RCOM 2020/2021

Relatório - 2º Trabalho laboratorial

Eduardo Brito, up201806271
Pedro Ferreira, up201806506
Turma 1, Grupo 11

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Sumário

Este relatório documenta a jornada de trabalho, na unidade curricular de RCOM, em torno do desenvolvimento de uma aplicação de *download* de ficheiros, pelo protocolo FTP, e a criação e configuração de uma rede de computadores. O ambiente de desenvolvimento estabeleceu-se com o sistema operativo LINUX, a linguagem de programação C e recorrendo aos dispositivos CISCO presentes no laboratório.

Como conclusões, retiradas da concretização deste trabalho, são de destacar a consolidação efetiva da aprendizagem no campo dos protocolos de troca de informação, o estudo dos mecanismos por detrás dos mesmos, e o amadurecimento em relação à organização do trabalho, à estruturação do código, à validação teórica das formulações e de tudo o resto que envolveu a implementação deste projeto.

Conteúdo

Este relatório encontra-se organizado em duas partes distintas. Na primeira, será abordada a criação de uma aplicação de *download* de ficheiros, através do protocolo FTP. Na segunda parte, será abordada a configuração de uma rede, onde, entre outros objetivos, a aplicação será testada. A estrutura geral do relatório é a seguinte:

- Introdução - Objetivos gerais do trabalho realizado.
- Primeira Parte - Desenvolvimento da aplicação de *download*
    - Arquitetura - Funcionamento geral e estruturação da aplicação
    - Resultados - Descrição e testes efetuados
- Segunda Parte - Configuração e estudo de uma rede de computadores
    - Configurar uma rede IP - Configuração de redes IP, de modo que a comunicação entre duas máquinas seja possível.
    - Implementar duas LANs virtuais num Switch - Implementação de duas VLANs, tendo a primeira dois TUXs ligados e a segunda apenas um.
    - Configurar um Router em Linux - Configuração do TUX24 de forma a funcionar como Router entre as VLANs criadas na experiência anterior.
    - Configurar um Router comercial com utilização de NAT - Configuração de um Router comercial com NAT, possibilitando a comunicação da rede criada com redes externas.
    - DNS - Configuração do serviço DNS nos computadores da rede criada.
    - Ligações TCP - Utilização e teste da aplicação criada na rede configurada.
- Conclusões - síntese da reflexão feita no desenvolvimento do trabalho prático e possíveis recomendações.
- Anexos - Código referente à aplicação criada e scripts usados nas experiências realizadas.

Introdução

O seguinte trabalho foi realizado no âmbito da disciplina de Redes de Computadores (RCOM), tendo como tema o estudo de uma rede de computadores.

Os principais objetivos assentam no desenvolvimento de uma aplicação de transferência de ficheiros, recorrendo à utilização do protocolo FTP (File Transfer Protocol), e na configuração de uma rede com vários dispositivos, aprofundando diversos conceitos importantes relacionados com o tema, tais como o funcionamento de certos dispositivos - Switches e Routers, e o aprofundamento do conhecimento acerca de protocolos e técnicas, entre elas, NAT, DNS, ARQ, ligações TCP (Transmission Control Protocol), e vários outros assuntos.

Aplicação de *download*

Assente no protocolo de transferência de ficheiros (FTP), foi desenvolvida uma aplicação que permite testar a rede criada, no âmbito geral deste projeto. A implementação desta aplicação teve em conta as normas `RFC 959` (leitura e análise das respostas do servidor) e `RFC 1738` (utilização e tratamento dos endereços URL). A aplicação segue uma estrutura de organização funcional e é executada sequencialmente, automatizando o processo de comunicação com o servidor, como um script mecanizado e resiliente a erros. O processo é executado partindo do módulo `ftp-client.c` e recorre a várias funções utilitárias, presentes no módulo `utils.h`, que permitem gerir e concretizar os procedimentos necessários. Existe, ainda, uma máquina de estados, no módulo `state.h`, que permite a deteção e avaliação dos códigos numéricos de três dígitos, enviados pelo servidor.

A aplicação começa por avaliar os argumentos da linha de comandos, fornecidos pelo utilizador. A função

```
ftp_uri *parse_arguments(char *uri)
```

soluciona esta questão recorrendo a um mecanismo de avaliação de expressões regulares que se compromete em garantir o preenchimento correto de todos os campos necessários. A exceção é apenas o *username* e *password* que, no caso de não serem fornecidos, são definidos com os valores `anonymous` e `password`, respetivamente. De seguida, é obtida a informação relacionada com a entidade, através da função

```
struct hostent *get_ip(char *host)
```

que fornece, entre outros campos, o endereço IP do servidor com o qual se vai comunicar, e abre-se, então, a primeira ligação ao servidor, através de um socket que se conecta à porta 21 - indicada para conexões FTP. Um a um e em sequência, são enviados automaticamente os comandos e recebidas as respostas, com a avaliação funcional dos códigos de 3 dígitos que as acompanham. A função usada é

```
int send_command(int socket_fd, char *cmd, char *buffer, res_code *code)
```

que envia o comando e armazena a resposta e o código de controlo, recorrendo, para isto, à função

```
int get_response(int socket_fd, res_code *code)
```

que dá uso, internamente, à máquina de estados referida em cima. Convém notar a necessidade de avaliar o código de controlo, em cada resposta recebida, e, em especial, avaliar o primeiro dígito do código e proceder conforme o especificado nas normas, o que é feito em todos os comandos e respetivas respostas. Os comandos enviados, por ordem, são os seguintes:

1. `user username`

2. `pass password` - a ignorar quando não for obrigatório especificar, em alguns servidores.
   Neste ponto, o login foi aceite e pode-se proceder ao pedido do ficheiro.
3. `pasv` - para entrar em modo passivo e requisitar uma nova conexão paralela e individual.
   Aqui é recebida a informação necessária para criar a segunda conexão e abrir o socket correspondente. A porta a aceder é calculada pela função

```
int parse_connection(char *buffer, char *ip_address)
```

   que, através de expressões regulares, captura o endereço IP e as informações para gerar a nova porta. Segue-se, então, a criação do socket.
4. `retr filename` - enviado para a conexão original na porta 21.

Neste ponto, tendo sido aceite o pedido, na nova conexão, começam a surgir os dados do ficheiro a ser transferido. A função

```
int download_file(int socket_client_fd, char *filename)
```

fica responsável por receber os bytes e guardá-los localmente num ficheiro com o mesmo nome. Assim que a função `read` do sistema UNIX terminar de ler o que estiver no socket, a conexão é fechada e é feita uma última requisição de estado à porta original que responderá com o estado da transferência. Em todo o caso, o processo termina e esta conexão é concluída.

Esta aplicação foi testada de múltiplas formas, tendo passado, com sucesso, a todos os testes efetuados. Os testes realizados encontram-se no ficheiro `test.sh` e demonstram a validação do processo descrito, em servidores diferentes e com argumentos diferentes.

Experiência 1

Esta experiência teve por objetivo a configuração de redes IP, de modo a que a comunicação entre duas máquinas fosse possível (TUX23 e TUX24). Para tal, foi necessário recorrer ao comando *ifconfig*, estabelecendo o endereço IP das interfaces de ambos os dispositivos, e ao comando *route*, para criar uma rota de troca de pacotes entre as máquinas, adicionando-a à tabela de reencaminhamento do TUX23.

(TUX23) `ifconfig eth0 up 172.16.20.1/24 ; route add default gw 172.16.20.254`

(TUX24) `ifconfig eth0 up 172.16.20.254/24`

No curso desta experiência, as tabelas ARP foram removidas, antes da tentativa de comunicação, e este ato permitiu observar um fenómeno relacionado com o Address Resolution Protocol (ARP), um protocolo utilizado na resolução de endereços da camada de rede (endereços IP) em endereços da camada de ligação de dados (ethernet). Deste modo, tendo sido eliminada a tabela ARP no TUX23, este deixa de ter o endereço MAC associado ao TUX24. Assim, ao tentar comunicar com o TUX24, um pacote ARP tem de ser enviado primeiro. Este pacote é difundido em *broadcast*, para todas as máquinas conectadas à mesma subrede, na tentativa de descobrir o endereço MAC correspondente ao endereço IP com o qual quer comunicar. Existindo tal dispositivo, vai receber um pacote com uma resposta contendo o endereço MAC correspondente ao TUX24 (00:08:54:50:3f2c), permitindo a comunicação entre os computadores.

No.	Time	Source	Destination	Protocol	Length	Info
5	5.8235…	HewlettP_5a:7d:12	Broadcast	ARP	42	Who has 172.16.20.254? Tell 172.16.20.1
6	5.8236…	Netronix_50:3f:2c	HewlettP_5a:7d:12	ARP	60	172.16.20.254 is at 00:08:54:50:3f:2c

Figura 1.1 - Pacotes ARP trocados entre o TUX23 e o TUX24, revelando o seu endereço MAC.

