Redes de Computadores

Ângela Cardoso e Bruno Madeira

23 de Dezembro de 2015

Sumário

Este relatório tem como objectivo reportar o segundo trabalho prático relativo a Redes de Computadores da Licenciatura com Mestrado em Engenharia Informática e Computação que consiste na configuração de uma rede e na implementação de uma aplicação de download de ficheiros.

Conteúdo

1 Introdução **3**

2 Aplicação **4**

3 Experiências **5**

3.1 Experiência 1 - Configurar uma Rede IP 5
3.2 Experiência 2 - Implementar 2 LANs num switch 5
3.3 Experiência 3 - Configurar um Router em Linux 5
3.4 Experiência 4 - Configurar um Router Comercial e Implementar NAT 6
3.5 Experiência 5 - DNS 6
3.6 Experiência 6 - Conexões TCP 7
3.7 Experiência 7 - Implementar NAT em Linux 8

4 Conclusões **9**

5 Bibliografia **10**

Anexos **11**

A Enderaços MAC **12**

B Scripts finais de configuração, excluindo experiência 7 **13**

B.1 tux1 13
B.2 tux2 13
B.3 tux4 13
B.4 switch 14
B.5 router 14

C Logs and statistics **15**

C.1 Experiência 1 15
C.1.1 Captura Wireshark no tux1 - ARP 15
C.1.2 Captura Wireshark no tux1 - ICMP 15
C.2 Experiência 2 16
C.2.1 Alínea 7 - Captura Wireshark no tux1 16
C.2.2 Alínea 7 - Captura Wireshark no tux2 16
C.2.3 Alínea 7 - Captura Wireshark no tux4 16
C.2.4 Alínea10 - Captura Wireshark no tux1 17
C.2.5 Alínea10 - Captura Wireshark no tux2 17
C.2.6 Alínea10 - Captura Wireshark no tux4 17
C.3 Experiência 3 17
C.3.1 Capturas Wireshark no tux4.eth0 17
C.3.2 Capturas Wireshark no tux4.eth1 18

C.4 Experiência 4 . . . 19
C.4.1 Alínea 4 Console Log . . . 19
C.5 Experiência 5 . . . 20
C.5.1 Capturas Wireshark de DNS no tux1 . . . 20
C.6 Experiência 6 . . . 20
C.6.1 Capturas Wireshark dos 'hanshakes' no tux1 . . . 20
C.6.2 Primeiro 'Handshake' no tux1 em Detalhe . . . 20
C.6.3 Capturas Wireshark no tux1 de ACKs . . . 21
C.6.4 Capturas Wireshark no tux1 de Dup ACK, Fast Retransmission e Retransmission 22
C.6.5 Gráfico de Tráfego no tux1 . . . 23
C.6.6 Gráfico de Tráfego no tux2 . . . 23
C.6.7 Gráfico de Window Size no tux1 de pacotes TCP recebidos na porta de dados . . 24
C.7 Experiência 7 . . . 24
C.7.1 Capturas Wireshark TCP no tux4.eth0 e tux4.eth1 . . . 24
C.7.2 Captura Wireshark UDP e timeout no tux4.eth0 . . . 25
C.7.3 Capturas Wireshark TCP no tux4.eth0 . . . 25

D Código Fonte **26**
D.1 Ficheiro downloader.c . . . 26
D.2 Ficheiro ftp.h . . . 28
D.3 Ficheiro ftp.c . . . 28
D.4 Ficheiro socket.h . . . 33
D.5 Ficheiro socket.c . . . 33
D.6 Ficheiro Utilities.h . . . 34

1 Introdução

No âmbito da unidade curricular de Redes de Computadores foi-nos proposta a realização de um trabalho prático cujo objetivo principal era configurar uma rede e compreender os vários aspetos dessa mesma configuração. Além disso, implementamos também uma aplicação de download de ficheiros, por forma a testar uma parte da rede.

O primeiro capítulo deste relatório incide sobre a aplicação desenvolvida. A aplicação de download de ficheiros foi implementada fora das aulas práticas e o relatório tenta esclarecer detalhes de implementação da mesma, a fim de eliminar possíveis dúvidas que possam surgir.

O segundo capítulo do relatório incide sob os sete exercícios realizados nas aulas práticas relacionados com a configuração de rede. O relatório evita relatar os exercícios detalhadamente uma vez que estes podem ser consultados no guião do trabalho e tenta focar-se mais na análise e interpretação dos resultados obtidos com o software Wireshark.

Na análise de dados realizada no segundo capítulo do relatório pode ser útil consultar o Anexo A que apresenta os endereços MAC dos tuxs. É importante referir também que apesar deste relatório referir muitas vezes o tux2, da experiência 4 até à 7, qualquer referência ao tux2 corresponde na realidade ao tux3 uma vez que o tux2 deixou de estar disponível a partir de dada altura. Para que o relatório respeite os nomes referidos no guião e usados nos anexos, mantendo a continuidade entre experiências, decidimos continuar a referir-nos ao terceiro computador usado na rede como sendo o tux2.

No final do relatório apresentamos uma conclusão com as nossas considerações face ao trabalho, assim como uma auto-avaliação da nossa prestação.

2 Aplicação

A aplicação desenvolvida realiza o download de um ficheiro fazendo uso do protocolo FTP, cuja especificação se encontra em RFC959. Para tal são usadas duas sockets, uma para comandos e outra para dados, de acordo com o modelo descrito na Secção 2.3 de RFC959. Os comandos usados podem ser verificados na Secção 4 (páginas 25 a 34) e na página 47 de RFC959. É usado o comando PASV sendo que o servidor não usa a porta default para os dados (porta 20) e fica à espera que o cliente estabeleça a ligação.

Todas as funcionalidades desenvolvidas ligadas ao protocolo FTP podem ser verificadas no ficheiro `ftp.c` e `ftp.h` disponíveis nos Anexos D.3 e D.2, respetivamente. Apesar de existir uma função denominada `ftp_abort` esta não envia um comando `ABORT` (embora esta tenha sido a funcionalidade inicialmente pensada para o mesmo). Esta função apenas fecha as sockets em caso de erro.

Para efectuar ligação ao servidor a aplicação deve também receber um `URL` no formato descrito em RFC1738. Não consideramos utilizadores anónimos, apesar do que é referido na Secção 3.2.1. de RFC1738. No ficheiro `downloader.c` (ver Anexo D.1) é realizado o parsing do `URL` ficando guardado numa estrutura o nome de utilizador, a sua password, o nome do host, o caminho até ao ficheiro e o nome do ficheiro.

