

Redes de Computadores

Relatório do 2º Trabalho Laboratorial
Desenvolvimento de uma aplicação de download e
Configuração de uma Rede

Mestrado Integrado em Engenharia Informática e Computação (MIEIC)

Turma 5

23 de Dezembro de 2019

Andreia Gouveia up201706430@fe.up.pt

Cláudia Martins up201704136@fe.up.pt

Filipa Senra up201704077@fe.up.pt

Índice

Introdução	3
Parte 1 – Aplicação de download.....	4
Parte 2 – Configuração da rede e análise	6
Experiência 1 – Configurar um IP de rede.....	6
Experiência 2 – Implementar duas LANs virtuais no switch.....	8
Experiência 3 – Configurar um router em Linux	9
Experiência 4.....	10
Experiência 5.....	12
Experiência 6.....	12
Conclusão.....	15
Anexos.....	16

Introdução

O segundo trabalho laboratorial da unidade curricular de Redes de Computadores (RCOM), do primeiro semestre do terceiro ano do Mestrado Integrado em Engenharia Informática e Computação (MIEIC), teve como objetivo a elaboração de uma aplicação para *download de ficheiros* e a configuração de uma rede de computadores.

Para a aplicação de *download*, desenvolvida em linguagem C e em ambiente Linux, o objetivo era desenvolver um cliente FTP capaz de comunicar com um servidor remoto usando *sockets* TCP para descarregar um ficheiro da Internet, identificado por um *Uniform Resource Locator* (URL).

A configuração da rede de computadores teve como principal objetivo correr a aplicação de *download* num dos nós da rede. No guião, esta parte do trabalho foi dividida em diversas experiências que detalham a construção de redes progressivamente mais complexas que envolvem o uso de um *switch* ligado a vários computadores (com DNS configurado) e a um *router*, o qual tinha NAT configurada e estava ligado à Internet. Estas experiências pretendem complementar diversos tópicos abordados nas aulas teóricas sobre as camadas mais altas do modelo TCP/IP (camadas de rede, de transporte e de aplicação).

O objetivo deste relatório é apresentar, de forma sucinta, os passos e as decisões tomadas para o desenvolvimento da aplicação e da rede de computadores.

O seguimento deste relatório subdivide-se nas seguintes partes:

1. **Desenvolvimento da aplicação de *download***, onde será apresentada a arquitetura do programa desenvolvido e o resultado da sua execução.
2. **Configuração da rede e a sua análise**, onde são descritos, para cada experiência realizada, a arquitetura da rede, os objetivos de cada experiência, comandos de configuração e análise dos registos gravados durante a sua realização;
3. **Conclusões**, onde são redigidas as últimas análises e opinião final do grupo ao projeto;

Parte 1 – Aplicação de download

Na presente secção, será descrita a aplicação de cliente FTP desenvolvida para descarregar ficheiros de servidores remotos. A Aplicação de cliente FTP foi desenvolvida de acordo com o RFC959 que aborda o protocolo de transferência de ficheiro (FTP) e o RFC1738 que aborda o uso de *URLs* e o seu tratamento.

A compilação do programa poderá ser efetuada através do comando *make*. A aplicação poderá ser executada escrevendo na linha de comandos “./download ftp://[<user>:<password>@]<host>/<url-path>”.

O *port* é, por omissão, 21.

1. Arquitetura

A aplicação foi desenvolvida com duas camadas: a do processamento do URL e a do cliente FTP.

A camada do processamento do URL (*url.h* e *url.c*) possui a estrutura URL que contém diversos membros que caracterizam a ligação a efetuar posteriormente: *user*, *password*, *host*, *ip*, *path*, *filename*, *port*. Esta camada possui três funções. De seguida, apresentamos uma breve descrição destas:

1. **initURL**: inicializa os argumentos da estrutura URL.
2. **parseURL**: faz o processamento do URL.
3. **getIpHost**: obtém o IP do Host.

A camada referente ao cliente FTP (*server.h* e *server.c*) possui a estrutura SERVER que contém o descritor de ficheiro do *socket* de controlo e o descritor de ficheiro do *socket* de dados. Esta camada possui várias funções. De seguida, apresentamos uma breve descrição das funções mais pertinentes:

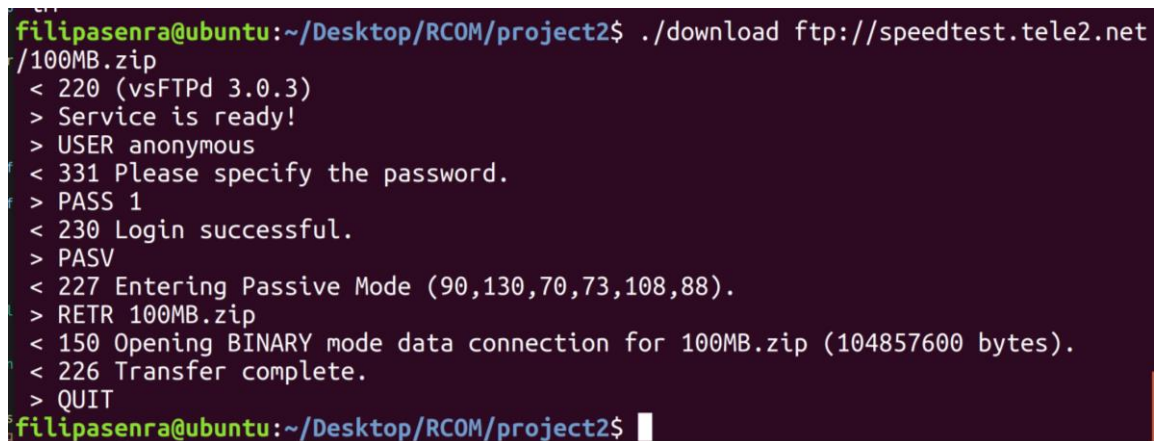
1. **connectToServer**: responsável pela abertura da ligação de controlo com o servidor (utiliza a função auxiliar *clientTCP*).
2. **loginServer**: responsável pelo login do utilizador no servidor.
3. **psvModeServer**: responsável por entrar no modo passivo com o servidor e pela criação da *socket* para a transferência de dados.
4. **retrServer**: responsável pela solicitação do download do ficheiro.
5. **downloadFromServer**: responsável pela captura dos dados na *socket* de dados e pela criação do ficheiro transferido.
6. **disconnectToServer**: responsável pelo fecho da ligação de dados com o servidor.