O comando *ping*, depois dos pacotes ARP e da obtenção do endereço MAC, passa a enviar pacotes de solicitação ICMP (Internet Control Message Protocol) para o recetor e aguarda uma resposta ICMP, como visível no extrato seguinte:

No.	Time	Source	Destination	Protocol	Length	Info
7	5.8236…	172.16.20.1	172.16.20.254	ICMP	98	Echo (ping) request id=0x37fc, seq=1/256, ttl=64 (reply in 8)
8	5.8237…	172.16.20.254	172.16.20.1	ICMP	98	Echo (ping) reply id=0x37fc, seq=1/256, ttl=64 (request in 7)

Figura 1.2 - Comandos *ping* enviados do TUX23 para o TUX24.

Os pacotes ICMP são trocados entre o TUX23, de IP 172.16.20.1 e endereço MAC 00:21:5a:5a:7d:12, e o TUX24, com o IP 172.16.20.254 e MAC 00:08:54:50:3f2c. Esta informação é equivalente à trocada pelos pacotes ARP, descritos acima. Para fazer a distinção entre uma trama ARP e IP, é necessário avaliar o seu cabeçalho, correspondente aos 2 primeiros bytes do pacote. O valor 0x0806 revela uma trama do tipo ARP e o valor 0x0800 indica uma trama IP. Como ICMP é um subprotocolo do protocolo IP, estes pacotes são identificados pelo valor do IP Header - 1 indica um pacote ICMP, caso contrário é do tipo genérico IP. O comprimento destes pacotes está contido no seu cabeçalho.

Durante esta experiência, verificou-se a existência da interface *loopback*. Esta interface encontra-se sempre ativa e acessível, desde que a rota para esse endereço IP esteja disponível na tabela de reencaminhamento da máquina, podendo ser utilizada para fins de diagnóstico e solução de problemas. De facto, testemunhou-se o envio de pacotes LOOP, com intervalo de alguns segundos, que permitiam verificar o estado da ligação e a configuração da carta de rede.

Experiência 2

Esta experiência seguiu com a criação de duas VLANs (VLAN20 e VLAN21), tendo a primeira, duas máquinas ligadas (TUX23 e TUX24) e a segunda, apenas uma (TUX22). Deste modo, o TUX22, como se encontra numa subrede diferente, não tem acesso aos TUXs que se encontram na VLAN20. Os comandos seguintes, que permitiram a criação das VLANs, foram introduzidos no Switch:

```
# CRIAR VLANS

conf t
vlan 20
vlan 21
end
```

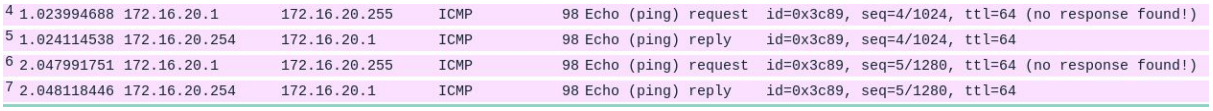
```
# TUX 3 ETH0

conf t
interface fastethernet 0/1
switchport mode access
switchport access vlan 20
end
```

```
# TUX 4 ETH0

conf t
interface fastethernet 0/3
switchport mode access
switchport access vlan 20
end
```

```
# TUX 2 ETH0

conf t
interface fastethernet 0/2
switchport mode access
switchport access vlan 21
end
```

Posto isto, a comunicação entre máquinas ficou só disponível, neste ponto da experiência, para os TUX23 e TUX24, na VLAN20, com o TUX22 a ficar, por enquanto, isolado na sua subrede e sem capacidade de comunicar com os restantes.

Configuradas as VLANs, foi chamado o comando `ping -b 172.16.(VLAN).255` a partir do TUX23 e do TUX22. Através dos logs, verifica-se que o TUX23 obteve resposta do TUX24, enquanto o TUX22 não obteve qualquer resposta.



4	1.023994688	172.16.20.1	172.16.20.255	ICMP	98	Echo (ping) request id=0x3c89, seq=4/1024, ttl=64 (no response found!)
5	1.024114538	172.16.20.254	172.16.20.1	ICMP	98	Echo (ping) reply id=0x3c89, seq=4/1024, ttl=64
6	2.047991751	172.16.20.1	172.16.20.255	ICMP	98	Echo (ping) request id=0x3c89, seq=5/1280, ttl=64 (no response found!)
7	2.048118446	172.16.20.254	172.16.20.1	ICMP	98	Echo (ping) reply id=0x3c89, seq=5/1280, ttl=64

Figura 2.1 - Envio do comando *ping* em *broadcast*, a partir do TUX23, com resposta do TUX24.

Assim, concluímos que existem dois domínios de broadcast diferentes, cada um correspondente à sua VLAN.

Experiência 3

O objetivo da seguinte experiência foi a conversão do TUX24 num Router que permitisse uma ligação entre as VLANs criadas na experiência anterior (VLAN20 e VLAN21). A sua interface ETH1 foi configurada, estabelecendo um novo endereço IP (172.16.21.253) e adicionando-a à VLAN21, abrindo uma nova porta também no Switch. Tanto no TUX22 como no TUX23 foi adicionada uma nova rota, para comunicar com a VLAN oposta, recorrendo ao TUX24. Estas rotas foram criadas a partir dos seguintes comandos:

```
route add -net 172.16.20.0/24 gw 172.16.21.253
route add -net 172.16.21.0/24 gw 172.16.20.254
```

As tabelas de reencaminhamento destas máquinas passaram a ter a informação sobre estas rotas, com a identificação do endereço para o qual os pacotes devem ser reencaminhados, para cada endereço ou gama de endereços IP de destino final. Casos não especificados por estas rotas ficam abrangidos pelas entradas *default* já existentes nas tabelas.

Nas posteriores tentativas de comunicação, verificou-se os pedidos das mensagens ARP em relação ao endereço físico da *gateway* e não do destino final, como definido pelas rotas estabelecidas. Estas interfaces intermédias atuam, assim, como ponte de reencaminhamento de pacotes entre as duas VLANs. A partir deste momento, são já observados pacotes ICMP request e ICMP reply trocados entre todos os pares de TUXs.

Netronix_50:3f:2c	HewlettP_5a:7d:12	ARP	42 Who has 172.16.20.1? Tell 172.16.20.254
HewlettP_5a:7d:12	Netronix_50:3f:2c	ARP	60 172.16.20.1 is at 00:21:5a:5a:7d:12
172.16.20.1	172.16.21.1	ICMP	98 Echo (ping) request id=0x3df4, seq=6/153
172.16.21.1	172.16.20.1	ICMP	98 Echo (ping) reply id=0x3df4, seq=6/153
HewlettP_5a:7d:12	Netronix_50:3f:2c	ARP	60 Who has 172.16.20.254? Tell 172.16.20.1
Netronix_50:3f:2c	HewlettP_5a:7d:12	ARP	42 172.16.20.254 is at 00:08:54:50:3f:2c

HewlettP_a6:a4:f1	HewlettP_61:2b:72	ARP	42 Who has 172.16.21.1? Tell 172.16.21.253
HewlettP_61:2b:72	HewlettP_a6:a4:f1	ARP	60 172.16.21.1 is at 00:21:5a:61:2b:72
172.16.20.1	172.16.21.1	ICMP	98 Echo (ping) request id=0x3df4, seq=6/153
172.16.21.1	172.16.20.1	ICMP	98 Echo (ping) reply id=0x3df4, seq=6/153
HewlettP_61:2b:72	HewlettP_a6:a4:f1	ARP	60 Who has 172.16.21.253? Tell 172.16.21.1
HewlettP_a6:a4:f1	HewlettP_61:2b:72	ARP	42 172.16.21.253 is at 00:22:64:a6:a4:f1

Figura 3.1 - Comunicação entre o TUX23 e TUX22. Do lado esquerdo, a captura na interface ETH0 e do lado direito, na interface ETH1, do TUX24.

Focando na ligação entre o TUX23 e TUX22, os pacotes ICMP request, capturados na interface ETH0 do TUX24, contêm, como endereço de destino, o endereço MAC do TUX24, e os pacotes ICMP reply contêm, como endereço de origem, mesmo o endereço, uma vez que é este que faz a ponte e o redirecionamento da comunicação entre as duas VLANs. Algo análogo se verifica na interface ETH1, com os mesmos pacotes, como exemplificado no diagrama seguinte.