Uma vez realizado o parsing tenta-se obter o IP do destino e cria-se uma ligação TCP para a porta 21 do servidor a fim de enviar os comandos para pedir a recepção do ficheiro. As funções usadas para obter o IP e para estabelecer são as disponibilizadas nos exemplos do moodle da disciplina. A conexão é realizada com a função `connect` e não com a função `bind` uma vez que a aplicação está do lado do cliente. É utilizada a função `gethostbyname` para obter o IP, que funciona mas está depreciada segundo Beej's Guide to Network Programming.

Em termos de estrutura foram desenvolvidos apenas 4 módulos que apresentamos seguidamente.

- `downloader` - Onde se encontra a função `main` da aplicação. Também é responsável pelo parsing e por obter o IP do destino.
- `ftp` - Implementa e disponibiliza comandos do protocolo FTP. Os file descriptors das sockets também se encontram neste módulo.
- `socket` - Apenas disponibiliza uma função para conectar sockets.
- `utilities` - Apenas disponibiliza funções auxiliares para debug.

3 Experiências

3.1 Experiência 1 - Configurar uma Rede IP

Nesta experiência criou-se uma rede LAN com o tux1 e o tux4, tendo sido configurados os seus endereços IP. Usando o comando `ping` na etapa 7, pudemos verificar o envio de um comando ARP em broadcast pelo tux1 que procurava o endereço físico do tux4, necessário ao protocolo ethernet usado para poder comunicar dentro de uma mesma rede local. Seguidamente verificou-se a resposta do tux4 e foi realizado o ping com sucesso.

Atentando nos pacotes capturados com o Wireshark do Anexo C.1, é possível verificar que as tramas de tipo ARP são identificáveis pelo cabeçalho Ethernet x0806, enquanto que os pacotes IP têm o cabeçalho x0800. As mensagens de ping podem ser identificadas pelo cabeçalho Ethernet correspondente ao protocolo IP e pelo cabeçalho de IP x01 que corresponde ao protocolo ICMP.

Verificamos também que o tamanho da trama recebida encontra-se indicado entre o bit 16 e 31 do cabeçalho IP tal como é descrito na Secção 3.1 do RFC791

Na lista de pacotes recebidos existem também pacotes do tipo loopback. Estes são pacotes são redireccionados para a máquina que os emitiu, tipicamente com a finalidade de verificar se esta se encontra em estado operacional. Neste caso, os pacotes recebidos aparentam ser do switch, tendo como endereço de origem e destino o CiscoInc_3a:f1:03.

3.2 Experiência 2 - Implementar 2 LANs num switch

Foram criadas duas LANs. A primeira, com o tux1 e o tux4 na rede 172.16.60.0 (máscara de 24 bits), corresponde à experiência 1. A outra, com o tux2, na rede 172.16.61.0. Foram atribuídos endereços IP às máquinas relativos à rede em que se deviam encontrar e configurando o switch de modo a funcionarem como 2 redes distintas. Constatou-se que apenas os computadores que se encontravam na mesma rede virtual local podiam comunicar entre si.

Nos Anexos C.2.2 e C.2.3 verifica-se que pings realizados do tux1 em broadcast (alínea 7 do trabalho prático) chegam ao tux4 mas não ao tux2. Similarmente, não foi possível encontrar pacotes de ICMP no tux1 e no tux4 quando realizado ping a partir do tux2 como se pode observar nos restantes anexos da Secção C.2. Pode-se concluir que apenas existiam dois domínios de broadcast (broadcast domains).

3.3 Experiência 3 - Configurar um Router em Linux

No seguimento da experiência anterior, foi configurada a rede de modo a que o tux4 funcionasse como um router entre as duas LANs criadas. O tux4.eth0 continuou com o endereço 172.16.60.254 e ao tux4.eth1 foi atribuído o endereço 172.16.61.253. Foram também reconfigurados o tux1 e tux2 de modo a fazerem uso do router (tux4) para poderem comunicar entre si.

Nas tabelas de encaminhamento (forwarding tables) do tux1 e tux2 aparecem, respectivamente, os gateways 172.16.60.254 e 172.16.61.253 para aceder à rede vizinha. Estes gateways são os endereços para os quais devem ser encaminhados os pacotes IP que apresentam um endereço da rede vizinha como

destino. Os ARPs enviados quando o tux1 pretende comunicar com o tux2 (ou vise-versa), percorrem apenas a LAN na qual foram emitidos com o objectivo de descobrir o endereço MAC do gateway. Os pacotes capturados pelo Wireshark no Anexo C.3.1 ilustram esta situação. No Anexo referido estamos à escuta no tux4.eth0 e podemos verificar que é recebido um ARP de origem no tux1 a perguntar pelo endereço MAC do tux4.eth0. O tux1 quer realizar ping ao tux2 como se pode concluir pelos pacotes ICMP seguintes, e, só o faz depois de receber a resposta do tux4 ao seu ARP, que é necessário ao protocolo ethernet na camada de enlace (data-link layer).

Relativamente aos endereços dos pacotes ICMP é possível verificar que apresentam sempre o mesmo endereço IP de origem e destino na camada de rede (network layer), mas que o endereço MAC de origem e destino varia consoante a rede em que se encontram. Um pacote de ping ICMP proveniente do tux1 para o o tux2 apresenta inicialmente o endereço de origem do tux1.eth0 e de destino o tux4.eth0 (gateway). Depois de recebido pelo gateway (tux4) é enviado para o tux2 com os endereços MAC de origem em tux4.eth1 e destino tux2.eth0. O Anexo C.3.2 mostra esta última situação.

3.4 Experiência 4 - Configurar um Router Comercial e Implementar NAT

A experiência quatro é composta por duas partes. A primeira consiste em conectar um router comercial, RC, à rede do laboratório e à rede 172.16.61.0/24 e definir como routers default o tux4 para o tux1 e o RC para o tux2 e tux4. Esta configuração fez com que os pacotes enviados do tux2 para o tux1, após a remoção da rota na alínea quatro, percorram um caminho maior sendo encaminhados para o RC que estava definido como default e só depois enviados para o tux4. Usando a rota via tux4 o encaminhamento foi directo. Quando não se usou esta rota e se activou o redirecionamento ICMP o tux2 foi informado que existe uma rota melhor via tux4 pelo RC. No Anexo C.4.1 apresentamos o output em consola do traceroute ao fazer uso do redirecionamento ICMP.

A segunda parte consistiu em adicionar a funcionalidade de NAT (Network Address Translation) ao RC. O NAT permite criar uma separação entre uma rede LAN e uma outra rede (tipicamente maior, WAN por exemplo). Esta separação permite usar IPs dentro da LAN que podem já estar em uso fora desta. Funciona como solução ao limite de endereços do IPv4 e confere alguma segurança adicional à rede não permitindo acessos directos às máquinas desta. Na prática ele mapeia portas do gateway a pares de endereço e porta dentro da LAN. Na experiência 7 veremos um pouco melhor tudo isto.