NOTA: todas estas funções utilizam as funções auxiliares `sendToServer` e `readFromServer`.

2. Resultados de download

Para demonstrar o bom funcionamento da aplicação desenvolvida, foram realizadas várias experiências de download de diferentes ficheiros de diversos servidores.

Na figura abaixo, é apresentado o output da aplicação para a consola após ter sido executado o comando `./download ftp://speedtest.tele2.net/100MB.zip`.



```
filipasenra@ubuntu:~/Desktop/RCOM/project2$ ./download ftp://speedtest.tele2.net/100MB.zip
< 220 (vsFTPd 3.0.3)
> Service is ready!
> USER anonymous
< 331 Please specify the password.
> PASS 1
< 230 Login successful.
> PASV
< 227 Entering Passive Mode (90,130,70,73,108,88).
> RETR 100MB.zip
< 150 Opening BINARY mode data connection for 100MB.zip (104857600 bytes).
< 226 Transfer complete.
> QUIT
filipasenra@ubuntu:~/Desktop/RCOM/project2$
```

Figura 1 – Output da aplicação com o comando `./download ftp://speedtest.tele2.net/100MB.zip`.

Todos os testes foram concluídos com sucesso.

Parte 2 – Configuração da rede e análise

Experiência 1 – Configurar um IP de rede

A experiência 1 teve como objetivo associar endereços IP a interfaces de computadores, para que estes possam comunicar entre si. Nesta experiência, a rede é composta pelo *tux1* e *tux2* ligados a um *switch*.

1. Que são os pacotes de ARP e para que são usados?

O ARP é um protocolo de pergunta e resposta utilizado para mapear dinamicamente endereços entre duas camadas distintas. No caso desta experiência, são usados para obter o endereço MAC associado a um dado endereço IP. Por outras palavras, os pacotes de ARP mapeia, o endereço da rede a um endereço físico.

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

Nesta experiência, fizemos *ping* do *tux1* para o *tux4*.

O *tux4* (**tux** a ser pingado) envia um pacote ARP com o intuito de identificar o endereço MAC da entidade que lhe está a tentar enviar algo. O pacote ARP é composto por duas partes importantes: o endereço IP e MAC do emissor e o endereço IP e MAC do recetor. O pacote enviado pelo *tux4* possui como endereços IP e MAC emissores os endereços de si (172.16.20.254 e 00:08:54:50:3f:2c respetivamente) do *tux4*. O pacote possui como endereços IP e MAC do receptor o endereço IP do *tux1* e o valor default do MAC (172.16.20.1 e 00:00:00:00:00:00 respetivamente). O endereço MAC encontra-se com o valor *default*, pois este pacote tem como o intuito decodificar o endereço MAC do *tux* de IP recetor (neste caso, decodificar o endereço MAC do *tux1*). Estes valores podem ser conferidos no anexo A.

O *tux1* irá responder com outro pacote ARP a confirmar que é o *tux* que possui o endereço IP procurado, enviando o seu endereço MAC (00:c0:df:08:d5:b2). Estes valores podem ser conferidos no anexo B.

3. Que pacotes são gerados pelo comando ping?

O comando *ping* gerou dois tipos de pacotes: ARP e ICMP. Os pacotes ARP têm como objetivo a obtenção do endereço MAC do *tux* que gera o *ping*. Os pacotes ICMP

(*Internet Control Message Protocol*) testam a conectividade entre os dois *tuxs* (o *tux* que gera o *ping* e o *tux* pingado).

4. Quais são os endereços MAC e IP dos pacotes de ping?

Ao invocar o comando *ping*, vamos gerar pacotes com dois tipos de pares de endereços MAC e IP:

Pacote de pedido (anexo C) – emissor Tux1 e recetor Tux4:

1. Endereço MAC do emissor: 172.16.20.1
2. Endereço IP do emissor: 00:c0:df:08:d5:b2
3. Endereço MAC do recetor: 00:08:54:50:3f:2c
4. Endereço IP do recetor: 172.16.20.254

Pacote de resposta (anexo D) – emissor Tux4 e recetor Tux1:

1. Endereço MAC do emissor: 00:08:54:50:3f:2c
2. Endereço IP do emissor: 172.16.20.254
3. Endereço MAC do recetor: 00:c0:df:08:d5:b2
4. Endereço IP do recetor: 172.16.20.1

5. Como é que se determina se uma trama recetora Ethernet é ARP, IP, ICMP?

Para determinar o tipo de trama, inspecionamos o *Ethernet Header* do pacote. Caso o campo possuía o valor:

1. 0x0800: a trama é do tipo IP.
2. 0x0001: a trama é do tipo ICMP.
3. 0x0806: a trama é do tipo ARP.

6. Como determinar o comprimento de uma trama recetora?

O comprimento de uma trama recetora encontra-se definido no *Ethernet Header* da trama. Com a ajuda do Wireshark é possível observar o comprimento da trama no campo *frame length* (Anexo E).

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

A interface *loopback* é uma interface de rede virtual que permite ao computador comunicar consigo mesmo. Esta interface tem como objetivo realizar testes de diagnóstico, aceder a servidores na própria máquina, como se fosse um cliente. (anexo F).

Experiência 2 – Implementar duas LANs virtuais no switch

A experiência 2 teve como objetivo a configuração de LANs (*vlan*y0 e *vlan*y1).