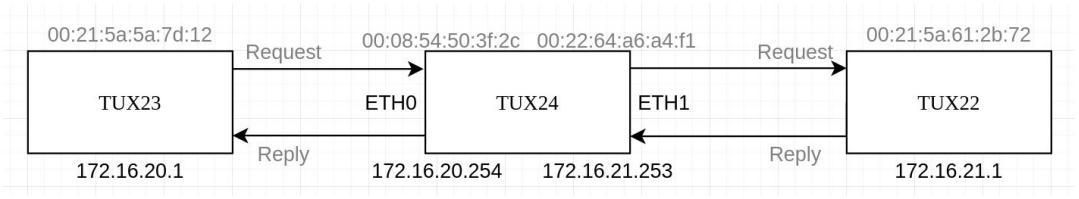


Figura 3.2 - Comunicação entre o TUX23 e TUX22. Rota seguida pelos pacotes ICMP, passando pelo TUX24.

Experiência 4

Nesta experiência, foi adicionado mais um dispositivo à rede. Um router comercial foi configurado com determinados endereços IP (172.16.21.254 na interface GE0 e 172.16.1.29 na interface GE1) e, por isso, inserido na VLAN 21, com rotas estáticas especificamente definidas, tanto para comunicar com a VLAN 20, como para servir de ponte para o resto da internet pública. Os comandos que permitem a criação destas rotas são

```
ip route 0.0.0.0 0.0.0.0 172.16.1.254
ip route 172.16.20.0 255.255.255.0 172.16.21.253
```

com formato geral `ip route prefix mask (ip-address | interface-type interface-number [ip-address])`. A segunda rota é importante porque permite o reencaminhamento de pacotes, direcionados para a VLAN20, para a interface ETH1 do TUX24, que, assim, procede à sua distribuição nessa subrede. A primeira rota é responsável por encaminhar todos os outros pacotes para o router presente no laboratório que, depois, fará a respetiva gestão.

Seguindo o curso da experiência, sem redirecionamento ICMP e sem a rota do TUX22 para o TUX24, ao fazer ping no TUX22 para o TUX23, verifica-se (através do comando *traceroute*) que os pacotes vão até ao RC onde são reencaminhados para o TUX24 e daí para o TUX23. Ao ser atribuída, novamente, a rota do TUX22 para o TUX24, os pacotes são diretamente encaminhados para o TUX24 e depois para o TUX23. Por fim, ao remover, mais uma vez, a rota direta do TUX22 para o TUX24, e ao ativar o redirecionamento ICMP, quando se enviam os pacotes, o RC responde apenas uma vez ao TUX22 com *ICMP Redirect, o* que permite a este computador fazer a ligação entre si e o TUX24, de forma a otimizar o envio de pacotes e a rota passa a ser, de novo, direta.

```
2 0.129205784 172.16.21.1     172.16.20.1     ICMP  98 Echo (ping) request  id=0x74d7, seq=13/3328, ttl=64 (reply in 4)
3 0.129504217 172.16.21.254   172.16.21.1     ICMP  70 Redirect             (Redirect for host)
4 0.129875914 172.16.20.1     172.16.21.1     ICMP  98 Echo (ping) reply    id=0x74d7, seq=13/3328, ttl=63 (request in 2)
5 1.149195361 172.16.21.1     172.16.20.1     ICMP  98 Echo (ping) request  id=0x74d7, seq=14/3584, ttl=64 (reply in 7)
6 1.149496449 172.16.21.254   172.16.21.1     ICMP  70 Redirect             (Redirect for host)
7 1.149866889 172.16.20.1     172.16.21.1     ICMP  98 Echo (ping) reply    id=0x74d7, seq=14/3584, ttl=63 (request in 5)
```

Figura 4.1 - No TUX22, sem rota para VLAN20 e sem redirecionamento ICMP.

No caso do TUX23 e da sua tentativa de comunicar com o router do laboratório (172.16.1.254), ainda sem NAT habilitado no RC, os pacotes saem, efetivamente, da rede, mas não conseguem encontrar o destinatário, quando regressam.

```
3 2.240594328 172.16.20.1     172.16.1.254    ICMP  98 Echo (ping) request  id=0x0383, seq=1/256, ttl=64 (no response found!)
4 3.254943253 172.16.20.1     172.16.1.254    ICMP  98 Echo (ping) request  id=0x0383, seq=2/512, ttl=64 (no response found!)
```

Figura 4.2 - No TUX23, ainda sem NAT configurado no RC.

Configurando o NAT (Network Address Translation) no RC, técnica que consiste em reescrever, através de uma tabela hash, os endereços IP de origem de um pacote, para que seja possível com apenas um endereço público servir vários endereços privados, fica possível essa comunicação entre qualquer dispositivo na rede e a Internet pública, através do RC. Assim, consegue-se poupar o espaço de endereçamento público, com um mecanismo que assenta na tradução do endereço e da porta do dispositivo num identificador que é inserido na tabela, para que, quando a resposta externa chegar, seja possível encaminhá-la para a interface correta.

Experiência 5

Neste ponto, é configurado o DNS nos dispositivos e adicionada uma nova entrada que permite aceder ao servidor FTP disponibilizado e existente na rede da faculdade. Isto é possível com o comando seguinte, que adiciona o nome *netlab1.fe.up.pt* à lista e condiciona a consulta de nomes, primeiramente, ao servidor DNS existente no laboratório:

```
printf "search netlab1.fe.up.pt\nnameserver 172.16.1.1\n" > /etc/resolv.conf
```

O Domain Name System (DNS) é um sistema de gestão de nomes que permite ao utilizador, através apenas de um nome, descobrir o endereço correspondente, na tabela de entradas que relaciona cada *hostname* ao respetivo endereço IP. Este é um sistema hierárquico e distribuído de gestão de nomes para computadores, serviços, ou qualquer máquina conectada à Internet, ou a uma rede

privada. Nesta experiência, os pacotes usam o protocolo User Datagram Protocol (UDP) na porta 53 do servidor e as mensagens DNS trocadas consistem num pedido UDP que parte do cliente e uma resposta UDP que vem do servidor. Este trata de resolver o nome requisitado e, eventualmente e assim que obtiver sucesso, responder com o endereço IP da máquina a comunicar, entre outras informações.

1	0.000000000	172.16.20.1	172.16.1.1	DNS	76	Standard query 0xc463 A netlab1.fe.up.pt
2	0.000009430	172.16.20.1	172.16.1.1	DNS	76	Standard query 0x866d AAAA netlab1.fe.up.pt
3	0.001648227	172.16.1.1	172.16.20.1	DNS	252	Standard query response 0xc463 A netlab1.fe.up.pt A 192.168.109.136 NS magoo.fe.up.pt
4	0.001747758	172.16.1.1	172.16.20.1	DNS	119	Standard query response 0x866d AAAA netlab1.fe.up.pt SOA ns1.fe.up.pt
5	0.001907147	172.16.20.1	192.168.109.136	ICMP	98	Echo (ping) request id=0x0643, seq=1/256, ttl=64 (reply in 6)
6	0.003011482	192.168.109.136	172.16.20.1	ICMP	98	Echo (ping) reply id=0x0643, seq=1/256, ttl=61 (request in 5)

Figura 5.1 - Mensagens DNS enviadas e recebidas quando executado `ping netlab1.fe.up.pt`

Experiência 6

Nesta experiência, a rede e a aplicação de download são testadas, realizando um conjunto de transferências de ficheiros com o protocolo FTP. Qualquer uma destas transferências abre 2 ligações TCP, ao longo da sua execução. A primeira acontece através da porta 21 do servidor, onde o cliente envia comandos de controlo e recebe as respetivas respostas, e a segunda, através da porta especificada na resposta ao comando *pasv*, onde receberá os dados do ficheiro.

TCP	74	40392 → 21 [SYN] Seq=0 Win=29200 Len=0 MSS=1
TCP	74	21 → 40392 [SYN, ACK] Seq=0 Ack=1 Win=65160
TCP	66	40392 → 21 [ACK] Seq=1 Ack=1 Win=29312 Len=0
FTP	100	Response: 220 Welcome to netlab-FTP server
TCP	66	40392 → 21 [ACK] Seq=1 Ack=35 Win=29312 Len=
FTP	76	Request: user rcom
TCP	66	21 → 40392 [ACK] Seq=35 Ack=11 Win=65280 Ler
FTP	100	Response: 331 Please specify the password.
FTP	76	Request: pass rcom

TCP	74	52452 → 40871 [SYN] Seq=0 Win=29200 Len=0 MSS
TCP	74	40871 → 52452 [SYN, ACK] Seq=0 Ack=1 Win=6516
TCP	66	52452 → 40871 [ACK] Seq=1 Ack=1 Win=29312 Len
FTP	80	Request: retr pipe.txt
TCP	66	21 → 40392 [ACK] Seq=146 Ack=40 Win=65280 Len
FTP	134	Response: 150 Opening BINARY mode data connec
FTP-DATA	1514	FTP Data: 1448 bytes (PASV) (retr pipe.txt)
TCP	66	52452 → 40871 [ACK] Seq=1 Ack=1449 Win=32128

Figura 6.1 - As duas ligações TCP abertas durante a transferência do ficheiro pipe.txt

Nestas ligações, o protocolo TCP tem 3 fases: estabelecimento da ligação (método *3 way handshake*), transferência de dados e fecho da ligação. Primeiro, o cliente envia a mensagem SYN e, junto, um número de sequência X. O servidor responde com um [SYN, ACK] e, junto, um número aleatório Y, para SYN, e o número X+1 para ACK. O cliente envia o comando ACK, confirmando a receção do segundo SYN, e o número de sequência Y+1. A comunicação fica estabelecida e começa a fase de transferência de dados. No fim da transferência, fecha-se a ligação através do envio do comando FIN-ACK para o servidor. Estes conjuntos de mensagens ACK, em paralelo com a função de *timeout*, são parte do mecanismo ARQ TCP. O método utilizado é *selective repeat*, isto é, o emissor envia vários pacotes seguidos, sem esperar pelas respostas ACK. No caso de não receber a resposta antes do tempo de *timeout*, o pacote em causa é retransmitido, escutando passivamente a nova resposta, antes do novo *timeout*, ou antes de chegar ao máximo de retransmissões possível. Estes pacotes contêm, entre outras informações, campos acerca das portas de origem e de destino, do número de sequência, do *Acknowledgement Number*, do tipo de mensagem, do tamanho da janela (Window) e do código de deteção de erros (checksum).