3.5 Experiência 5 - DNS

Na experiência 5 foi configurado o DNS (Domain Name System) com o servidor lixa.netlab.fe.up.pt (172.16.1.1) alterando o ficheiro resolv.conf.

O DNS permite associar strings a endereços. Graças a ele pode-se aceder a sites/plataformas sem ter que usar os seus endereços directamente. O nome, de um site a que se quer aceder por exemplo, é verificado no servidor DNS definido e caso exista é devolvido o respectivo IP. Caso o servidor DNS não tenha conhecimento do IP respectivo pode questionar outros servidores DNS por este.

Pode verificar-se a query DNS e a respectiva resposta do ping que realizamos para o *sapo.pt* no Anexo C.5.1.

3.6 Experiência 6 - Conexões TCP

Nesta experiência usámos a aplicação desenvolvida para realizar o download de um ficheiro. Foi chamada a aplicação inicialmente no tux1 e seguidamente após um pequeno intervalo de tempo no tux2. Foi observado o trafego nos 2 tuxs através do Wireshark. Observou-se que o tux1 e tux2 realizaram parte do seu download em simultâneo e consequentemente as velocidades de recepção dos ficheiros descarregados em ambos foram afectadas.

O protocolo TCP é um protocolo orientado a conexões sendo necessário estabelecer ligação entre o cliente e servidor. Pudemos verificar o *3-way handshake* de duas conexões TCP no tux1. O primeiro relativo à ligação usada para envio de comandos e o segundo relativo a de envio de dados que podem ser verificados no Anexo C.6.1. Este estabelecimento de conexão consiste num pedido do cliente ao servidor (SYN) seguido da resposta do servidor (SYN, ACK) e de uma confirmação final pelo cliente (ACK) que podem ser melhor observados no Anexo C.6.2, onde é mostrado também o número de sequência e de confirmação em cada pacote.

O TCP é também um protocolo fiável. Parte desta fiabilidade é conferida por um mecanismo de ARQ (Automatic Repeat Request) que no TCP é uma variante do Go-Back-N, onde o servidor envia confirmações relativas a cada segmento que recebe. No Anexo C.6.3 é mostrada uma destas confirmações em detalhe.

Outra característica do TCP é a sua capacidade de se adaptar à rede e ao hardware.

Pouco depois do tux1 atingir o seu plateau máximo de tráfego, entre os 14 e 15 segundos do gráfico C.6.5, podem ser observadas vários pacotes do tipo [TCP Dup ACK], [Previous Segment not captured, [TCP Fast Retransmission], [TCP Out-Of-Order] e [TCP Retransmission], que parecem indicar congestionamento na rede causado pelo slow-start do protocolo TCP. O gráfico de I/O e de window size nos Anexos C.6.5 e C.6.7 parecem sugerir o mesmo, podendo observar-se um aumento na taxa de transmissão/recepção e no tamanho da janela até ao segundo 14.

Observamos alguns dos comportamentos do TCP no tux1 no momento referido acima. Segundo o RFC2581 o receptor deve enviar um duplicate ACK quando é recebido um segmento fora de ordem e pode ocorrer uma retransmissão, *fast retransmit*, após a recepção de 3 confirmações duplicadas (duplicate ACKs) pelo transmissor. Na experiência foram capturados pelo Wireshark pacotes que parecem demonstrar este comportamento, como se pode ver pela segunda imagem do Anexo C.6.4.

Na realização da última alínea pudemos verificar que a recepção de dados, quando usada uma segunda ligação no tux2, era afectada. Pode observar-se nos Anexos C.6.5 e C.6.6 que a recepção tende para um plateau máximo no tux1 que é quebrado devido à ligação estabelecida pelo tux2. Observando o gráfico relativo ao tux2 podemos ver que este atinge um plateau máximo perto do final da sua ligação, que ocorre devido ao tux1 já ter terminado o download. Além deste plateau máximo podemos verificar que os gráficos são complementares no sentido em que a soma das funções dos dois (alinhando-os consoante os seus pontos mínimos e máximos dado que as leituras em Wireshark não foram iniciadas em simultâneo) resulta aproximadamente numa função constante que apresenta uma taxa entre 10000 e 12000 packets por segundo. Uma vez que este valor é semelhante ao plateau atingido pelo tux2 é plausível que o servidor esteja limitado a esta taxa.

Detalhes adicionais relativos ao protocolo TCP podem ser consultados no RFC793.

3.7 Experiência 7 - Implementar NAT em Linux

Nesta experiência implementamos NAT no tux4 e geramos diferentes tipos de tráfego para internet. Foram usados os comando wget, traceroute e ping sendo consecutivamente observado o tráfego no tux4.eth0 e no tux4.eth1.

Ante de mais, tivemos que adicionar o IP do tux4.eth1 às permissões do router, uma vez que não fazia parte dos endereços permitidos inicialmente. Ora, com NAT configurada no tux4, o tráfego de qualquer máquina passa para fora como proveniente do tux4, logo foi necessário dar permissões a esta máquina no router.

Verificou-se que usando o NAT no tux4 os endereços IP de destino da camada de Rede nos pacotes TCP recebidos (como resposta aos enviados) variavam consoante a rede em que se encontravam, tal como era esperado ao usar NAT. O encaminhamento só é realizado devido às portas indicadas na camada de transporte (TCP) sendo que o tux4 re-encaminhou para o tux1 pacotes associados à porta 37351 como se pode ver no Anexo C.7.1.

Foram enviados pacotes UDP ao realizar o traceroute. O protocolo UDP não é orientado a ligações e não é fiável ao contrário do TCP. No protocolo UDP não existem confirmações de pacotes nem re-transmissões ou outros mecanismos que garantem a entrega de dados ao destinatário. Esta propriedade pode ser observada no Anexo C.7.2 onde não foi recebida resposta a alguns dos pacotes UDP enviados. O RFC768 apresenta detalhes adicionais relativos ao protocolo UDP.

Foi observada a recepção de pacotes ICMP como resposta aos pacotes de traceroute e de ping. O protocolo ICMP não faz uso de portas como o UDP e TCP sendo que usando NAT só é possível realizar o encaminhamento correctamente devido ao uso de um “identifier"como referido na página 15 do RFC792. O Anexo C.7.3 mostra um par de pacotes ping onde se pode observar que apresentam o mesmo “identifier". Mais detalhes relativos ao uso de ICMP com NAT estão disponíveis na Secção 3 do RFC5508 .