A rede era composta por três máquinas (*tux* 1, 2 e 4) ligados a um *switch*, com duas *virtual LANs* (VLANs). A *vlan*y0 tem os *tux*s 1 e 4, a outra, a *vlan*y1 máquina 2.

NOTA: A letra y significa o número da bancada em que foram realizados os testes.

*1. Como configurar vlan*y0?

A *vlan*y0 foi configurada seguindo os seguintes passos:

1. Ligar a porta eth0 do *tux*1 à porta 0/1 do switch.
2. Ligar a porta eth0 do *tux*4 à porta 0/2 do switch.
3. Ligar a porta eth0 do *tux*2 à porta 0/3 do switch.
4. Configuração do switch:
 - i. Criação da *vlan*y0.
 - ii. Adicionar à *vlan*y0 as portas do *tux*1 e do *tux*4 (porta 11 e 12).
 - iii. Criação da *vlan*y1.
 - iv. Adicionar à *vlan*y1 a porta do *tux*2 (porta 13).

A sequência de comandos para a criação de um VLAN é a seguinte: > **configure terminal** > **vlan y0** > **end**, em y o número da bancada.

A sequência de comandos para a adição de portas a uma VLAN é a seguinte: > **configure terminal** > **interface fastethernet 0/n** > **switchport mode access** > **switchport access vlan y0** > **end**, em que n representa o número da porta do switch e y o número da bancada.

Os comandos supra devem ser efetuados no GTKTerm depois de ser feito o login com o switch.

2. Quantos domínios de transmissão existem? Como concluir isso através dos registos?

Existem dois domínios de transição: o que contém os *tuxs* 1 e 4 e o que contém o *tux2*. Podemos concluir isso pois, quando é efetuado um *ping broadcast* no *tux1*, este obtém resposta do 4, mas não do 2; quando é efetuado um *ping broadcast* no *tux2*, este não obtém nenhuma resposta.

Experiência 3 – Configurar um router em Linux

A experiência 3 teve como objetivo configurar um computador como *router* para que duas *VLANs* distintas possam comunicar. Esta experiência é similar à experiência 2, com a diferença de que o *tux4* também se encontra na *vlan1* (usando outra *interface*).

1. Que rotas existem nos tuxs? Qual o seu significado?

Na experiência 4, o *tux4* foi configurado como router entre as duas *VLANs* criadas na experiência anterior (o *tux4* passou a fazer parte tanto da *vlan0* como da *vlan1*).

Nas *VLANs* associadas existem duas rotas (Anexo J):

1. *Tux1* para a *vlan0* (172.16.y0.0) pela *gateway* 172.16.y0.1.
2. *Tux2* para a *vlan1* (172.16.y1.0) pela *gateway* 172.16.y1.1.
3. *Tux4* para a *vlan0* (172.16.y0.0) pela *gateway* 172.16.y1.254.
4. *Tux4* para a *vlan1* (172.16.y1.0) pela *gateway* 172.16.y1.253.
5. *Tux1* para a *vlan1* (172.16.y1.0) pela *gateway* 172.16.y0.254.
6. *Tux2* para a *vlan0* (172.16.y0.0) pela *gateway* 172.16.y1.253.

Após a configuração das rotas supra, foi possível pingar o *tux1* a partir do *tux2*. Isto deve-se ao facto do *tux4* estar conectado às duas *VLANs* permitindo a comunicação entre *tuxs* em *VLANs* díspares.

2. Que informação contém uma entrada da tabela de forwarding?

Uma entrada na tabela de *forwarding* contém a seguinte informação:

1. **Destination:** IP de destino da rota.
2. **Gateway:** IP de encaminhamento da rota.
3. **Genmask:** *netmask* da *network*.
4. **Flags:** informações sobre a rota.
5. **Metric:** custo da rota.
6. **Use:** contador de pesquisa pela rota (pode ter o número de falhas (-F) ou sucessos (-C)).

7. *iface*: interface de saída dos pacotes (por exemplo: *eth0* e *eth1*).

3. Que mensagens ARP e endereços MAC associados são observados e porquê?

Quando um *tux* pinga outro, o *tux* pingado não conhece o endereço MAC do que enviou o *ping*. Por isso, envia outra mensagem ARP ‘perguntando’ quando o endereço MAC com o endereço MAC de destino igual ao *default*. Esta primeira mensagem ARP tem como MAC de origem, o endereço MAC do *tux* pingado.

O *tux* que pingou responde com uma mensagem ARP com o seu endereço MAC. Assim, esta segunda mensagem ARP tem como MAC de origem o seu endereço MAC e como MAC de destino o endereço MAC do *tux* pingado.

4. Que pacotes ICMP são observados e porquê?

Como referido anteriormente, com as configurações adicionadas nesta experiência, todos os *tuxs* conseguem comunicar entre si. Deste modo, os pacotes ICMP observados são pacotes de *request* e *reply*. Caso este não fosse o caso, seriam observados pacotes de *host unreachable*.

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

Os endereços IP e MAC associados aos pacotes ICMP são os dos endereços dos *tuxes* de origem e destino, por exemplo, se o *tux1* pingar o *tux4* os endereços de origem vão ser os endereços IP e MAC do *tux1* e os de destino os do *tux4*.

Experiência 4

A experiência 4 teve como objetivo a configuração de um *router* comercial (e a implementação do NAT no *router*). Esta experiência mostra-se semelhante à experiência 3, mas incluindo um *router* comercial na *valny1*.

1. Como se configura um *router* estático num *router* comercial?

O *router* foi configurado seguindo os seguintes passos:

1. Ligar a porta T4, da régua 1, à porta do *router*, na régua 2.
2. Ligar o *router*, porta T3 da régua 1, à porta S0 de um dos *Tuxs*.
3. Configuração do *router* (IP do *router* é 172.16.1.y9/24 – encontra-se na *valny1*):
 - i. Configuração do seu IP.
 - ii. Configuração das suas rotas.