O mecanismo de controlo de congestão TCP assenta no número de respostas ACK recebidas pelo emissor, por unidade de tempo, calculados com o tempo de ida e volta - *Round Trip Time (RTT)*. O TCP aborda a questão com uma estratégia *multi-faced congestion-control*, baseando-se numa janela de congestão, inquirindo, inicialmente, a rede e determinando a sua capacidade (*slow start*) e, depois, detetando e recuperando falhas (*fast retransmit* e *fast recovery*). Com a janela, é possível limitar a quantidade de pacotes enviados, antes destes receberem as suas respostas, sendo uma janela dinâmica,

porque consegue aumentar se existir menos congestão (por cada RTT, aumenta em um valor - *additive increase*) e diminuir caso haja o aumento de tráfego na rede (sempre que ocorra um *timeout*, ou uma perda de pacote, é decrementada para metade, garantindo o controlo do tráfego - *multiplicative decrease*). Isto é verificado no primeiro gráfico de análise, com vários picos e vales a serem sucessivamente atingidos, num avanço gradual e global da capacidade.

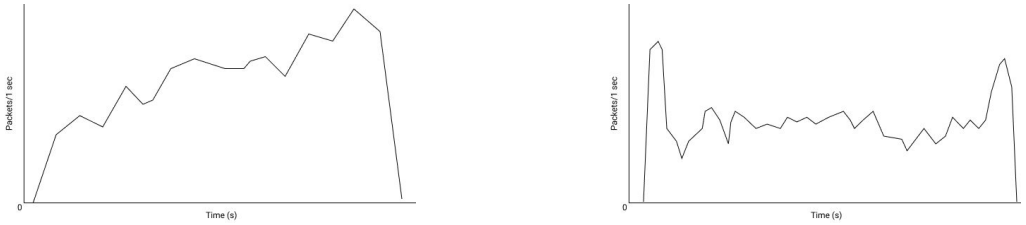


Figura 6.2 - Dois gráficos de análise da relação pacotes/s, que comprovam o mecanismo de controlo de congestão.

O segundo gráfico é relativo à experiência de colocar duas transferências em simultâneo no mesmo canal de tráfego, observando-se a mudança forçada no seu débito. Aumentando o congestionamento da rede, a taxa de transferência diminui e, eventualmente, estabiliza, de maneira a permitir ambas as operações. Nesta experiência, a transferência do TUX23 começou primeiro e sozinha, observando-se o seu *slow start*, e, momentos depois, é iniciada a transferência do TUX22 que, por sinal, acabaria primeiro e permitiria, novamente, o aumento da janela no TUX23, até esta transferência também terminar.

Conclusão

Com a concretização deste trabalho, consolidou-se, efetivamente, a aprendizagem no campo dos protocolos de transferência de informação. Conceitos como redes virtuais, endereços IP, rotas, ligações TCP, Routers, Switches, NAT, DNS, entre outros, foram absorvidos com sucesso e notámos um amadurecimento no que toca à organização do trabalho, à estruturação do código, à validação teórica das formulações e de tudo o resto que envolveu a implementação deste projeto.

Anexos

- ftp-client.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <ctype.h>
#include <netdb.h>
#include <arpa/inet.h>
#include "state.h"
#include "utils.h"

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

   if (argc != 2)
   {
       logging(LOG, "Wrong number of arguments");
       exit(1);
   }

   struct hostent *host_entity;

   logging(LOG, "FTP CLIENT - Input Data:\n-----------------------------------------\n");

   ftp_uri *data = parse_arguments(argv[1]);
   host_entity = get_ip(data->host);

   logging(NONE, "-----------------------------------------\n");

   int socket_fd;
   int socket_client_fd;
   struct sockaddr_in server_addr;
   struct sockaddr_in server_addr_client;

   // server address handling
   bzero((char *)&server_addr, sizeof(server_addr));

   server_addr.sin_family = AF_INET;
   // 32 bit Internet address network byte ordered
   server_addr.sin_addr.s_addr = inet_addr(inet_ntoa(*((struct in_addr *)host_entity->h_addr)));
   // server TCP port must be network byte ordered
   server_addr.sin_port = htons(PORT);

   // opens a TCP socket
   if ((socket_fd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
   {
       perror("socket()");
       exit(0);
   }

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

   if (server_init(socket_fd) != 0)
```

```
    {
        logging(LOG, "Server is not ready\n");
        exit(1);
    }

    res_code code;

    char user_cmd[100];
    char pass_cmd[100];
    sprintf(user_cmd, "user %s\n", data->username);
    sprintf(pass_cmd, "pass %s\n", data->password);

    logging(LOG, "Sending Username");
        if (send_command(socket_fd, user_cmd, NULL, &code) != 0 || (get_res_code(&code) != 331) &&
get_res_code(&code) != 230)
    {
        logging(LOG, "Could not set username\n");
        exit(1);
    }
    else if(get_res_code(&code) == 331)
    {
        logging(LOG, "Sending Password");
        if (send_command(socket_fd, pass_cmd, NULL, &code) != 0 || get_res_code(&code) != 230)
        {
            logging(LOG, "Could not set password\n");
            exit(1);
        }
        logging(LOG, "Password Received\n");
    }

    char pasv_cmd[] = "pasv\n";
    char *buffer = malloc(1000 * sizeof(char));

    logging(LOG, "Requesting Passive mode");
    if (send_command(socket_fd, pasv_cmd, buffer, &code) != 0 || get_res_code(&code) != 227)
    {
        logging(LOG, "Could not enter passive mode\n");
        exit(1);
    }

    logging(LOG, "In Passive mode\n");

    int port;

    if ((port = parse_connection(buffer, inet_ntoa(*((struct in_addr *)host_entity->h_addr)))) < 0)
    {
        logging(LOG, "Could not get new port in the same address\n");
        exit(1);
    }

    // server address handling
    bzero((char *)&server_addr_client, sizeof(server_addr_client));

    server_addr_client.sin_family = AF_INET;
    // 32 bit Internet address network byte ordered
    server_addr_client.sin_addr.s_addr = inet_addr(inet_ntoa(*((struct in_addr *)host_entity->h_addr)));
    // server TCP port must be network byte ordered
    server_addr_client.sin_port = htons(port);

    // opens a TCP socket
    if ((socket_client_fd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
        perror("socket()");
        exit(1);
```

```
    }

    // connects to the server
    if (connect(socket_client_fd,
                (struct sockaddr *)&server_addr_client,
                sizeof(server_addr_client)) < 0)
    {
        perror("connect()");
        exit(1);
    }

    char retr_cmd[200];
    if(strcmp(data->path, data->filename) != 0)
        sprintf(retr_cmd, "retr %s/%s\n", data->path, data->filename);
    else
        sprintf(retr_cmd, "retr %s\n", data->filename);

    logging(LOG, "Requesting Retrieve File");
    if (send_command(socket_fd, retr_cmd, NULL, &code) != 0 || get_res_code(&code) != 150)
    {
        logging(LOG, "Could not request File\n");
        exit(1);
    }
    logging(LOG, "Started Downloading File");

    if(download_file(socket_client_fd, data->filename) != 0)
    {
        logging(LOG, "Could not download File\n");
        exit(1);
    }

    logging(LOG, "Completed File Download");
    if (send_command(socket_fd, NULL, NULL, &code) != 0 || get_res_code(&code) != 226)
    {
        logging(LOG, "An error occurred. Terminating Connection\n");
    }

    logging(LOG, "Closing Connection\n");

    fflush(stdout);
    free(data);
    free(buffer);
    close(socket_fd);
    close(socket_client_fd);
    exit(0);
}
```

- utils.h