4 Conclusões

O grupo conseguiu realizar todos os exercícios propostos. A realização destes exercícios e a elaboração do relatório visando responder às perguntas do guião ajudaram a sedimentar os conceitos leccionados através de uma metodologia prática e que estimula a reflexão crítica dos estudantes. Foi também possível aprofundar alguns detalhes através de pesquisa autónoma.

A parte que achamos mais difícil no projecto foi perceber e analisar o funcionamento do protocolo TCP.

5 Bibliografia

[1] “Request for comments (rfc).” [Online]. Available: https://www.ietf.org/rfc.html

[2] B. Hall, “Beej’s guide to network programming using internet sockets,” 2015. [Online]. Available: http://beej.us/guide/bgnet/output/html/multipage/index.html

[3] “Wireshark wiki.” [Online]. Available: https://wiki.wireshark.org/

Anexos

A Enderaços MAC

- tux1 eth0: 00:0f:fe:8c:af:71
- tux2 eth0: 00:21:5a:5a:7d:9c
- tux3 eth0: 00:21:5a:61:2f:4e
- tux4 eth0: 00:21:5a:c5:61:bb
- tux4 eth1: 00:c0:df:04:20:8c

B Scripts finais de configuração, excluindo experiência 7

B.1 tux1

```
#!/bin/bash
/etc/init.d/networking restart
ip addr flush dev eth0
ifconfig eth0 up
ifconfig eth0 172.16.60.1/24
route add default gw 172.16.60.254
route -n
echo 1 > /proc/sys/net/ipv4/ip_forward
echo 0 > /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts
```

B.2 tux2

```
#!/bin/bash
/etc/init.d/networking restart
ip addr flush dev eth0
ifconfig eth0 up
ifconfig eth0 172.16.61.1/24
route add -net 172.16.60.0/24 gw 172.16.61.253
route add default gw 172.16.61.254
route -n
echo 1 > /proc/sys/net/ipv4/ip_forward
echo 0 > /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts
echo 1 > /proc/sys/net/ipv4/conf/eth0/accept_redirects
echo 1 > /proc/sys/net/ipv4/conf/all/accept_redirects
```

B.3 tux4

```
#!/bin/bash
/etc/init.d/networking restart
ip addr flush dev eth0
ip addr flush dev eth1
ifconfig eth0 up
ifconfig eth0 172.16.60.254/24
ifconfig eth1 up
ifconfig eth1 172.16.61.253/24
```

```
route add default gw 172.16.61.254
route -n
echo 1 > /proc/sys/net/ipv4/ip_forward
echo 0 > /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts
```

B.4 switch

```
configure terminal
interface fastethernet 0/1
switchport mode access
switchport access vlan 60
interface fastethernet 0/4
switchport mode access
switchport access vlan 60
interface fastethernet 0/2
switchport mode access
switchport access vlan 61
interface fastethernet 0/5
switchport mode access
switchport access vlan 61
interface fastethernet 0/6
switchport mode access
switchport access vlan 61
end
```

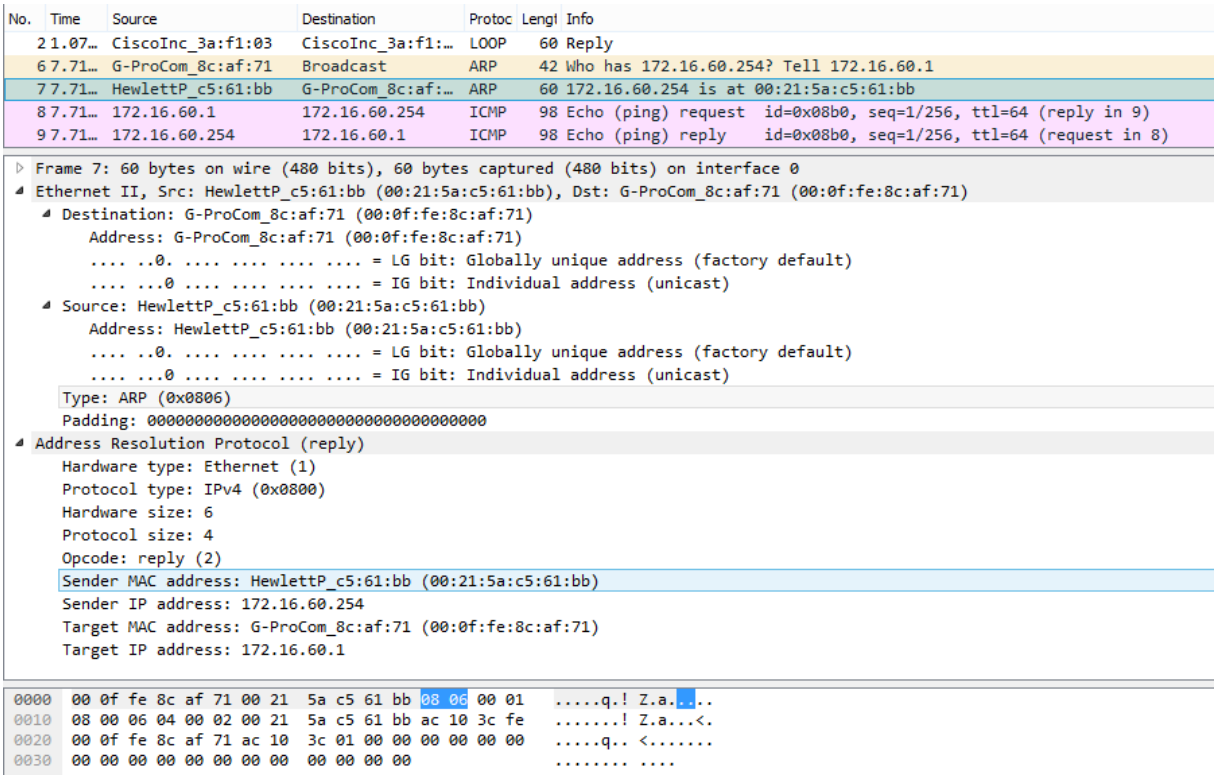
B.5 router

```
conf t
interface gigabitethernet 0/0
ip address 172.16.61.254 255.255.255.0 no shutdown
ip nat inside
exit
interface gigabitethernet 0/1
ip address 172.16.1.69 255.255.255.0 no shutdown
ip nat outside
exit
ip nat pool ovrld 172.16.1.69 172.16.1.69 prefix 24
ip nat inside source list 1 pool ovrld overload
access-list 1 permit 172.16.60.0 0.0.0.7
access-list 1 permit 172.16.61.0 0.0.0.7
ip route 0.0.0.0 0.0.0.0 172.16.1.254
ip route 172.16.60.0 255.255.255.0 172.16.61.253 end
```