A sequência de comandos de configuração das rotas do *router* é a seguinte: > **configure terminal > ip route [ip rota de destino] [máscara] [ip gateway] > exit.**

A sequência de comandos de configuração do IP do *router* é a seguinte: **interface gigabitethernet 0/0 > ip address IP máscara > no shutdown > exit.**

Os comandos supra devem ser efetuados no *GTKTerm* do *Tux* à qual o *router* se encontra conectado depois de ser feito o login do *router*.

2. *Quais são as rotas seguidas pelos pacotes durante a experiência? Explique.*

Existem duas opções de rotas para os pacotes:

1. Caso a rota exista, os pacotes seguem essa rota.
2. Caso a rota não exista, os pacotes seguem a rota *default*, *tux4*.

3. *Como configura o NAT num router comercial?*

Para configurar o NAT num *router* comercial, configuramos a interface interna no processo de NATL. Esta configuração foi efetuando introduzindo os comandos presentes no anexo G, no *GTKTerm*.

4. *O que faz o NAT?*

O NAT (*Network Address Translation*) vai possibilitar a comunicação entre os computadores da rede criada (rede privada) com redes externas. Sendo uma rede privada, os seus IPs não são reconhecíveis nas redes externas. O NAT vai resolver este problema ao reescrever os IPs de modo a poderem aceder a estas redes externas, gerando um número de 16 bits, guardando esse valor numa *hash table* e escrevendo-o

no campo da porta de origem. Na resposta será só necessário reverter o processo e o *router* passa a saber o endereço para onde vai enviar a resposta.

Experiência 5

A experiência 5 teve como objetivo a configuração do DNS nos *tuxs* que irá permitir o acesso a redes externas (por exemplo, a Internet) através da rede interna criada anteriormente.

1. Como configurar o serviço DNS num host?

Para configurar o DNS, temos que, em todos os *hosts* da rede criada, editar o ficheiro **resolv.conf** de modo a conter o nome do servidor do DNS (*search netlab.fe.up.pt*) e o seu endereço IP (*nameserver 172.16.1.1*) (anexo H).

2. Que pacotes são trocados pelo DNS e que informações são transportadas?

Inicialmente, é transportado um pacote enviado pelo *host* para o *server* que contém o *hostname* desejado, pedindo o seu endereço de IP. (Anexo I, linha 38)

O servidor responde com um pacote que contém o endereço IP do *hostname*. (Anexo I, linha 40).

Experiência 6

Todos os dados recolhidos da aplicação FTP foram obtidos no Tux1 com o comando seguinte: *./download ftp://speedtest.tele2.net/100MB.zip*.

1. Quantas conexões TCP foram abertas pela aplicação FTP?

A aplicação abriu duas conexões TCP: uma para comunicar com o servidor, mandando comandos FTP e recebendo respostas; outra para receber os dados enviados pelo servidor e enviar respostas ao cliente.

2. Em que conexão é transportada a informação de controlo do FTP?

A informação de controlo é transportada na primeira conexão descrita supra: a que troca comandos FTP com o servidor.

3. Quais são as fases de uma conexão TCP?

A conexão TCP possui 3 fases: o estabelecimento da conexão, a troca de informação e o encerramento da conexão.

4. Como funciona o mecanismo ARQ TCP? Que informação relevante pode ser observada nos registos?

O TCP (*Transmission Control Protocol*) utiliza o mecanismo ARQ (*Automatic Repeat ReQuest*) na variante de *Selective Repeat*. O fluxo normal da transmissão de informação pelo TCP na variante de *Selective Repeat* é a seguinte: o emissor envia pacotes numa janela de tamanho N e o recetor confirma a receção de todos os pacotes sejam estes enviados por ordem ou não. Neste caso, o recetor possuiu um buffer para guardar pacotes que não estão por ordem e ordená-los. O transmissor reenvia pacotes perdidos e move a janela para a frente.

Os campos dos pacotes TCP relevantes para o mecanismo de ARQ são o *Sequence Number*, o *Acknowledgement Number* e o *Window Size* assinalados na figura 2.

TCP	66	37273 → 21 [ACK]	Seq=43 Ack=203 Win=29312 Len=0 TSval=16979908 TSecr=1904065596
FTP-D...	1514	FTP Data: 1448 bytes (PASV) (RETR 100MB.zip)	
TCP	66	49106 → 29662 [ACK]	Seq=1 Ack=1449 Win=32128 Len=0 TSval=16979909 TSecr=1904065596

Figura 2 - Captura no Wireshark na interface eth0 do Tux1 dos pacotes gerados pela aplicação de download.

5. Como funciona o mecanismo de congestionamento do TCP? Quais são os campos revelantes? Como evoluiu o fluxo de dados ao longo do tempo? Está de acordo com o mecanismo de congestionamento do TCP?

A Janela de Congestionamento do TCP é uma forma de evitar que a conexão entre o transmissor e o recetor fique sobrecarregada com tráfego. A rede, como o recetor, pode e limita a taxa de transferência de dados. Se a rede não conseguir entregar os dados tão rapidamente quanto criados pelo emissor, o emissor deverá diminuir a velocidade de envio. Noutras palavras, além do recetor, a rede é uma segunda entidade que determina o tamanho da janela do emissor (a janela de congestionamento).