```
#ifndef UTILS_H
#define UTILS_H

#define MAXLEN 100
#define PORT 21

/**
* Logging Types
*/
typedef enum
{
   NONE,
   LOG,
   SERVER
} logtype;
```

```c
/**
* Struct to store username credentials and request info
*/
typedef struct
{
   char username[MAXLEN];
   char password[MAXLEN];
   char host[MAXLEN];
   char path[MAXLEN];
   char filename[MAXLEN];
} ftp_uri;

/**
* Struct to store server 3 digit response code
*/
typedef struct
{
   char n1;
   char n2;
   char n3;
} res_code;

/**
* Server response evaluation states
*/
typedef enum
{
   INIT,
   N1,
   N2,
   N3,
   LAST,
   STOP
} state;

/**
* Gets the server response code string as an integer
* @param res server response code struct
* @return integer representation of the response code
*/
int get_res_code(res_code *res);

/**
* printf Wrapper for logging
* @param logtype
* @param format printf first arg
* @param ...
*/
void logging(logtype logtype, const char *format, ...);

/**
* Parses the command line arguments
* @param uri ftp url syntax-like string
* @return pointer to a ftp_uri data struct
*/
ftp_uri *parse_arguments(char *uri);


/**
* Gets the host entity info
* @param host ftp host
* @return pointer to a hostent struct
*/
struct hostent *get_ip(char *host);
```

```
/**
* Gets a server response
* @param socket_fd socket file descriptor to be read
* @param code 3 digit response code
* @return 0 if success, 1 otherwise
*/
int get_response(int socket_fd, res_code *code);


/**
* Sends a command to the server and waits for a response
* @param socket_fd socket file descriptor to be read
* @param cmd command to be sent to the server
* @param buffer to fill with the entire server response string
* @param code 3 digit response code
* @return 0 if success, 1 otherwise
*/
int send_command(int socket_fd, char *cmd, char *buffer, res_code *code);

/**
* Parses the buffer string with the address of the new connection
* for downloading the file
* @param buffer string containing an address of the form (ip1,ip2,ip3,ip4,p1,p2)
* @param ip_address original ip address to compare with the response
* @return new port number if success, -1 otherwise
*/
int parse_connection(char *buffer, char *ip_address);

/**
* Reads the server welcome message
* @param socket_fd socket file descriptor to be read
* @return 0 if successful, 1 otherwise
*/
int server_init(int socket_fd);

/**
* Downloads the file from socket_client_fd
* @param socket_client_fd stream transmitting the file
* @param filename name of the file to be downloaded
* @return 0 if successful
*/
int download_file(int socket_client_fd, char *filename);

#endif
```

- utils.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <stdarg.h>
#include <fcntl.h>
#include <netdb.h>
#include <arpa/inet.h>
#include "state.h"
#include "utils.h"

int get_res_code(res_code *res)
{
   char scode[3] = {res->n1, res->n2, res->n3};
   return atoi(scode);
```

```
}

void logging(logtype logtype, const char *format, ...)
{
   char *logs[] = {"", "LOG", "SVR"};
   va_list args;

   va_start(args, format);

   if (logtype != NONE)
       printf("\n[%s]\t", logs[logtype]); // prints a log prefix

   vprintf(format, args); // calls printf with format + args

   va_end(args);
   fflush(stdout); // flushes out the printed stuff
}

ftp_uri *parse_arguments(char *uri)
{

   ftp_uri *data = malloc(sizeof(ftp_uri));
   memset(data, 0, sizeof(ftp_uri));

   // parses the args uri
   int res = sscanf(uri, "ftp://%[^:]:%[^@]@%[^/]/%s",
                    data->username,
                    data->password,
                    data->host,
                    data->path);

   if (res != 4) // assumes username and password were not provided and tries again
   {

       strcpy(data->username, "anonymous"); // default username value if not provided
       strcpy(data->password, "password");  // default password value if not provided

       res = sscanf(uri, "ftp://%[^/]/%s",
                    data->host,
                    data->path);

       if (res != 2) // if another error occurred then finally aborts
       {
           logging(LOG, "Wrong arguments format\n");
           exit(1);
       }
   }

   int n = strlen(data->path);
   char *suffix = data->path + n; // parses the filename as a suffix of the path

   while (0 < n && data->path[--n] != '/')
       ;

   if (data->path[n] == '/')
   {
       suffix = data->path + n + 1;
       data->path[n] = '\0';
   }

   if (strlen(suffix) > 0)
       strcpy(data->filename, suffix);
   else
       strcpy(data->filename, data->path);
```

```
    logging(NONE, "\tUsername: %s\n\tPassword: %s\n\tHost: %s\n\tPath: %s\n\tFilename: %s\n",
            data->username,
            data->password,
            data->host,
            data->path,
            data->filename);

    return data;
}

struct hostent *get_ip(char *host)
{
    struct hostent *host_entity = malloc(sizeof(struct hostent));
    memset(host_entity, 0, sizeof(struct hostent));

    if ((host_entity = gethostbyname(host)) == NULL) // gets host ip address by its name
    {
        herror("Error calling gethostbyname function");
        exit(1);
    }

    logging(NONE, "\tHost Ip Address: %s\n",
            inet_ntoa(
                *(
                    (struct in_addr *)
                        host_entity->h_addr)));

    return host_entity;
}

int get_response(int socket_fd, res_code *code)
{
    char c;
    state st = INIT;

    printf("\n");

    while (st != STOP)
    {
        if (read(socket_fd, &c, 1) != 1) // reads char by char the server response
        {
            logging(LOG, "Error reading response from ftp server\n");
            return 1;
        }

        printf("%c", c);

        st = getState(c, st, code); // evaluates the server response and returns a new state
    }

    printf("\n");

    return 0;
}

int send_command(int socket_fd, char *cmd, char *buffer, res_code *code)
{
    int res = 0;

    do
    {
        if (cmd != NULL)
            write(socket_fd, cmd, strlen(cmd)); // sends a command to the server
```

```
        logging(SERVER, "Server Response:");

        if (buffer != NULL) // if buffer is not NULL, it will be filled with the server response
        {
            freopen("/dev/null", "a", stdout);
            setbuf(stdout, buffer); // sends stdout data to buffer
        }

        res = get_response(socket_fd, code);

        if (buffer != NULL)
        {
            freopen("/dev/tty", "a", stdout); // restores stdout
            printf("%s", buffer);
        }
    } while (code->n1 == '4'); // server code to ask for resending command

    if (code->n1 == '5') // server code to abort operation
        return 1;

    return res;
}

int parse_connection(char *buffer, char *ip_address)
{
    char dummy[100];
    char ip1[10], ip2[10], ip3[10], ip4[10];
    char p1[10], p2[10];
    char ip_addr[50];
    int port1, port2;

    int res = sscanf(buffer, "%[^(](%[^,],%[^,],%[^,],%[^,],%[^)])", dummy, ip1, ip2, ip3, ip4, p1, p2);

    sprintf(ip_addr, "%s.%s.%s.%s", ip1, ip2, ip3, ip4);

    if (res != 7 || strcmp(ip_addr, ip_address) != 0)
        return -1;
    else
        return atoi(p1) * 256 + atoi(p2); // calculates the new port number
}

int server_init(int socket_fd)
{
    logging(SERVER, "Server Welcome Response:\n");

    res_code code;
    if (get_response(socket_fd, &code) != 0 && get_res_code(&code) != 220)
        return 1;

    logging(LOG, "Server is ready\n");
    return 0;
}

int download_file(int socket_client_fd, char *filename)
{

    int fd = open(filename, O_CREAT | O_RDWR, 0777);

    char buffer[1000];
    int bytes;
    while ((bytes = read(socket_client_fd, buffer, 1000)) > 0)
    {
        bytes = write(fd, buffer, bytes);
```

```
    }

    close(fd);
    return 0;
}
```

- state.h

```
#include <stdlib.h>
#include "utils.h"

#ifndef STATE_H
#define STATE_H

/**
* Server Response State Machine - Init State
* @param input to be evaluated
* @param res server response code struct
* @return new state after evaluation
*/
state init_state(char input, res_code * res);

/**
* Server Response State Machine - N1 State
* @param input to be evaluated
* @param res server response code struct
* @return new state after evaluation
*/
state n1_state(char input, res_code * res);

/**
* Server Response State Machine - N2 State
* @param input to be evaluated
* @param res server response code struct
* @return new state after evaluation
*/
state n2_state(char input, res_code * res);

/**
* Server Response State Machine - N3 State
* @param input to be evaluated
* @return new state after evaluation
*/
state n3_state(char input);

/**
* Server Response State Machine - Last State
* @param input to be evaluated
* @return new state after evaluation
*/
state last_state(char input);

/**
* Server Response State Machine - Main Switch
* @param input to be evaluated
* @param current_state to call the corresponding function
* @param res server response code struct
* @return new state after evaluation
*/
state getState(char input, state current_state, res_code * res);

#endif
```