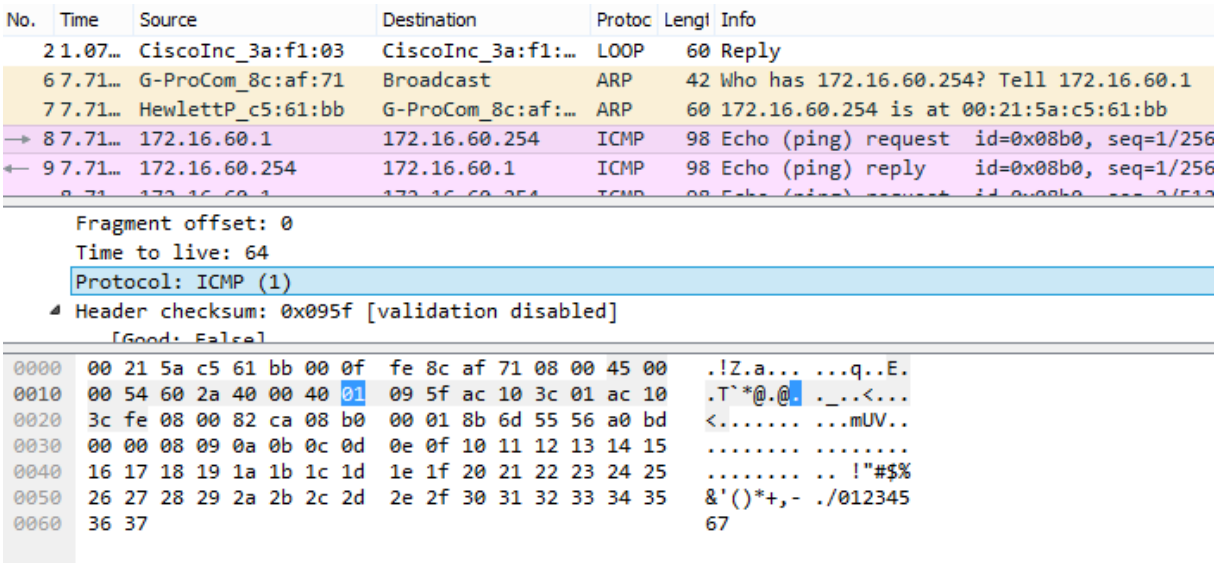
C Logs and statistics

C.1 Experiência 1

C.1.1 Captura Wireshark no tux1 - ARP



C.1.2 Captura Wireshark no tux1 - ICMP



C.2 Experiência 2

C.2.1 Alínea 7 - Captura Wireshark no tux1

No.	Time	Source	Destination	Protoc	Length	Info
5	6.659…	CiscoInc_3a:f1:03	CiscoInc_3a:f1:03	LOOP	60	Reply
	16.66…	CiscoInc_3a:f1:03	CiscoInc_3a:f1:03	LOOP	60	Reply
	22.72…	CiscoInc_3a:f1:03	CDP/VTP/DTP/PAgP/U…	CDP	453	Device ID: tux-sw6 Port ID: FastEthernet0/1
	26.67…	CiscoInc_3a:f1:03	CiscoInc_3a:f1:03	LOOP	60	Reply
	36.67…	CiscoInc_3a:f1:03	CiscoInc_3a:f1:03	LOOP	60	Reply
	37.69…	172.16.60.1	172.16.60.255	ICMP	98	Echo (ping) request id=0x1031, seq=1/256, ttl=64 (no response found!)
	37.69…	172.16.60.254	172.16.60.1	ICMP	98	Echo (ping) reply id=0x1031, seq=1/256, ttl=64
	38.69…	172.16.60.1	172.16.60.255	ICMP	98	Echo (ping) request id=0x1031, seq=2/512, ttl=64 (no response found!)
	38.69…	172.16.60.254	172.16.60.1	ICMP	98	Echo (ping) reply id=0x1031, seq=2/512, ttl=64
	39.69…	172.16.60.1	172.16.60.255	ICMP	98	Echo (ping) request id=0x1031, seq=3/768, ttl=64 (no response found!)
	39.69…	172.16.60.254	172.16.60.1	ICMP	98	Echo (ping) reply id=0x1031, seq=3/768, ttl=64
	40.69…	172.16.60.1	172.16.60.255	ICMP	98	Echo (ping) request id=0x1031, seq=4/1024, ttl=64 (no response found!)
	40.69…	172.16.60.254	172.16.60.1	ICMP	98	Echo (ping) reply id=0x1031, seq=4/1024, ttl=64
	41.69…	172.16.60.1	172.16.60.255	ICMP	98	Echo (ping) request id=0x1031, seq=5/1280, ttl=64 (no response found!)
	41.69…	172.16.60.254	172.16.60.1	ICMP	98	Echo (ping) reply id=0x1031, seq=5/1280, ttl=64
	42.69…	172.16.60.1	172.16.60.255	ICMP	98	Echo (ping) request id=0x1031, seq=6/1536, ttl=64 (no response found!)
	42.69…	172.16.60.254	172.16.60.1	ICMP	98	Echo (ping) reply id=0x1031, seq=6/1536, ttl=64
	42.70…	HewlettP_c5:61:bb	G-ProCom_8c:af:71	ARP	60	Who has 172.16.60.1? Tell 172.16.60.254
	42.70…	G-ProCom_8c:af:71	HewlettP_c5:61:bb	ARP	42	172.16.60.1 is at 00:0f:fe:8c:af:71
	43.69…	172.16.60.1	172.16.60.255	ICMP	98	Echo (ping) request id=0x1031, seq=7/1792, ttl=64 (no response found!)
	43.69…	172.16.60.254	172.16.60.1	ICMP	98	Echo (ping) reply id=0x1031, seq=7/1792, ttl=64

C.2.2 Alínea 7 - Captura Wireshark no tux2

No.	Time	Source	Destination	Protoc	Length	Info
2	0.461…	CiscoInc_3a:f1:04	CDP/VTP/DTP/PAgP/U…	CDP	453	Device
3	0.881…	CiscoInc_3a:f1:04	CiscoInc_3a:f1:04	LOOP	60	Reply
9	10.88…	CiscoInc_3a:f1:04	CiscoInc_3a:f1:04	LOOP	60	Reply
	20.89…	CiscoInc_3a:f1:04	CiscoInc_3a:f1:04	LOOP	60	Reply
	30.89…	CiscoInc_3a:f1:04	CiscoInc_3a:f1:04	LOOP	60	Reply
	40.90…	CiscoInc_3a:f1:04	CiscoInc_3a:f1:04	LOOP	60	Reply
	50.90…	CiscoInc_3a:f1:04	CiscoInc_3a:f1:04	LOOP	60	Reply
	60.46…	CiscoInc_3a:f1:04	CDP/VTP/DTP/PAgP/U…	CDP	453	Device
	60.91…	CiscoInc_3a:f1:04	CiscoInc_3a:f1:04	LOOP	60	Reply

