/7/fcf1fb:fb:3a:fd:80 Cost = 0 Port = 0x8003
13 8.504442643 172.16.30.1	172.16.30.254	ICMP	98 Echo (ping) request id=0xb2b2e, seq=4/1024, ttl=64 (reply in 14)
14 8.5047840929 172.16.30.254	172.16.30.1	ICMP	98 Echo (ping) reply id=0xb2b2e, seq=4/1024, ttl=64 (request in 13)
15 18.824249368 Cisco_3a:fa:83	Spanning-tree-(for-bridge)	STP	60 Conf. Root = 32768/36/fcf1fb:fb:3a:fd:88 Cost = 0 Port = 0x8003
16 12.634422953 Routerboardc_1c:9f:50	Spanning-tree-(for-brdg-)	STP	60 RST. Root = 32768/0/7/fcf1fb:fb:3a:fd:80 Cost = 0 Port = 0x8003
17 12.618458957 172.16.30.1	172.16.30.254	ICMP	98 Echo (ping) request id=0xb2b2e, seq=1/256, ttl=64 (reply in 18)
18 12.618959512 172.16.30.1	172.16.30.1	ICMP	98 Echo (ping) reply id=0xb2b2e, seq=1/256, ttl=63 (request in 17)
19 13.617461355 172.16.30.1	172.16.31.1	ICMP	98 Echo (ping) request id=0xb2b2e, seq=2/512, ttl=64 (reply in 20)
20 13.617718982 172.16.31.1	172.16.30.1	ICMP	98 Echo (ping) reply id=0xb2b2e, seq=2/512, ttl=63 (request in 19)
21 18.631563494 Cisco_3a:fa:83	CDP/VTP/DTP/PagP/UDL	CDP	435 Device ID: tux-sw3 Port ID: FastEthernet0/1
22 14.034151975 Routerboardc_1c:9f:50	Spanning-tree-(for-brdg-)	STP	60 RST. Root = 32768/0/7/fcf1fb:fb:3a:fd:80 Cost = 0 Port = 0x8003
23 14.618465185 172.16.30.1	172.16.31.1	ICMP	98 Echo (ping) request id=0xb2b2e, seq=3/768, ttl=64 (reply in 24)
24 14.618937880 172.16.31.1	172.16.30.1	ICMP	98 Echo (ping) reply id=0xb2b2e, seq=3/768, ttl=63 (request in 23)
25 15.618443454 172.16.30.1	172.16.31.1	ICMP	98 Echo (ping) request id=0xb2b2e, seq=4/1024, ttl=64 (reply in 26)
26 15.618761723 172.16.31.1	172.16.30.1	ICMP	98 Echo (ping) reply id=0xb2b2e, seq=4/1024, ttl=63 (request in 25)
27 16.038597478 Routerboardc_1c:9f:50	Spanning-tree-(for-brdg-)	STP	60 RST. Root = 32768/0/7/fcf1fb:fb:3a:fd:80 Cost = 0 Port = 0x8003
28 16.616462022 172.16.30.1	172.16.31.1	ICMP	98 Echo (ping) request id=0xb2b2e, seq=5/1280, ttl=64 (reply in 29)