- state.c

```
#include <ctype.h>
#include "state.h"

state init_state(char input, res_code *res)
{
   if (isdigit(input))
   {
       if (res != NULL)
           res->n1 = input;
       return N1;
   }
   else
   {
       return INIT;
   }
}

state n1_state(char input, res_code *res)
{
   if (isdigit(input))
   {
       if (res != NULL)
           res->n2 = input;
       return N2;
   }
   else
   {
       return INIT;
   }
}

state n2_state(char input, res_code *res)
{
   if (isdigit(input))
   {
       if (res != NULL)
           res->n3 = input;
       return N3;
   }
   else
   {
       return INIT;
   }
}

state n3_state(char input)
{
   if (input == ' ')
   {
       return LAST;
   }
   else
   {
       return INIT;
   }
}

state last_state(char input)
{
   if (input == '\n' || input == '\r')
   {
```

```
        return STOP;
    }
    else
    {
        return LAST;
    }
}

state getState(char input, state current_state, res_code *res)
{
    switch (current_state)
    {
    case INIT:
        return init_state(input, res);
    case N1:
        return n1_state(input, res);
    case N2:
        return n2_state(input, res);
    case N3:
        return n3_state(input);
    case LAST:
        return last_state(input);
    case STOP:
        return STOP;
    default:
        return INIT;
    }
}
```

- test.sh

```
#!/bin/bash

if [ $1 = 1 ]
then
    ./download ftp://netlab1.fe.up.pt/pub.txt
elif [ $1 = 2 ]
then
    ./download ftp://rcom:rcom@netlab1.fe.up.pt/pipe.txt
elif [ $1 = 3 ]
then
    ./download ftp://ftp.up.pt/pub/parrot/last-sync.stamp
elif [ $1 = 4 ]
then
    ./download ftp://ftp.up.pt/pub/parrot/README.html
elif [ $1 = 5 ]
then
    ./download ftp://ftp.up.pt/pub/parrot/keyring.gpg
else
    ./download ftp://ftp.up.pt/pub/parrot/keyring.gpg
    ./download ftp://ftp.up.pt/pub/parrot/README.html
    ./download ftp://ftp.up.pt/pub/parrot/last-sync.stamp
    ./download ftp://rcom:rcom@netlab1.fe.up.pt/pipe.txt
    ./download ftp://netlab1.fe.up.pt/pub.txt
fi
```

- tux2.sh

```
#!/bin/bash

# Updateimage

echo -n "Restarting networking service... "
service networking restart
echo "done!"

ifconfig eth0 up 172.16.21.1/24
echo "IP Address set."

# EXP 3

route add default gw 172.16.21.254
echo "Default gateway route set."

route add -net 172.16.20.0/24 gw 172.16.21.253
echo "Route to VLAN 20 set."

printf "search netlab1.fe.up.pt\nnameserver 172.16.1.1\n" > /etc/resolv.conf
echo "DNS set."
```

- tux3.sh

```
#!/bin/bash

# Updateimage

echo -n "Restarting networking service... "
service networking restart
echo "done!"

ifconfig eth0 up 172.16.20.1/24
echo "IP Address set."

# EXP 3 ...

route add default gw 172.16.20.254
echo "Default gateway route set."

route add -net 172.16.21.0/24 gw 172.16.20.254
echo "Route to VLAN 21 set."

printf "search netlab1.fe.up.pt\nnameserver 172.16.1.1\n" > /etc/resolv.conf
echo "DNS set."
```

- tux4.sh

```
#!/bin/bash

# Updateimage

echo -n "Restarting networking service... "
service networking restart
echo "done!"

ifconfig eth0 up 172.16.20.254/24

# EXP 3 ...
```

```
ifconfig eth1 up 172.16.21.253/24
echo "IP Addresses set."

route add default gw 172.16.21.254
echo "Default gateway route set."

echo 1 > /proc/sys/net/ipv4/ip_forward
echo 0 > /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts

echo "Linux router functions activated."

printf "search netlab1.fe.up.pt\nnameserver 172.16.1.1\n" > /etc/resolv.conf
echo "DNS set."
```

- switch.sh

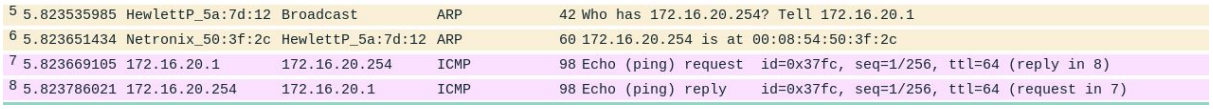
```
#!/bin/bash

# reiniciair switch

enable
password:8nortel
configure terminal
no vlan 2-4094
exit
copy flash:tux2-clean startup-config
reload

# CONFIGURAR SWITCH

# EXP 2 ...

conf t
vlan 20
end

conf t
vlan 21
end

# TUX 3 ETH0

conf t
interface fastethernet 0/1
switchport mode access
switchport access vlan 20
end

# TUX 4 ETH0

conf t
interface fastethernet 0/3
switchport mode access
switchport access vlan 20
end

# TUX 2 ETH0

conf t
interface fastethernet 0/2
switchport mode access
switchport access vlan 21
end
```

```
# EXP 3

# TUX 4 ETH1

conf t
interface fastethernet 0/8
switchport mode access
switchport access vlan 21
end

# ROUTER

conf t
interface fastethernet 0/9
switchport mode access
switchport access vlan 21
end

show vlan brief

copy running-config flash:edupedro-T1B2

# reiniciair switch mas com nossas configurações

enable
password:8nortel
configure terminal
no vlan 2-4094
exit
copy flash:edupedro-T1B2 startup-config
reload
```

- router.sh

```
conf t
interface gigabitethernet 0/0
ip address 172.16.21.254 255.255.255.0
no shutdown
ip nat inside
exit

interface gigabitethernet 0/1
ip address 172.16.1.29 255.255.255.0
no shutdown
ip nat outside
exit

ip nat pool ovrld 172.16.1.29 172.16.1.29 prefix 24
ip nat inside source list 1 pool ovrld 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 0.0.0.0 172.16.1.254
ip route 172.16.20.0 255.255.255.0 172.16.21.253
end
```

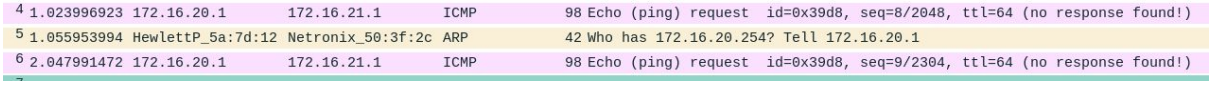
- Experiência 1



5	5.823535985	HewlettP_5a:7d:12	Broadcast	ARP	42	Who has 172.16.20.254? Tell 172.16.20.1
6	5.823651434	Netronix_50:3f:2c	HewlettP_5a:7d:12	ARP	60	172.16.20.254 is at 00:08:54:50:3f:2c
7	5.823669105	172.16.20.1	172.16.20.254	ICMP	98	Echo (ping) request id=0x37fc, seq=1/256, ttl=64 (reply in 8)
8	5.823786021	172.16.20.254	172.16.20.1	ICMP	98	Echo (ping) reply id=0x37fc, seq=1/256, ttl=64 (request in 7)

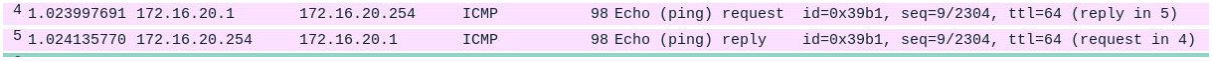
Figura 1.1 - No TUX23, *ping* ao TUX24 - primeiro, envia pacotes ARP na tentativa de resolver o endereço IP e depois passa a enviar pacotes ICMP.

- Experiência 2



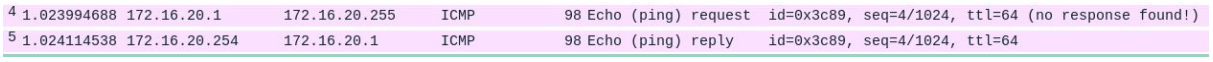
4	1.023996923	172.16.20.1	172.16.21.1	ICMP	98	Echo (ping) request id=0x39d8, seq=8/2048, ttl=64 (no response found!)
5	1.055953994	HewlettP_5a:7d:12	Netronix_50:3f:2c	ARP	42	Who has 172.16.20.254? Tell 172.16.20.1
6	2.047991472	172.16.20.1	172.16.21.1	ICMP	98	Echo (ping) request id=0x39d8, seq=9/2304, ttl=64 (no response found!)

Figura 2.1 - No TUX23, *ping* ao TUX22 - não consegue obter resposta, porque ainda não existe ligação entre os dois.