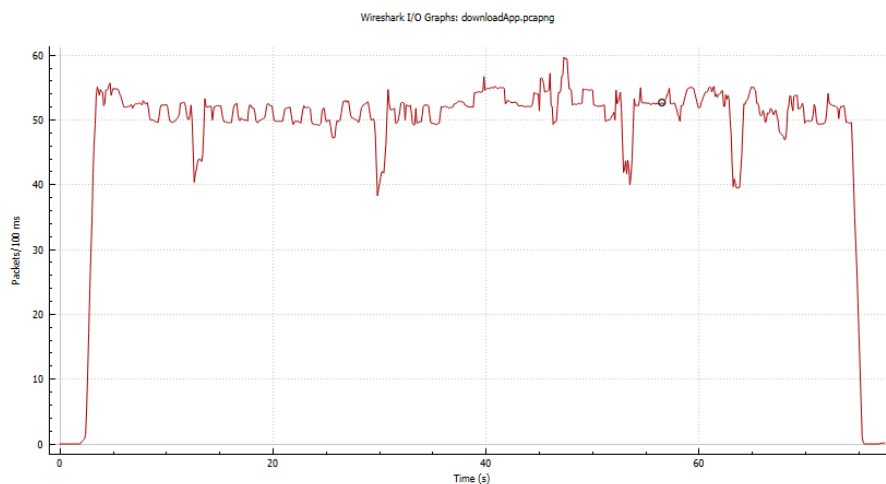


Gráfico 1 – Variação dos Pacotes Recebidos pelo Tux1 Ao Longo do Tempo.

Ao fazer o *download* de um ficheiro usando o cliente FTP desenvolvido, o número de pacotes recebidos por unidade de tempo tem tendência a ir aumentando ao longo do tempo com algumas quedas abruptas. Esta observação é coerente com os mecanismos de controlo de congestão usados em TCP em que a perda de um pacote faz encolher bastante a dimensão da janela de congestionamento, enquanto que incrementos ao comprimento desta são mais graduais e de menor valor.

6. A ligação de dados TCP é afetada pelo aparecimento de uma segunda conexão TCP?

Uma segunda conexão TCP pode levar a uma queda na taxa de transferência de dados, uma vez que a rede deverá transportar dados de duas conexões. Podemos observar este fenómeno comparando o gráfico 1 e 2. A transferência dos dados com o aparecimento de uma segunda conexão TCP demora cerca de 75s, enquanto numa conexão dedicada demora abaixo dos 70s.

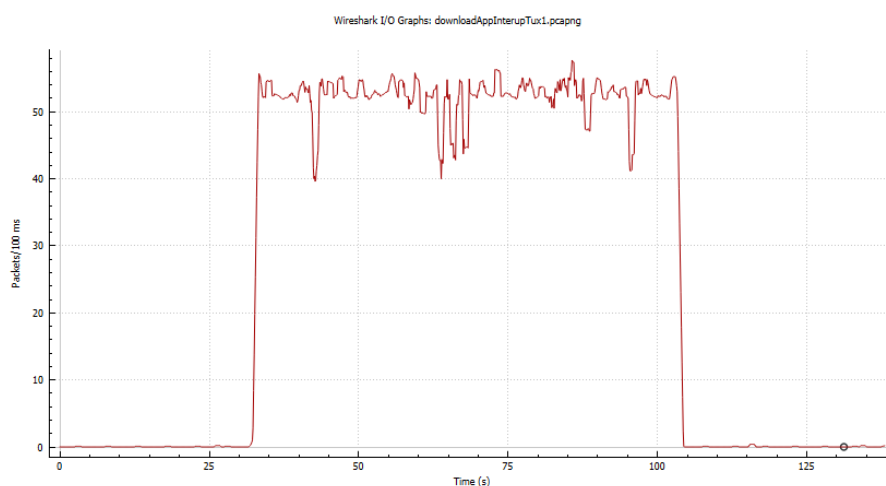


Gráfico 2 – Variação dos Pacotes Recebidos pelo Tux1 Ao Longo do Tempo com o aparecimento de uma segunda conexão TCP.

Conclusão

Este segundo projeto de Redes e Computadores, teve por objetivo a configuração de uma rede e a implementação do cliente de download.

O trabalho foi concluído com sucesso, onde cumprimos todos os objetivos propostos, sendo que consideramos que a elaboração deste trabalho foi fundamental para consolidar a matéria lecionada nas aulas teóricas e praticas.

Após a conclusão do segundo projeto da unidade curricular de Redes de Computadores (*RCOM*), os elementos do grupo viram-se minados de conhecimentos básicos para uma coerente implementação do guião fornecido para o desenvolvimento da aplicação *download*.

A configuração de rede foi concluída com sucesso dando a todos os elementos do grupo uma noção de como uma rede funciona e assim, possivelmente, aplicar esta prática a nível profissional.

Anexos

Anexo A – Pacote ARP para descodificação de endereço

MAC

27	14.067797774	Netronix_50:3f:2c	Kye_08:d5:b2	ARP	60	Who has 172.16.20.1? Tell 172.16.20.254
28	14.067825579	Kye_08:d5:b2	Netronix_50:3f:2c	ARP	42	172.16.20.1 is at 00:c0:df:08:d5:b2
29	14.868358767	172.16.20.1	172.16.20.254	ICMP	98	Echo (ping) request id=0xad7, seq=7/179
30	14.868471068	172.16.20.254	172.16.20.1	ICMP	98	Echo (ping) reply id=0xad7, seq=7/179

<

> Frame 27: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface eth0, id 0
> Ethernet II, Src: Netronix_50:3f:2c (00:08:54:50:3f:2c), Dst: Kye_08:d5:b2 (00:c0:df:08:d5:b2)
v Address Resolution Protocol (request)
 Hardware type: Ethernet (1)
 Protocol type: IPv4 (0x0800)
 Hardware size: 6
 Protocol size: 4
 Opcode: request (1)
 Sender MAC address: Netronix_50:3f:2c (00:08:54:50:3f:2c)
 Sender IP address: 172.16.20.254
 Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
 Target IP address: 172.16.20.1

Anexo B – Pacote ARP de resposta para descodificação de endereço MAC

28	14.067825579	Kye_08:d5:b2	Netronix_50:3f:2c	ARP	42	172.16.20.1 is at 00:c0:df:08:d5:b2
29	14.868358767	172.16.20.1	172.16.20.254	ICMP	98	Echo (ping) request id=0xad7, seq=7/179