Figura 39: Log 1 da Exp. 4 - Ping de tux23 para tux24.eth0

33 28.946703541 172.16.30.1	172.16.31.254	ICMP	98 Echo (ping) request id=0xb2b39, seq=1/256, ttl=64 (reply in 34)
34 28.947443821 172.16.31.254	172.16.30.1	ICMP	98 Echo (ping) reply id=0xb2b39, seq=1/256, ttl=254 (request in 33)
35 21.947705363 172.16.30.1	172.16.31.254	ICMP	98 Echo (ping) request id=0xb2b39, seq=2/512, ttl=64 (reply in 36)
36 21.948329645 172.16.31.254	172.16.30.1	ICMP	98 Echo (ping) reply id=0xb2b39, seq=2/512, ttl=254 (request in 35)
37 22.615377571 Routerboardc_1c:9f:50	Spanning-tree-(for-brdg-)	STP	60 RST. Root = 32768/0/7/fcf1fb:fb:3a:fa:80 Cost = 0 Port = 0x8003
38 22.948458017 172.16.30.1	172.16.31.254	ICMP	98 Echo (ping) request id=0xb2b39, seq=3/768, ttl=64 (reply in 39)
39 22.949067767 172.16.30.1	172.16.30.1	ICMP	98 Echo (ping) reply id=0xb2b39, seq=3/768, ttl=254 (request in 38)
40 23.948474725 172.16.30.1	172.16.31.254	ICMP	98 Echo (ping) request id=0xb2b39, seq=4/1024, ttl=64 (reply in 41)
41 23.949059593 172.16.31.254	172.16.30.1	ICMP	98 Echo (ping) reply id=0xb2b39, seq=4/1024, ttl=254 (request in 40)
42 28.633691719 Routerboardc_1c:9f:50	Spanning-tree-(for-brdg-)	STP	60 RST. Root = 32768/0/7/fcf1fb:fb:3a:fa:80 Cost = 0 Port = 0x8003
43 24.948498437 172.16.30.1	172.16.31.254	ICMP	98 Echo (ping) request id=0xb2b39, seq=5/1280, ttl=64 (reply in 44)
44 24.949143997 172.16.31.254	172.16.30.1	ICMP	98 Echo (ping) reply id=0xb2b39, seq=5/1280, ttl=254 (request in 43)
45 25.948454087 172.16.30.1	172.16.31.254	ICMP	98 Echo (ping) request id=0xb2b39, seq=6/1536, ttl=64 (reply in 46)
46 25.949067664 172.16.31.254	172.16.30.1	ICMP	98 Echo (ping) reply id=0xb2b39, seq=6/1536, ttl=254 (request in 45)
47 25.960401053 G-ProCom_P_5a:7d:4d	HewlettP_5a:7d:74	ARP	42 Who has 172.16.30.254? Tell 172.16.30.1
48 25.960664199 HewlettP_5a:7d:74	G-ProCom_Bb:84:4d	ARP	60 172.16.30.254 is at 00:21:5a:5a:7d:74
49 26.021417217 HewlettP_5a:7d:74	G-ProCom_Bb:84:4d	ARP	60 Who has 172.16.30.1? Tell 172.16.30.254

Figura 40: Log 2 da Exp. 4 - Ping de tux23 para o router

24 10.010514439 Routerboardc_1c:9f...	Spanning-tree-(for-bridge)	STP	60 RST. Root = 32768/0/7/fcf1fb:fb:3a:fd:80 Cost = 0 Port = 0x8003
25 10.495869424 172.16.21.1	172.16.20.1	ICMP	98 Echo (ping) request id=0xb2b39, seq=1/256, ttl=64 (request in 33)
26 10.496075879 172.16.21.254	172.16.21.1	ICMP	126 Redirect
27 10.496306149 172.16.20.1	172.16.21.1	ICMP	98 Echo (ping) reply id=0xb2b39, seq=1/256, ttl=254 (request in 32)
28 11.519865381 172.16.21.1	172.16.20.1	ICMP	98 Echo (ping) request id=0xb2b39, seq=2/512, ttl=64 (reply in 41)
29 11.520076794 172.16.21.254	172.16.21.1	ICMP	126 Redirect
30 11.520306994 172.16.20.1	172.16.21.1	ICMP	98 Echo (ping) reply id=0xb2b39, seq=2/512, ttl=254 (request in 40)
31 12.012366909 Routerboardc_1c:9f...	Spanning-tree-(for-bridge)	STP	60 RST. Root = 32768/0/7/fcf1fb:fb:3a:fd:80 Cost = 0 Port = 0x8003
32 12.543862294 172.16.21.1	172.16.20.1	ICMP	98 Echo (ping) request id=0xb2b39, seq=3/768, ttl=64 (request in 43)
33 12.544059808 172.16.21.254	172.16.21.1	ICMP	126 Redirect
34 12.544350003 172.16.20.1	172.16.21.1	ICMP	98 Echo (ping) reply id=0xb2b39, seq=3/768, ttl=254 (request in 42)
35 13.264342256 172.16.21.1	172.16.1.1	DNS	86 Standard query 0
36 13.265175826 172.16.1.1	172.16.21.1	DNS	138 Standard query response
37 13.265336254 172.16.21.1	172.16.1.1	DNS	84 Standard query 0
38 12.266300059 172.16.1.1	172.16.21.1	DNS	124 Standard query response
> Frame 26: 126 bytes on wire (1008 bits), 126 bytes captured			
> Ethernet II, Src: Routerboardc_ea:74:a5 (74:4d:00:00:00:00)			
> Internet Protocol Version 4, Src: 172.16.21.254			
> Internet Control Message Protocol			
Type: 5 (Redirect)			
Code: 1 (Redirect for host)			
Checksum: 0x38f1 [correct]			
[Checksum Status: Good]			
Gateway Address: 172.16.21.253			
> Internet Protocol Version 4, Src: 172.16.21.1			