4	1.023997691	172.16.20.1	172.16.20.254	ICMP	98	Echo (ping) request id=0x39b1, seq=9/2304, ttl=64 (reply in 5)
5	1.024135770	172.16.20.254	172.16.20.1	ICMP	98	Echo (ping) reply id=0x39b1, seq=9/2304, ttl=64 (request in 4)

Figura 2.2 - No TUX23, *ping* ao TUX24.



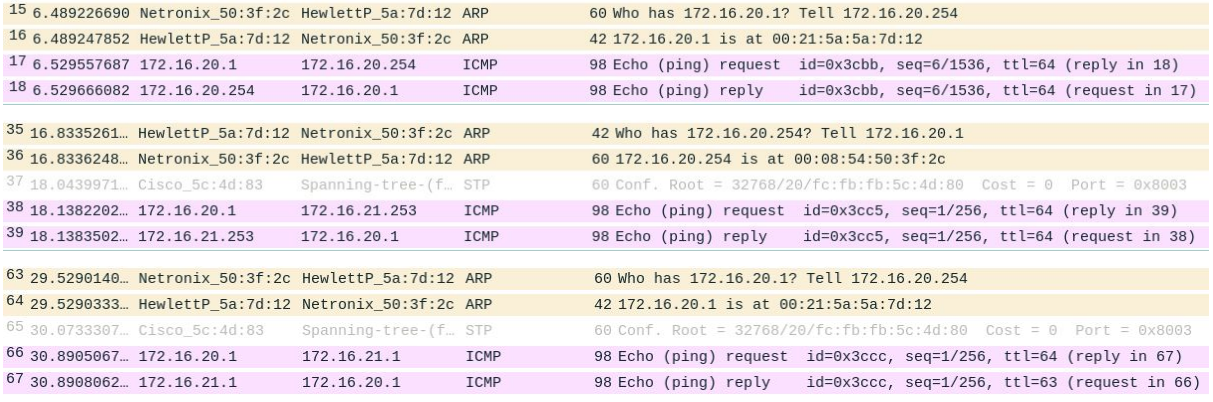
4	1.023994688	172.16.20.1	172.16.20.255	ICMP	98	Echo (ping) request id=0x3c89, seq=4/1024, ttl=64 (no response found!)
5	1.024114538	172.16.20.254	172.16.20.1	ICMP	98	Echo (ping) reply id=0x3c89, seq=4/1024, ttl=64

Figura 2.3 - No TUX23, *ping broadcast* - apenas obtém resposta do TUX24, o único na sua subrede.

4	1.023998333	172.16.21.1	172.16.21.255	ICMP	98	Echo (ping) request id=0x3420, seq=5/1280, ttl=64 (no response found!)

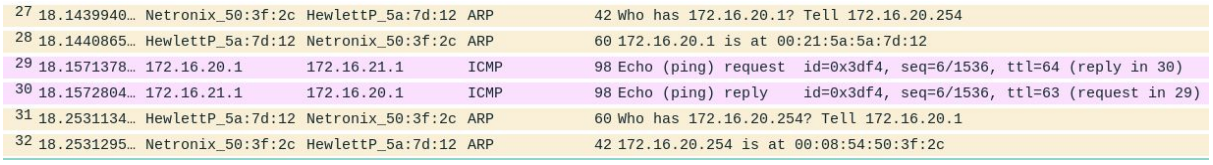
Figura 2.4 - No TUX22, *ping broadcast* - não obtém nenhuma resposta, porque não há mais dispositivos na sua subrede.

- Experiência 3



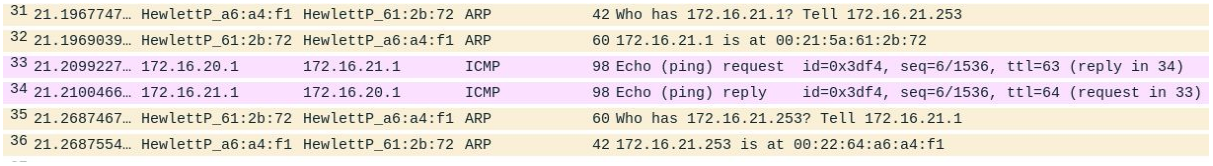
15	6.489226690	Netronix_50:3f:2c	HewlettP_5a:7d:12	ARP	60	Who has 172.16.20.1? Tell 172.16.20.254
16	6.489247852	HewlettP_5a:7d:12	Netronix_50:3f:2c	ARP	42	172.16.20.1 is at 00:21:5a:5a:7d:12
17	6.529557687	172.16.20.1	172.16.20.254	ICMP	98	Echo (ping) request id=0x3cbb, seq=6/1536, ttl=64 (reply in 18)
18	6.529666082	172.16.20.254	172.16.20.1	ICMP	98	Echo (ping) reply id=0x3cbb, seq=6/1536, ttl=64 (request in 17)
35	16.8335261…	HewlettP_5a:7d:12	Netronix_50:3f:2c	ARP	42	Who has 172.16.20.254? Tell 172.16.20.1
36	16.8336248…	Netronix_50:3f:2c	HewlettP_5a:7d:12	ARP	60	172.16.20.254 is at 00:08:54:50:3f:2c
37	18.0439971…	Cisco_5c:4d:83	Spanning-tree-(f…	STP	60	Conf. Root = 32768/20/fc:fb:fb:5c:4d:80 Cost = 0 Port = 0x8003
38	18.1382202…	172.16.20.1	172.16.21.253	ICMP	98	Echo (ping) request id=0x3cc5, seq=1/256, ttl=64 (reply in 39)
39	18.1383502…	172.16.21.253	172.16.20.1	ICMP	98	Echo (ping) reply id=0x3cc5, seq=1/256, ttl=64 (request in 38)
63	29.5290140…	Netronix_50:3f:2c	HewlettP_5a:7d:12	ARP	60	Who has 172.16.20.1? Tell 172.16.20.254
64	29.5290333…	HewlettP_5a:7d:12	Netronix_50:3f:2c	ARP	42	172.16.20.1 is at 00:21:5a:5a:7d:12
65	30.0733307…	Cisco_5c:4d:83	Spanning-tree-(f…	STP	60	Conf. Root = 32768/20/fc:fb:fb:5c:4d:80 Cost = 0 Port = 0x8003
66	30.8905067…	172.16.20.1	172.16.21.1	ICMP	98	Echo (ping) request id=0x3ccc, seq=1/256, ttl=64 (reply in 67)
67	30.8908062…	172.16.21.1	172.16.20.1	ICMP	98	Echo (ping) reply id=0x3ccc, seq=1/256, ttl=63 (request in 66)

Figura 3.1 - No TUX23, *ping* às restantes interfaces, tendo apagado as entradas da tabela ARP.



27	18.1439940…	Netronix_50:3f:2c	HewlettP_5a:7d:12	ARP	42	Who has 172.16.20.1? Tell 172.16.20.254
28	18.1440865…	HewlettP_5a:7d:12	Netronix_50:3f:2c	ARP	60	172.16.20.1 is at 00:21:5a:5a:7d:12
29	18.1571378…	172.16.20.1	172.16.21.1	ICMP	98	Echo (ping) request id=0x3df4, seq=6/1536, ttl=64 (reply in 30)
30	18.1572804…	172.16.21.1	172.16.20.1	ICMP	98	Echo (ping) reply id=0x3df4, seq=6/1536, ttl=63 (request in 29)
31	18.2531134…	HewlettP_5a:7d:12	Netronix_50:3f:2c	ARP	60	Who has 172.16.20.254? Tell 172.16.20.1
32	18.2531295…	Netronix_50:3f:2c	HewlettP_5a:7d:12	ARP	42	172.16.20.254 is at 00:08:54:50:3f:2c

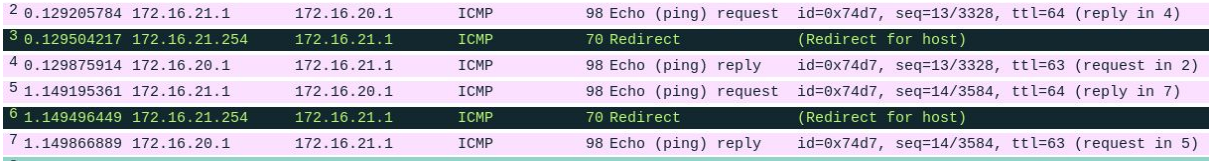
Figura 3.2 - Na interface ETH0 do TUX24, TUX23 *ping* ao TUX22, tendo apagado as entradas da tabela ARP.