<

>

> Frame 28: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface eth0, id 0
> Ethernet II, Src: Kye_08:d5:b2 (00:c0:df:08:d5:b2), Dst: Netronix_50:3f:2c (00:08:54:50:3f:2c)
v Address Resolution Protocol (reply)
 Hardware type: Ethernet (1)
 Protocol type: IPv4 (0x0800)
 Hardware size: 6
 Protocol size: 4
 Opcode: reply (2)
 Sender MAC address: Kye_08:d5:b2 (00:c0:df:08:d5:b2)
 Sender IP address: 172.16.20.1
 Target MAC address: Netronix_50:3f:2c (00:08:54:50:3f:2c)
 Target IP address: 172.16.20.254

Anexo C – Pacote Pedido Ping

24	13.808475204	172.16.20.254	172.16.20.1	ICMP	98 Echo (ping) reply	Id=0x0ad7, Seq=0/13
25	13.876319319	Kye_08:d5:b2	Netronix_50:3f:2c	ARP	42 Who has 172.16.20.254? Tell 172.16.20.1	
26	13.876399800	Netronix_50:3f:2c	Kve_08:d5:b2	ARP	60 172.16.20.254 is at 00:08:54:50:3f:2c	
<						
>						
Frame 25: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface eth0, id 0						
Ethernet II, Src: Kye_08:d5:b2 (00:c0:df:08:d5:b2), Dst: Netronix_50:3f:2c (00:08:54:50:3f:2c)						
Address Resolution Protocol (request)						
Hardware type: Ethernet (1)						
Protocol type: IPv4 (0x0800)						
Hardware size: 6						
Protocol size: 4						
Opcode: request (1)						
Sender MAC address: Kye_08:d5:b2 (00:c0:df:08:d5:b2)						
Sender IP address: 172.16.20.1						
Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)						
Target IP address: 172.16.20.254						

Anexo D – Pacote de Resposta Ping

25	13.876319319	Kye_08:d5:b2	Netronix_50:3f:2c	ARP	42 Who has 172.16.20.254? Tell 172.16.20.1	
26	13.876399800	Netronix_50:3f:2c	Kve_08:d5:b2	ARP	60 172.16.20.254 is at 00:08:54:50:3f:2c	
<						
>						
Frame 26: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface eth0, id 0						
Ethernet II, Src: Netronix_50:3f:2c (00:08:54:50:3f:2c), Dst: Kye_08:d5:b2 (00:c0:df:08:d5:b2)						
Address Resolution Protocol (reply)						
Hardware type: Ethernet (1)						
Protocol type: IPv4 (0x0800)						
Hardware size: 6						
Protocol size: 4						
Opcode: reply (2)						
Sender MAC address: Netronix_50:3f:2c (00:08:54:50:3f:2c)						
Sender IP address: 172.16.20.254						
Target MAC address: Kye_08:d5:b2 (00:c0:df:08:d5:b2)						
Target IP address: 172.16.20.1						

Anexo E – Tamanho da Trama Recetora

24	13.808475204	172.16.20.254	172.16.20.1	ICMP	98 Echo (ping) reply	Id=0x0ad7, Seq=0/13
25	13.876319319	Kye_08:d5:b2	Netronix_50:3f:2c	ARP	42 Who has 172.16.20.254? Tell 172.16.20.1	
26	13.876399800	Netronix_50:3f:2c	Kve_08:d5:b2	ARP	60 172.16.20.254 is at 00:08:54:50:3f:2c	
<						
>						
Frame 25: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface eth0, id 0						
Interface id: 0 (eth0)						
Encapsulation type: Ethernet (1)						
Arrival Time: Nov 14, 2019 11:54:43.986458361 Hora padrão de GMT						
[Time shift for this packet: 0.000000000 seconds]						
Epoch Time: 1573732483.986458361 seconds						
[Time delta from previous captured frame: 0.007844115 seconds]						
[Time delta from previous displayed frame: 0.007844115 seconds]						
[Time since reference or first frame: 13.876319319 seconds]						
Frame Number: 25						
Frame Length: 42 bytes (336 bits)						
Capture Length: 42 bytes (336 bits)						
[Frame is marked: False]						
[Frame is ignored: False]						

Anexo F – Interface Loopback

6	6.094260374	Cisco_5c:4d:83	Cisco_5c:4d:83	LOOP	60 Reply
7	6.937253028	Cisco_5c:4d:83	Spanning-tree-(for...	STP	60 Conf. Root = 32768/1/fc:fb:fb:5c:4
8	8.870609784	172.16.20.1	172.16.20.254	ICMP	98 Echo (ping) request id=0x0ad7, se
9	8.870740571	172.16.20.254	172.16.20.1	ICMP	98 Echo (ping) reply id=0x0ad7, se
10	8.950004108	Cisco_5c:4d:83	Spanning-tree-(for...	STP	60 Conf. Root = 32768/1/fc:fb:fb:5c:4
11	9.869614039	172.16.20.1	172.16.20.254	ICMP	98 Echo (ping) request id=0x0ad7, se
12	9.869724630	172.16.20.254	172.16.20.1	ICMP	98 Echo (ping) reolv id=0x0ad7. se

< >

> Frame 6: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface eth0, id 0

> Ethernet II, Src: Cisco_5c:4d:83 (fc:fb:fb:5c:4d:83), Dst: Cisco_5c:4d:83 (fc:fb:fb:5c:4d:83)

✓ Configuration Test Protocol (loopback)

 skipCount: 0

 Relevant function: Reply (1)

 Function: Reply (1)

 Receipt number: 0

> Data (40 bytes)

Anexo G – Configuração do Router Cisco com NAT

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

interface gigabitethernet 0/1*
ip address 172.16.1.y9 255.255.255.0
no shutdown
ip nat outside
exit

ip nat pool ovrlld 172.16.1.y9 172.16.1.y9 prefix 24
ip nat inside source list 1 pool ovrlld overload

access-list 1 permit 172.16.y0.0 0.0.0.7
access-list 1 permit 172.16.y1.0 0.0.0.7