Figura 41: Log 3 da Exp. 4 - Ping de tux22 para tux23, redirecionado pelo router através de tux24.eth1, com NAT desativado

Experiência 5

21	15.877682966	172.16.21.253	172.16.1.1	DNS	73 Standard query 0xcebd A www.google.pt
22	15.877694490	172.16.21.253	172.16.1.1	DNS	73 Standard query 0xfc7 AAAA www.google.pt
23	15.891595258	172.16.1.1	172.16.21.253	DNS	101 Standard query response 0xfc7 AAAA www.google.pt
24	15.891620889	172.16.1.1	172.16.21.253	DNS	89 Standard query response 0xcebd A www.google.pt
25	15.892003549	172.16.21.253	142.250.200.99	ICMP	98 Echo (ping) request id=0x412c, seq=1/256, tt
26	15.911912822	142.250.200.99	172.16.21.253	ICMP	98 Echo (ping) reply id=0x412c, seq=1/256, tt
27	15.912032879	172.16.21.253	172.16.1.1	DNS	87 Standard query 0x905d PTR 99.200.250.142.in-a
28	15.912744421	172.16.1.1	172.16.21.253	DNS	125 Standard query response 0x905d PTR 99.200.250
29	16.016428387	Routerbo_ic:9f:51	Spanning-tree-(for-) STP		60 RST, Root = 32768/0/74:4d:28:ea:74:a5 Cost =
30	16.893659184	172.16.21.253	142.250.200.99	ICMP	98 Echo (ping) request id=0x412c, seq=2/512, tt
31	16.912823182	142.250.200.99	172.16.21.253	ICMP	98 Echo (ping) reply id=0x412c, seq=2/512, tt
32	17.894910579	172.16.21.253	142.250.200.99	ICMP	98 Echo (ping) request id=0x412c, seq=3/768, tt
33	17.914926849	142.250.200.99	172.16.21.253	ICMP	98 Echo (ping) reply id=0x412c, seq=3/768, tt
34	18.018471202	Routerbo_ic:9f:51	Spanning-tree-(for-) STP		60 RST, Root = 32768/0/74:4d:28:ea:74:a5 Cost =
35	18.896006507	172.16.21.253	142.250.200.99	ICMP	98 Echo (ping) request id=0x412c, seq=4/1024, tt
36	18.915793279	142.250.200.99	172.16.21.253	ICMP	98 Echo (ping) reply id=0x412c, seq=4/1024, tt
37	19.897872924	172.16.21.253	142.250.200.99	ICMP	98 Echo (ping) request id=0x412c, seq=5/1280, tt
38	19.916743659	142.250.200.99	172.16.21.253	ICMP	98 Echo (ping) reply id=0x412c, seq=5/1280, tt
39	20.020558019	Routerbo_ic:9f:51	Spanning-tree-(for-) STP		60 RST, Root = 32768/0/74:4d:28:ea:74:a5 Cost =
40	20.899826572	172.16.21.253	142.250.200.99	ICMP	98 Echo (ping) request id=0x412c, seq=6/1536, tt
41	20.918716933	142.250.200.99	172.16.21.253	ICMP	98 Echo (ping) reply id=0x412c, seq=6/1536, tt
42	21.901799706	172.16.21.253	142.250.200.99	ICMP	98 Echo (ping) request id=0x412c, seq=7/1792, tt
43	21.920798879	142.250.200.99	172.16.21.253	ICMP	98 Echo (ping) reply id=0x412c, seq=7/1792, tt
44	22.022579324	Routerbo_ic:9f:51	Spanning-tree-(for-) STP		60 RST, Root = 32768/0/74:4d:28:ea:74:a5 Cost =
45	22.902880480	172.16.21.253	142.250.200.99	ICMP	98 Echo (ping) request id=0x412c, seq=8/2048, tt
46	22.921797031	142.250.200.99	172.16.21.253	ICMP	98 Echo (ping) reply id=0x412c, seq=8/2048, tt