C.2.3 Alínea 7 - Captura Wireshark no tux4

No.	Time	Source	Destination	Protoc	Length	Info
3	2.608…	CiscoInc_3a:f1:06	CiscoInc_3a:f1:06	LOOP	60	Reply
9	12.61…	CiscoInc_3a:f1:06	CiscoInc_3a:f1:06	LOOP	60	Reply
	13.62…	172.16.60.1	172.16.60.255	ICMP	98	Echo (ping) request
	13.62…	172.16.60.254	172.16.60.1	ICMP	98	Echo (ping) reply
	14.62…	172.16.60.1	172.16.60.255	ICMP	98	Echo (ping) request
	14.62…	172.16.60.254	172.16.60.1	ICMP	98	Echo (ping) reply
	15.62…	172.16.60.1	172.16.60.255	ICMP	98	Echo (ping) request
	15.62…	172.16.60.254	172.16.60.1	ICMP	98	Echo (ping) reply
	16.62…	172.16.60.1	172.16.60.255	ICMP	98	Echo (ping) request
	16.62…	172.16.60.254	172.16.60.1	ICMP	98	Echo (ping) reply
	17.62…	172.16.60.1	172.16.60.255	ICMP	98	Echo (ping) request

C.2.4 Alínea10 - Captura Wireshark no tux1

No.	Time	Source	Destination	Protoc	Length	Info
3	2.102…	CiscoInc_3a:f1:03	CiscoInc_3a:f1:03	LOOP	60	Reply
9	12.11…	CiscoInc_3a:f1:03	CiscoInc_3a:f1:03	LOOP	60	Reply
	22.11…	CiscoInc_3a:f1:03	CiscoInc_3a:f1:03	LOOP	60	Reply
	32.11…	CiscoInc_3a:f1:03	CiscoInc_3a:f1:03	LOOP	60	Reply
	37.99…	CiscoInc_3a:f1:03	CDP/VTP/DTP/PAgP/U…	CDP	453	Device ID
	42.11…	CiscoInc_3a:f1:03	CiscoInc_3a:f1:03	LOOP	60	Reply
	52.13…	CiscoInc_3a:f1:03	CiscoInc_3a:f1:03	LOOP	60	Reply
	62.13…	CiscoInc_3a:f1:03	CiscoInc_3a:f1:03	LOOP	60	Reply
	72.13…	CiscoInc_3a:f1:03	CiscoInc_3a:f1:03	LOOP	60	Reply

C.2.5 Alínea10 - Captura Wireshark no tux2

No.	Time	Source	Destination	Protoc	Length	Info
3	2.344…	CiscoInc_3a:f1:04	CiscoInc_3a:f1:04	LOOP	60	Reply
8	11.75…	CiscoInc_3a:f1:04	CDP/VTP/DTP/PAgP/U…	CDP	453	Device ID: tux-sw6 Port ID: FastEthernet0/2
	12.34…	CiscoInc_3a:f1:04	CiscoInc_3a:f1:04	LOOP	60	Reply
	19.31…	172.16.61.1	172.16.61.255	ICMP	98	Echo (ping) request id=0x11b6, seq=1/256, ttl=64 (no response found!)
	20.31…	172.16.61.1	172.16.61.255	ICMP	98	Echo (ping) request id=0x11b6, seq=2/512, ttl=64 (no response found!)
	21.31…	172.16.61.1	172.16.61.255	ICMP	98	Echo (ping) request id=0x11b6, seq=3/768, ttl=64 (no response found!)
	22.31…	172.16.61.1	172.16.61.255	ICMP	98	Echo (ping) request id=0x11b6, seq=4/1024, ttl=64 (no response found!)
	22.35…	CiscoInc_3a:f1:04	CiscoInc_3a:f1:04	LOOP	60	Reply
	23.31…	172.16.61.1	172.16.61.255	ICMP	98	Echo (ping) request id=0x11b6, seq=5/1280, ttl=64 (no response found!)
	24.31…	172.16.61.1	172.16.61.255	ICMP	98	Echo (ping) request id=0x11b6, seq=6/1536, ttl=64 (no response found!)
	25.31…	172.16.61.1	172.16.61.255	ICMP	98	Echo (ping) request id=0x11b6, seq=7/1792, ttl=64 (no response found!)
	26.31…	172.16.61.1	172.16.61.255	ICMP	98	Echo (ping) request id=0x11b6, seq=8/2048, ttl=64 (no response found!)
	27.31…	172.16.61.1	172.16.61.255	ICMP	98	Echo (ping) request id=0x11b6, seq=9/2304, ttl=64 (no response found!)

C.2.6 Alínea10 - Captura Wireshark no tux4

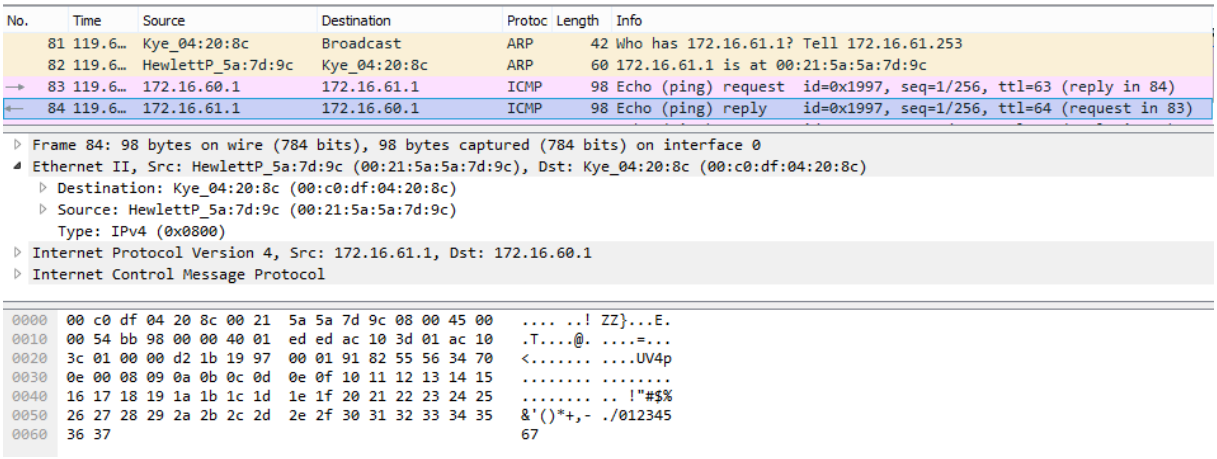
No.	Time	Source	Destination	Protoc	Length	Info
3	2.085…	CiscoInc_3a:f1:06	CiscoInc_3a:f1:06	LOOP	60	Reply
9	12.08…	CiscoInc_3a:f1:06	CiscoInc_3a:f1:06	LOOP	60	Reply
	22.08…	CiscoInc_3a:f1:06	CiscoInc_3a:f1:06	LOOP	60	Reply
	23.71…	CiscoInc_3a:f1:06	CDP/VTP/DTP/PAgP/U…	CDP	453	Device
	32.09…	CiscoInc_3a:f1:06	CiscoInc_3a:f1:06	LOOP	60	Reply
	42.10…	CiscoInc_3a:f1:06	CiscoInc_3a:f1:06	LOOP	60	Reply
	52.11…	CiscoInc_3a:f1:06	CiscoInc_3a:f1:06	LOOP	60	Reply
	62.10…	CiscoInc_3a:f1:06	CiscoInc_3a:f1:06	LOOP	60	Reply
	72.11…	CiscoInc_3a:f1:06	CiscoInc_3a:f1:06	LOOP	60	Reply