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

* In room I320 use interface fastethernet

Anexo H – Conteúdo do ficheiro resolv.conf para a configuração de um DNS

```
# Generated by NetworkManager
domain netlab.fe.up.pt
search netlab.fe.up.pt fe.up.pt
nameserver 172.16.1.1
nameserver 193.136.28.10
```

Anexo I – Pacotes trocados pelo DNS

38	4.318964489	172.16.40.1	172.16.1.1	DNS	69 Standard query 0x6c13 A ftp.up.pt
39	4.319000716	172.16.40.1	172.16.1.1	DNS	69 Standard query 0x6c13 AAAA ftp.up.pt
40	4.321856221	172.16.1.1	172.16.40.1	DNS	534 Standard query response 0x6c13 A ftp.up.pt CNAME mirrors.up.pt A 193.137.29.15 NS sns-pb.isc.org NS ns.dns.br NS f.dns.pt NS c-
41	4.321877385	172.16.1.1	172.16.40.1	DNS	546 Standard query response 0x6c13 AAAA ftp.up.pt CNAME mirrors.up.pt AAAA 2001:690:2200:1200::15 NS sns-pb.isc.org NS a.dns.pt NS-
42	4.322359702	172.16.40.1	193.137.29.15	ICMP	98 Echo (ping) request id=0x3d08, seq=1/256, ttl=64 (reply in 43)
43	4.325412865	193.137.29.15	172.16.40.1	ICMP	98 Echo (ping) reply id=0x3d08, seq=1/256, ttl=57 (request in 42)

Anexo J – Rotas na Experiência 3

```
tux41:~# route -n
Kernel IP routing table
Destination      Gateway         Genmask         Flags Metric Ref    Use Iface
172.16.40.0      172.16.40.1    255.255.255.0   UG      0      0      0 eth0
172.16.40.0      0.0.0.0        255.255.255.0   U        0      0      0 eth0
172.16.41.0      172.16.40.254  255.255.255.0   UG      0      0      0 eth0
tux41:~#

root@tux42:~# route -n
Kernel IP routing table
Destination      Gateway         Genmask         Flags Metric Ref    Use Iface
172.16.40.0      172.16.41.253  255.255.255.0   UG      0      0      0 eth0
172.16.41.0      172.16.41.1    255.255.255.0   UG      0      0      0 eth0
172.16.41.0      0.0.0.0        255.255.255.0   U        0      0      0 eth0
root@tux42:~#

root@tux44:~# route -n
Kernel IP routing table
Destination      Gateway         Genmask         Flags Metric Ref    Use Iface
172.16.40.0      172.16.40.254  255.255.255.0   UG      0      0      0 eth0
172.16.40.0      0.0.0.0        255.255.255.0   U        0      0      0 eth0
172.16.41.0      172.16.41.253  255.255.255.0   UG      0      0      0 eth1
172.16.41.0      0.0.0.0        255.255.255.0   U        0      0      0 eth1
root@tux44:~#
```

Anexo M

downloadClient.c

```
#include <stdio.h>

#include "url.h"
#include "server.h"

//ftp://anonymous:1@speedtest.tele2.net/1KB.zip
int main(int argc, char **argv)
{
    if (argc != 2){
        printf("Usage: %s ftp://[<user>:<password>@]<host>/<url-  
path>\n", argv[0]);
    }
```

```

        exit(1);

    }

    url URL;
    initURL(&URL);

    //1. Interpret command line
    if (parseURL(&URL, argv[1]) != 0)
        return 1;

    //2. Get Ip Address (similar to getip.c)
    if(getIpHost(&URL) != 0)
        return 1;

    server server;

    //3. Create TCP Socket (similar to clientTCP.c): using port 21 and IP
    address recovered in 2 and check if connection went through
    if (connectToServer(&server, URL.ip, URL.port) != 0)
        return 1;

    //4. Login User
    if(loginServer(server, URL.user, URL.password) != 0)
        return 1;

    //5. Entering Passive Mode and Creating a TCP Socket for data
    if(psvModeServer(&server) != 0)
        return 1;

    //6. Sending
    if(retrServer(server, URL.path) != 0)
        return 1;

    //7. Download From Server (tcp socket for data)
    if(downloadFromServer(server, URL.filename) != 0)
        return 1;

    //8. Disconnect from server
    if(disconnectToServer(server) != 0)
        return 1;

    return 0;
}

```

url.h

```
#include "string.h"

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

typedef struct URL {
    char user[256]; // string to user
    char password[256]; // string to password
    char host[256]; // string to host
    char ip[256]; // string to IP
    char path[256]; // string to path
    char filename[256]; // string to filename
    int port; // integer to port
} url;

void initURL(url* url);

int parseURL(url* url, char * argv);

int getIpHost(url* url);
```

url.c

```
#include "url.h"

void initURL(url* url) {

    url->port = 21;

}

int parseURL(url* url, char * argv){

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

    if (sscanf(argv, "ftp://%[^:]:%[^@]@%[/]/%s", url->user, url->password, url->host, url->path) != 4)
    {

        if(sscanf(argv, "ftp://%[/]/%s", url->host, url->path) != 2)
        {
```

```

        printf("Usage: %s ftp://[<user>:<password>@]<host>/<url-
path>\n", argv);
        return 1;
    }

    strcpy(url->user, "anonymous");
    strcpy(url->password, "1");
}

char * ptr = strrchr(url->path, '/');

if(ptr != NULL)
    ptr++;

if(ptr == NULL)
    strcpy(url->filename, url->path);
else
    strcpy(url->filename, ptr);

return 0;
}

int getIpHost(url* url){

    struct hostent *h;

    if ((h=gethostbyname(url->host)) == NULL) {
        perror("gethostbyname");
        return 1;
    }

    /*printf("Host name   : %s\n", h->h_name);
    printf("IP Address  : %s\n",inet_ntoa(*(struct in_addr *)h-
>h_addr));*/

    char* ip = inet_ntoa(*(struct in_addr *) h->h_addr));
    strcpy(url->ip, ip);

    return 0;
}