Figura 42: Log da Exp. 5 - Funcionamento do DNS, com o comando ping

Experiência 6

.. 3.8600033867	192.168.109.136	172.16.20.1	TCP	66 21 → 51930 [ACK] Seq=92 Ack=24 Win=65280 Len=0 TSval=
.. 3.860040921	172.16.20.1	192.168.109.136	FTP	73 Request: files
.. 3.860663625	192.168.109.136	172.16.20.1	TCP	66 21 → 51930 [ACK] Seq=92 Ack=31 Win=65280 Len=0 TSval=
.. 3.860723060	192.168.109.136	172.16.20.1	FTP	1... Response: 256 Directory successfully changed.
.. 3.860770971	172.16.20.1	192.168.109.136	FTP	70 Request: pasv
.. 3.861296456	192.168.109.136	172.16.20.1	TCP	66 21 → 51930 [ACK] Seq=129 Ack=35 Win=65280 Len=0 TSval=
.. 3.861303650	172.16.20.1	192.168.109.136	FTP	68 Request:
.. 3.861844500	192.168.109.136	172.16.20.1	TCP	66 21 → 51930 [ACK] Seq=129 Ack=37 Win=65280 Len=0 TSval=
.. 3.861965884	192.168.109.136	172.16.20.1	FTP	1... Response: 227 Entering Passive Mode (192,168,109,136,
.. 3.862034538	172.16.20.1	192.168.109.136	TCP	74 57490 → 42405 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SA
.. 3.862563027	192.168.109.136	172.16.20.1	TCP	74 42405 → 57490 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0
.. 3.862573992	172.16.20.1	192.168.109.136	TCP	66 57490 → 42405 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=
.. 3.862610449	172.16.20.1	192.168.109.136	FTP	70 Request: retr
.. 3.863127553	192.168.109.136	172.16.20.1	TCP	66 21 → 51930 [ACK] Seq=183 Ack=41 Win=65280 Len=0 TSval=
.. 3.863134677	172.16.20.1	192.168.109.136	FTP	76 Request: pic1.jpg
.. 3.863628524	192.168.109.136	172.16.20.1	TCP	66 21 → 51930 [ACK] Seq=183 Ack=51 Win=65280 Len=0 TSval=
.. 3.863763946	192.168.109.136	172.16.20.1	FTP	1... Response: 150 Opening BINARY mode data connection for
.. 3.864192212	192.168.109.136	172.16.20.1	FTP-D... 1...	FTP Data: 1448 bytes (PASV) (retr)
.. 3.864201990	172.16.20.1	192.168.109.136	TCP	66 57490 → 42405 [ACK] Seq=1 Ack=1449 Win=64128 Len=0 TS
.. 3.864314504	192.168.109.136	172.16.20.1	FTP-D... 1...	FTP Data: 1448 bytes (PASV) (retr)
.. 3.864322885	172.16.20.1	192.168.109.136	TCP	66 57490 → 42405 [ACK] Seq=1 Ack=2897 Win=63488 Len=0 TS
.. 3.864436098	192.168.109.136	172.16.20.1	FTP-D... 1...	FTP Data: 1448 bytes (PASV) (retr)
.. 3.864443431	172.16.20.1	192.168.109.136	TCP	66 57490 → 42405 [ACK] Seq=1 Ack=4345 Win=62080 Len=0 TS
.. 3.864558459	192.168.109.136	172.16.20.1	FTP-D... 1...	FTP Data: 1448 bytes (PASV) (retr)
.. 3.864566072	172.16.20.1	192.168.109.136	TCP	66 57490 → 42405 [ACK] Seq=1 Ack=5793 Win=60672 Len=0 TS
.. 3.864681170	192.168.109.136	172.16.20.1	FTP-D... 1...	FTP Data: 1448 bytes (PASV) (retr)
.. 3.864687037	172.16.20.1	192.168.109.136	TCP	66 57490 → 42405 [ACK] Seq=1 Ack=7241 Win=60544 Len=0 TS
.. 3.864805208	192.168.109.136	172.16.20.1	FTP-D... 1...	FTP Data: 1448 bytes (PASV) (retr)
.. 3.864811634	172.16.20.1	192.168.109.136	TCP	66 57490 → 42405 [ACK] Seq=1 Ack=8689 Win=59520 Len=0 TS
.. 3.864929945	192.168.109.136	172.16.20.1	FTP-D... 1...	FTP Data: 1448 bytes (PASV) (retr)

Figura 43: Log 1 da Exp. 6 - Transferência de um ficheiro, utilizando a aplicação *download*