31	21.1967747…	HewlettP_a6:a4:f1	HewlettP_61:2b:72	ARP	42	Who has 172.16.21.1? Tell 172.16.21.253
32	21.1969039…	HewlettP_61:2b:72	HewlettP_a6:a4:f1	ARP	60	172.16.21.1 is at 00:21:5a:61:2b:72
33	21.2099227…	172.16.20.1	172.16.21.1	ICMP	98	Echo (ping) request id=0x3df4, seq=6/1536, ttl=63 (reply in 34)
34	21.2100466…	172.16.21.1	172.16.20.1	ICMP	98	Echo (ping) reply id=0x3df4, seq=6/1536, ttl=64 (request in 33)
35	21.2687467…	HewlettP_61:2b:72	HewlettP_a6:a4:f1	ARP	60	Who has 172.16.21.253? Tell 172.16.21.1
36	21.2687554…	HewlettP_a6:a4:f1	HewlettP_61:2b:72	ARP	42	172.16.21.253 is at 00:22:64:a6:a4:f1

Figura 3.3 - Na interface ETH1 do TUX24, TUX23 *ping* ao TUX22, tendo apagado as entradas da tabela ARP.

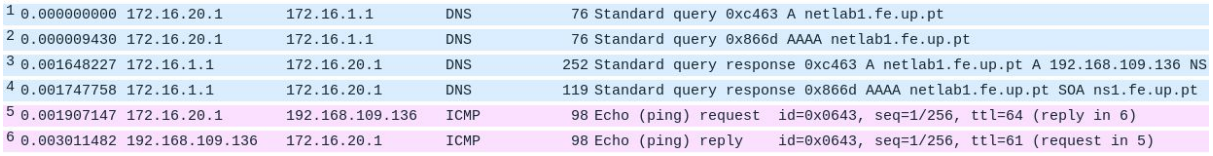
- Experiência 4



2	0.129205784	172.16.21.1	172.16.20.1	ICMP	98	Echo (ping) request id=0x74d7, seq=13/3328, ttl=64 (reply in 4)
3	0.129504217	172.16.21.254	172.16.21.1	ICMP	70	Redirect (Redirect for host)
4	0.129875914	172.16.20.1	172.16.21.1	ICMP	98	Echo (ping) reply id=0x74d7, seq=13/3328, ttl=63 (request in 2)
5	1.149195361	172.16.21.1	172.16.20.1	ICMP	98	Echo (ping) request id=0x74d7, seq=14/3584, ttl=64 (reply in 7)
6	1.149496449	172.16.21.254	172.16.21.1	ICMP	70	Redirect (Redirect for host)
7	1.149866889	172.16.20.1	172.16.21.1	ICMP	98	Echo (ping) reply id=0x74d7, seq=14/3584, ttl=63 (request in 5)

Figura 4.1 - No TUX22, sem rota e sem redirecionamento, ping ao TUX23, passando pelo Router.

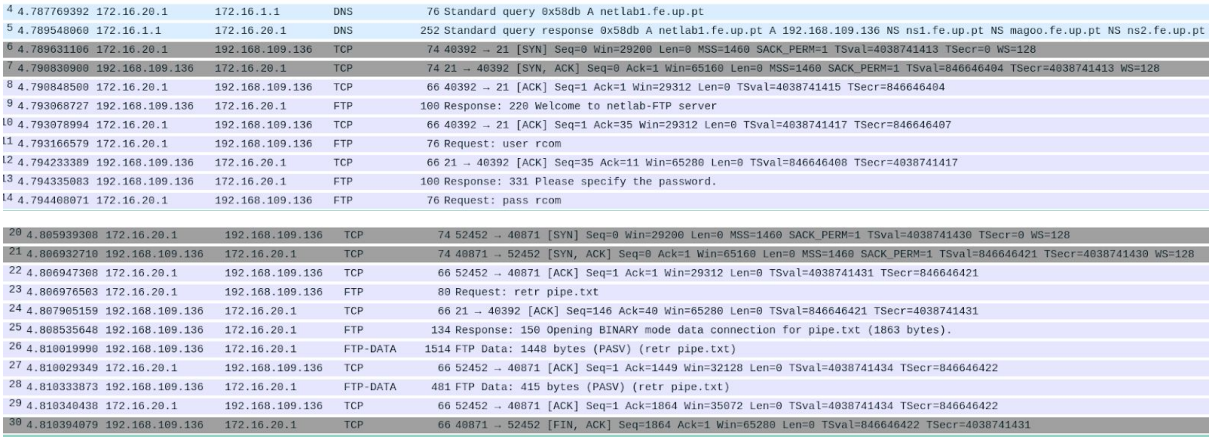
- Experiência 5



1	0.000000000	172.16.20.1	172.16.1.1	DNS	76	Standard query 0xc463 A netlab1.fe.up.pt
2	0.000009430	172.16.20.1	172.16.1.1	DNS	76	Standard query 0x866d AAAA netlab1.fe.up.pt
3	0.001648227	172.16.1.1	172.16.20.1	DNS	252	Standard query response 0xc463 A netlab1.fe.up.pt A 192.168.109.136 NS
4	0.001747758	172.16.1.1	172.16.20.1	DNS	119	Standard query response 0x866d AAAA netlab1.fe.up.pt SOA ns1.fe.up.pt
5	0.001907147	172.16.20.1	192.168.109.136	ICMP	98	Echo (ping) request id=0x0643, seq=1/256, ttl=64 (reply in 6)
6	0.003011482	192.168.109.136	172.16.20.1	ICMP	98	Echo (ping) reply id=0x0643, seq=1/256, ttl=61 (request in 5)

Figura 5.1 - No TUX23, ping ao servidor FTP netlab1.fe.up.pt.

- Experiência 6



4	4.787769392	172.16.20.1	172.16.1.1	DNS	76	Standard query 0x58db A netlab1.fe.up.pt
5	4.789548060	172.16.1.1	172.16.20.1	DNS	252	Standard query response 0x58db A netlab1.fe.up.pt A 192.168.109.136 NS ns1.fe.up.pt NS magoo.fe.up.pt NS ns2.fe.up.pt
6	4.789631106	172.16.20.1	192.168.109.136	TCP	74	40392 → 21 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=4038741413 TSecr=0 WS=128
7	4.790830900	192.168.109.136	172.16.20.1	TCP	74	21 → 40392 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 TSval=846646404 TSecr=4038741413 WS=128
8	4.790848500	172.16.20.1	192.168.109.136	TCP	66	40392 → 21 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=4038741415 TSecr=846646404
9	4.793068727	192.168.109.136	172.16.20.1	FTP	100	Response: 220 Welcome to netlab-FTP server
10	4.793078994	172.16.20.1	192.168.109.136	TCP	66	40392 → 21 [ACK] Seq=1 Ack=35 Win=29312 Len=0 TSval=4038741417 TSecr=846646407
11	4.793166579	172.16.20.1	192.168.109.136	FTP	76	Request: user rcom
12	4.794233389	192.168.109.136	172.16.20.1	TCP	66	21 → 40392 [ACK] Seq=35 Ack=11 Win=65280 Len=0 TSval=846646408 TSecr=4038741417
13	4.794335083	192.168.109.136	172.16.20.1	FTP	100	Response: 331 Please specify the password.
14	4.794408071	172.16.20.1	192.168.109.136	FTP	76	Request: pass rcom
20	4.805939308	172.16.20.1	192.168.109.136	TCP	74	52452 → 40871 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=4038741430 TSecr=0 WS=128
21	4.806932710	192.168.109.136	172.16.20.1	TCP	74	40871 → 52452 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 TSval=846646421 TSecr=4038741430 WS=128
22	4.806947308	172.16.20.1	192.168.109.136	TCP	66	52452 → 40871 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=4038741431 TSecr=846646421
23	4.806976503	172.16.20.1	192.168.109.136	FTP	80	Request: retr pipe.txt
24	4.807905159	192.168.109.136	172.16.20.1	TCP	66	21 → 40392 [ACK] Seq=146 Ack=40 Win=65280 Len=0 TSval=846646421 TSecr=4038741431
25	4.808535648	192.168.109.136	172.16.20.1	FTP	134	Response: 150 Opening BINARY mode data connection for pipe.txt (1863 bytes).
26	4.810019990	192.168.109.136	172.16.20.1	FTP-DATA	1514	FTP Data: 1448 bytes (PASV) (retr pipe.txt)
27	4.810029349	172.16.20.1	192.168.109.136	TCP	66	52452 → 40871 [ACK] Seq=1 Ack=1449 Win=32128 Len=0 TSval=4038741434 TSecr=846646422
28	4.810333873	192.168.109.136	172.16.20.1	FTP-DATA	481	FTP Data: 415 bytes (PASV) (retr pipe.txt)
29	4.810340438	172.16.20.1	192.168.109.136	TCP	66	52452 → 40871 [ACK] Seq=1 Ack=1864 Win=35072 Len=0 TSval=4038741434 TSecr=846646422
30	4.810394079	192.168.109.136	172.16.20.1	TCP	66	40871 → 52452 [FIN, ACK] Seq=1864 Ack=1 Win=65280 Len=0 TSval=846646422 TSecr=4038741431

Figura 6.1 - No TUX23, abertura e fecho das duas conexões TCP, ao longo da transferência de um ficheiro através do servidor FTP.