```

server.h

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

#include <sys/socket.h>
#include <unistd.h>
#include <signal.h>
#include <strings.h>

typedef struct SERVER
{
    int fd_control_socket; // file descriptor to control socket
    int fd_data_socket; // file descriptor to data socket
} server;

int clientTCP(char ip[], int port);

int connectToServer(server * server, char ip[], int port);

int readFromServer(server server, char * response, int size);

int loginServer(server server, char username[], char password[]);

int psvModeServer(server * server);

int retrServer(server server, char path[]);

int downloadFromServer(server server, char filename[]);

int disconnectToServer(server server);

int sendToServer(server server, char command[]);
```

server.c

```
#include "server.h"

#include "string.h"

int getIpFromResponse(char response[], char ** ip, int * port) {

    //227 Entering Passive Mode (193,137,29,15,221,177).

    int ip1, ip2, ip3, ip4;
    int port1, port2;

    if (sscanf(response, "227 Entering Passive Mode (%d,%d,%d,%d,%d,%d)",
        &ip1, &ip2, &ip3, &ip4, &port1, &port2) < 0) {

        printf("Cannot process information to calculating ip and port.\n"
    );

        return 1;
    }

    //IP
    if (sprintf((*ip), "%d.%d.%d.%d", ip1, ip2, ip3, ip4) < 0) {

        printf("Cannot form ip address.\n");
        return 1;
    }

    //Port
    (*port) = port1 * 256 + port2;

    return 0;
}

int checkNumCode(char response[], char expected_num_code[]){

    if(strncmp(response, expected_num_code, 3) == 0)
        return 0;

    return 1;
}

int clientTCP(char ip[], int port){

    int sockfd;
```



```

    struct sockaddr_in server_addr;

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

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

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

    return sockfd;
}

int connectToServer(server * server, char ip[], int port){

    server->fd_control_socket = clientTCP(ip, port);

    if(server->fd_control_socket == -1)
        return 1;

    char response[1024];

    if (readFromServer(*server, response, sizeof(response)) != 0)
        return 1;

    if(checkNumCode(response, "220") != 0)
    {
        printf(" > Service is not ready!\n");
        return 1;
    }

    printf(" > Service is ready!\n");
    return 0;
}

```

```

}

int loginServer(server server, char username[], char password[]){

    char command[1024];
    char response[1024];

    sprintf(command, "USER %s\n", username);

    if(sendToServer(server, command) != 0)
        return 1;

    printf(" > %s", command);

    if (readFromServer(server, response, sizeof(response)) != 0)
        return 1;

    if(checkNumCode(response, "331") != 0){

        printf("Response was not what was expected\n");
        return 1;
    }

    sprintf(command, "PASS %s\n", password);

    if(sendToServer(server, command) != 0)
        return 1;

    printf(" > %s", command);

    if (readFromServer(server, response, sizeof(response)) != 0)
        return 1;

    if(checkNumCode(response, "230") != 0){

        printf("Response was not what was expected\n");
        return 1;
    }

    return 0;
}

int psvModeServer(server * server){

    char command[1024];
    char response[1024];

    sprintf(command, "PASV\n");

```

```

    if(sendToServer(*server, command) != 0)
        return 1;

    printf(" > %s", command);

    if (readFromServer(*server, response, sizeof(response)) != 0)
        return 1;

    if(checkNumCode(response, "227") != 0){

        printf("Response was not what was expected\n");
        return 1;
    }

    char * ip = malloc(1024*sizeof(char));
    int port;
    if (getIpFromResponse(response, &ip, &port) != 0)
        return 1;

    server->fd_data_socket = clientTCP(ip, port);

    if(server->fd_data_socket == -1)
        return 1;

    return 0;
}

int retrServer(server server, char path[]) {

    char command[1024];
    char response[1024];

    sprintf(command, "RETR %s\n", path);

    if(sendToServer(server, command) != 0)
        return 1;

    printf(" > %s", command);

    if (readFromServer(server, response, sizeof(response)) != 0)
        return 1;

    if(checkNumCode(response, "150") != 0){

        printf("Response was not what was expected\n");
        return 1;
    }
}

```

```

        return 0;
    }

int downloadFromServer(server server, char filename[]){

    FILE* file;
    int received;

    if (!(file = fopen(filename, "w"))) {
        perror("downloadFromServer");
        return 1;
    }

    char buf[1024];

    while ((received = read(server.fd_data_socket, buf, sizeof(buf))) > 0
) {

        if (received < 0) {
            perror("downloadFromServer");
            return 1;
        }

        if ((received = fwrite(buf, received, 1, file)) < 0) {
            perror("downloadFromServer");
            return 1;
        }
    }

    fclose(file);
    close(server.fd_data_socket);

    return 0;
}

int disconnectToServer(server server){

    char response[1024];

    if (readFromServer(server, response, sizeof(response)) != 0)
        return 1;

    char command[1024];

    sprintf(command, "QUIT\r\n");

```

```

    if(sendToServer(server, command) != 0)
        return 1;

    printf(" > %s", command);

    if (server.fd_control_socket)
        close(server.fd_control_socket);

    if(checkNumCode(response, "226") != 0){

        printf("Response was not what was expected\n");
        return 1;
    }

    return 0;
}

int sendToServer(server server, char command[]){

    if (write(server.fd_control_socket, command, strlen(command)) <= 0) {
        perror("sendToServer");
        return 1;
    }

    return 0;
}

int readFromServer(server server, char * response, int size){

    FILE* fp = fdopen(server.fd_control_socket, "r");

    do {

        memset(response, 0, size);

        response = fgets(response, size, fp);

        if(response == NULL)
        {
            perror("readFromServer");
            return 1;
        }

        //int incr = recv(server.fd_control_socket, response, size, 0);

        printf(" < %s", response);
    } while (response != NULL);
}

```

```
    } while (!('1' <= response[0] && response[0] <= '5') || response[3] != ' ');  
  
    return 0;  
}
```