.. 13.995308393	193.137.29.15	172.16.20.1	FTP-D... 1...	[TCP Previous segment not captured] FTP Data: 1368 by
.. 13.995375028	172.16.20.1	193.137.29.15	TCP	78 42066 → 50083 [ACK] Seq=1 Ack=96441 Win=414720 Len=6
.. 13.995486916	193.137.29.15	172.16.20.1	FTP-D... 1...	[TCP Previous segment not captured] FTP Data: 1368 by
.. 13.995493062	172.16.20.1	193.137.29.15	TCP	86 [TCP Dup ACK 1211#1] 42066 → 50083 [ACK] Seq=1 Ack=96
.. 13.995603413	193.137.29.15	172.16.20.1	FTP-D... 1...	FTP Data: 1368 bytes (PASV) (retr)
.. 13.995609419	172.16.20.1	193.137.29.15	TCP	86 [TCP Dup ACK 1211#2] 42066 → 50083 [ACK] Seq=1 Ack=96
.. 13.995721865	193.137.29.15	172.16.20.1	FTP-D... 1...	FTP Data: 1368 bytes (PASV) (retr)
.. 13.995834381	193.137.29.15	172.16.20.1	FTP-D... 1...	FTP Data: 1368 bytes (PASV) (retr)
.. 13.995847651	172.16.20.1	193.137.29.15	TCP	86 [TCP Dup ACK 1211#3] 42066 → 50083 [ACK] Seq=1 Ack=96
.. 13.995933627	193.137.29.15	172.16.20.1	FTP-D... 1...	FTP Data: 1368 bytes (PASV) (retr)
.. 13.996075337	193.137.29.15	172.16.20.1	FTP-D... 1...	FTP Data: 1368 bytes (PASV) (retr)
.. 13.996183802	193.137.29.15	172.16.20.1	FTP-D... 1...	FTP Data: 1368 bytes (PASV) (retr)
.. 13.996200425	172.16.20.1	193.137.29.15	TCP	86 [TCP Window Update] 42066 → 50083 [ACK] Seq=1 Ack=964
.. 13.996288147	193.137.29.15	172.16.20.1	FTP-D... 1...	FTP Data: 1368 bytes (PASV) (retr)
.. 13.996413304	172.16.20.1	193.137.29.15	TCP	86 [TCP Dup ACK 1211#4] 42066 → 50083 [ACK] Seq=1 Ack=96
.. 13.996419869	193.137.29.15	172.16.20.1	FTP-D... 1...	FTP Data: 1368 bytes (PASV) (retr)
.. 13.996533993	193.137.29.15	172.16.20.1	FTP-D... 1...	FTP Data: 1368 bytes (PASV) (retr)
.. 13.996544538	172.16.20.1	193.137.29.15	TCP	86 [TCP Dup ACK 1211#5] 42066 → 50083 [ACK] Seq=1 Ack=96
.. 13.996655657	193.137.29.15	172.16.20.1	FTP-D... 1...	FTP Data: 1368 bytes (PASV) (retr)
.. 13.996767125	193.137.29.15	172.16.20.1	FTP-D... 1...	FTP Data: 1368 bytes (PASV) (retr)
.. 13.996780395	172.16.20.1	193.137.29.15	TCP	86 [TCP Dup ACK 1211#6] 42066 → 50083 [ACK] Seq=1 Ack=96
.. 13.996866441	193.137.29.15	172.16.20.1	FTP-D... 1...	FTP Data: 1368 bytes (PASV) (retr)
.. 13.997001166	193.137.29.15	172.16.20.1	FTP-D... 1...	FTP Data: 1368 bytes (PASV) (retr)
.. 13.997117873	193.137.29.15	172.16.20.1	FTP-D... 1...	FTP Data: 1368 bytes (PASV) (retr)
.. 13.997126743	172.16.20.1	193.137.29.15	TCP	86 [TCP Dup ACK 1211#7] 42066 → 50083 [ACK] Seq=1 Ack=96
.. 13.997237233	193.137.29.15	172.16.20.1	FTP-D... 1...	FTP Data: 1368 bytes (PASV) (retr)
.. 13.997350448	193.137.29.15	172.16.20.1	FTP-D... 1...	FTP Data: 1368 bytes (PASV) (retr)
.. 13.997361832	172.16.20.1	193.137.29.15	TCP	86 [TCP Dup ACK 1211#8] 42066 → 50083 [ACK] Seq=1 Ack=96
42.007144020C	402.407.00.4E	472.42.00.4	FTP-D... 4...	FTP Data: 1368 bytes (PASV) (retr)

Figura 44: Log 2 da Exp. 6 - Transferência simultânea de ficheiros em tux23 e tux22