C.3 Experiência 3

C.3.1 Capturas Wireshark no tux4.eth0

No.	Time	Source	Destination	Protoc	Length	Info
103	139.3…	G-ProCom_8c:af:71	Broadcast	ARP	60	Who has 172.16.60.254? Tell 172.16.60.1
104	139.3…	HewlettP_c5:61:bb	G-ProCom_8c:af:71	ARP	42	172.16.60.254 is at 00:21:5a:c5:61:bb
105	139.3…	172.16.60.1	172.16.61.1	ICMP	98	Echo (ping) request id=0x1997, seq=1/256,
106	139.3…	172.16.61.1	172.16.60.1	ICMP	98	Echo (ping) reply id=0x1997, seq=1/256,
107	140.3…	172.16.60.1	172.16.61.1	ICMP	98	Echo (ping) request id=0x1997, seq=2/512,
108	140.3…	172.16.61.1	172.16.60.1	ICMP	98	Echo (ping) reply id=0x1997, seq=2/512,

C.3.2 Capturas Wireshark no tux4.eth1



```
No.  Time     Source          Destination     Protoc Length Info
  81 119.6…  Kye_04:20:8c     Broadcast       ARP        42 Who has 172.16.61.1? Tell 172.16.61.253
  82 119.6…  HewlettP_5a:7d:9c Kye_04:20:8c    ARP        60 172.16.61.1 is at 00:21:5a:5a:7d:9c
  83 119.6…  172.16.60.1      172.16.61.1     ICMP       98 Echo (ping) request  id=0x1997, seq=1/256, ttl=63 (reply in 84)
  84 119.6…  172.16.61.1      172.16.60.1     ICMP       98 Echo (ping) reply    id=0x1997, seq=1/256, ttl=64 (request in 83)

▷ Frame 84: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface 0
◢ Ethernet II, Src: HewlettP_5a:7d:9c (00:21:5a:5a:7d:9c), Dst: Kye_04:20:8c (00:c0:df:04:20:8c)
  ▷ Destination: Kye_04:20:8c (00:c0:df:04:20:8c)
  ▷ Source: HewlettP_5a:7d:9c (00:21:5a:5a:7d:9c)
    Type: IPv4 (0x0800)
▷ Internet Protocol Version 4, Src: 172.16.61.1, Dst: 172.16.60.1
▷ Internet Control Message Protocol

0000  00 c0 df 04 20 8c 00 21  5a 5a 7d 9c 08 00 45 00   .... ..! ZZ}...E.
0010  00 54 bb 98 00 00 40 01  ed ed ac 10 3d 01 ac 10   .T....@. ....=...
0020  3c 01 00 00 d2 1b 19 97  00 01 91 82 55 56 34 70   <....... ....UV4p
0030  0e 00 08 09 0a 0b 0c 0d  0e 0f 10 11 12 13 14 15   ........ ........
0040  16 17 18 19 1a 1b 1c 1d  1e 1f 20 21 22 23 24 25   ........ .. !"#$%
0050  26 27 28 29 2a 2b 2c 2d  2e 2f 30 31 32 33 34 35   &'()*+,- ./012345
0060  36 37                                             67
```

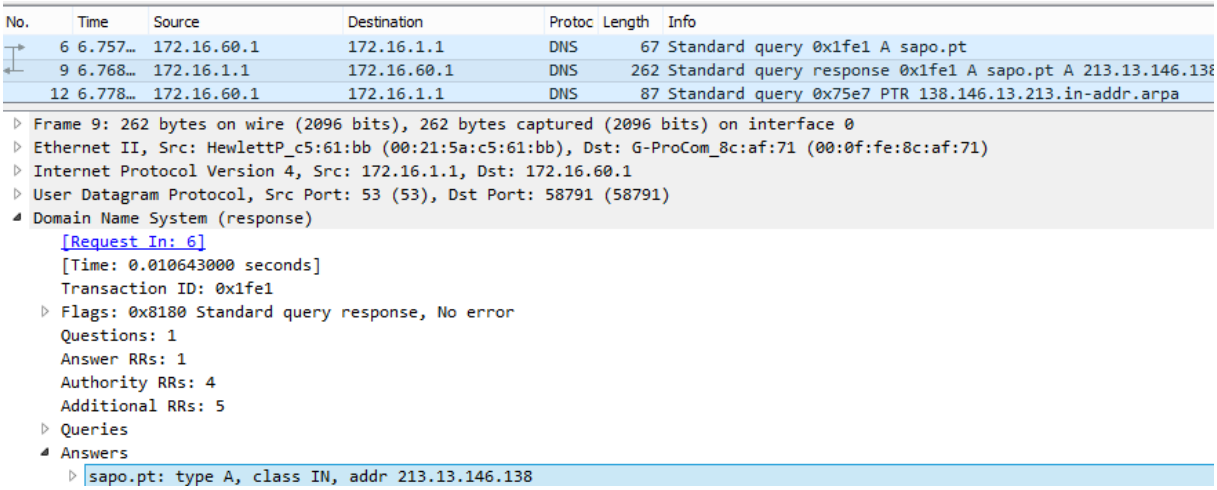
C.4 Experiência 4

C.4.1 Alínea 4 Console Log

```
tux63:~/Desktop/RCOM/scripts# route -n
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         172.16.61.254   0.0.0.0         UG    0      0        0 eth0
172.16.60.0     172.16.61.253   255.255.255.0   UG    0      0        0 eth0
172.16.61.0     172.16.61.254   255.255.255.0   UG    0      0        0 eth0
172.16.61.0     0.0.0.0         255.255.255.0   U     0      0        0 eth0
tux63:~/Desktop/RCOM/scripts# route del -net 172.16.60.0/24 gw 172.16.61.253
tux63:~/Desktop/RCOM/scripts# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         172.16.61.254   0.0.0.0         UG    0      0        0 eth0
172.16.61.0     172.16.61.254   255.255.255.0   UG    0      0        0 eth0
172.16.61.0     0.0.0.0         255.255.255.0   U     0      0        0 eth0
tux63:~/Desktop/RCOM/scripts# traceroute 172.16.60.1
traceroute to 172.16.60.1 (172.16.60.1), 30 hops max, 60 byte packets
 1  172.16.61.254 (172.16.61.254)  0.498 ms  0.548 ms  0.587 ms
 2  172.16.61.253 (172.16.61.253)  0.873 ms  0.500 ms  0.506 ms
 3  172.16.60.1 (172.16.60.1)  0.799 ms  0.792 ms  0.784 ms
tux63:~/Desktop/RCOM/scripts# ping 172.16.60.1
PING 172.16.60.1 (172.16.60.1) 56(84) bytes of data.
64 bytes from 172.16.60.1: icmp_seq=1 ttl=62 time=0.629 ms
64 bytes from 172.16.60.1: icmp_seq=2 ttl=62 time=0.594 ms
64 bytes from 172.16.60.1: icmp_seq=3 ttl=62 time=0.587 ms
64 bytes from 172.16.60.1: icmp_seq=4 ttl=62 time=0.569 ms
64 bytes from 172.16.60.1: icmp_seq=5 ttl=62 time=0.623 ms
^C
--- 172.16.60.1 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4000ms
rtt min/avg/max/mdev = 0.569/0.600/0.629/0.031 ms
tux63:~/Desktop/RCOM/scripts# traceroute 172.16.60.1
traceroute to 172.16.60.1 (172.16.60.1), 30 hops max, 60 byte packets
 1  172.16.61.253 (172.16.61.253)  0.465 ms  0.343 ms  0.344 ms
 2  172.16.60.1 (172.16.60.1)  0.666 ms  0.662 ms  0.654 ms
tux63:~/Desktop/RCOM/scripts# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         172.16.61.254   0.0.0.0         UG    0      0        0 eth0
172.16.61.0     172.16.61.254   255.255.255.0   UG    0      0        0 eth0
172.16.61.0     0.0.0.0         255.255.255.0   U     0      0        0 eth0
tux63:~/Desktop/RCOM/scripts#
```

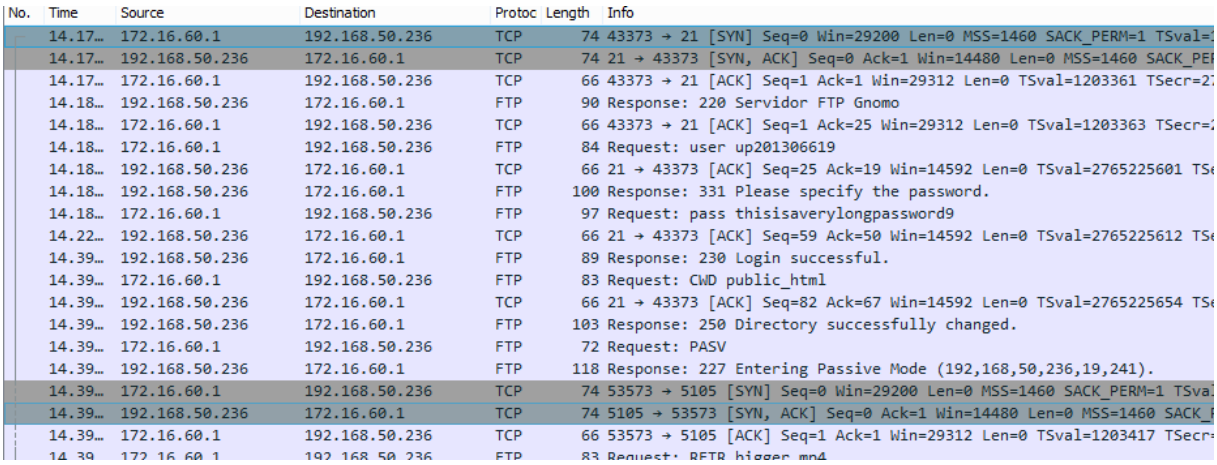
C.5 Experiência 5

C.5.1 Capturas Wireshark de DNS no tux1

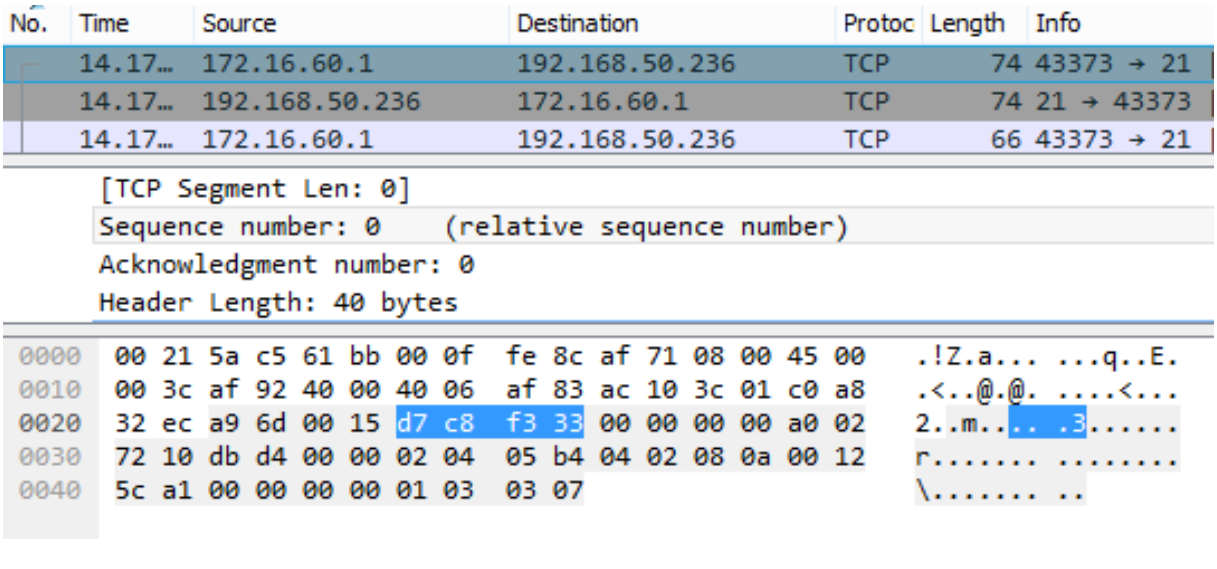


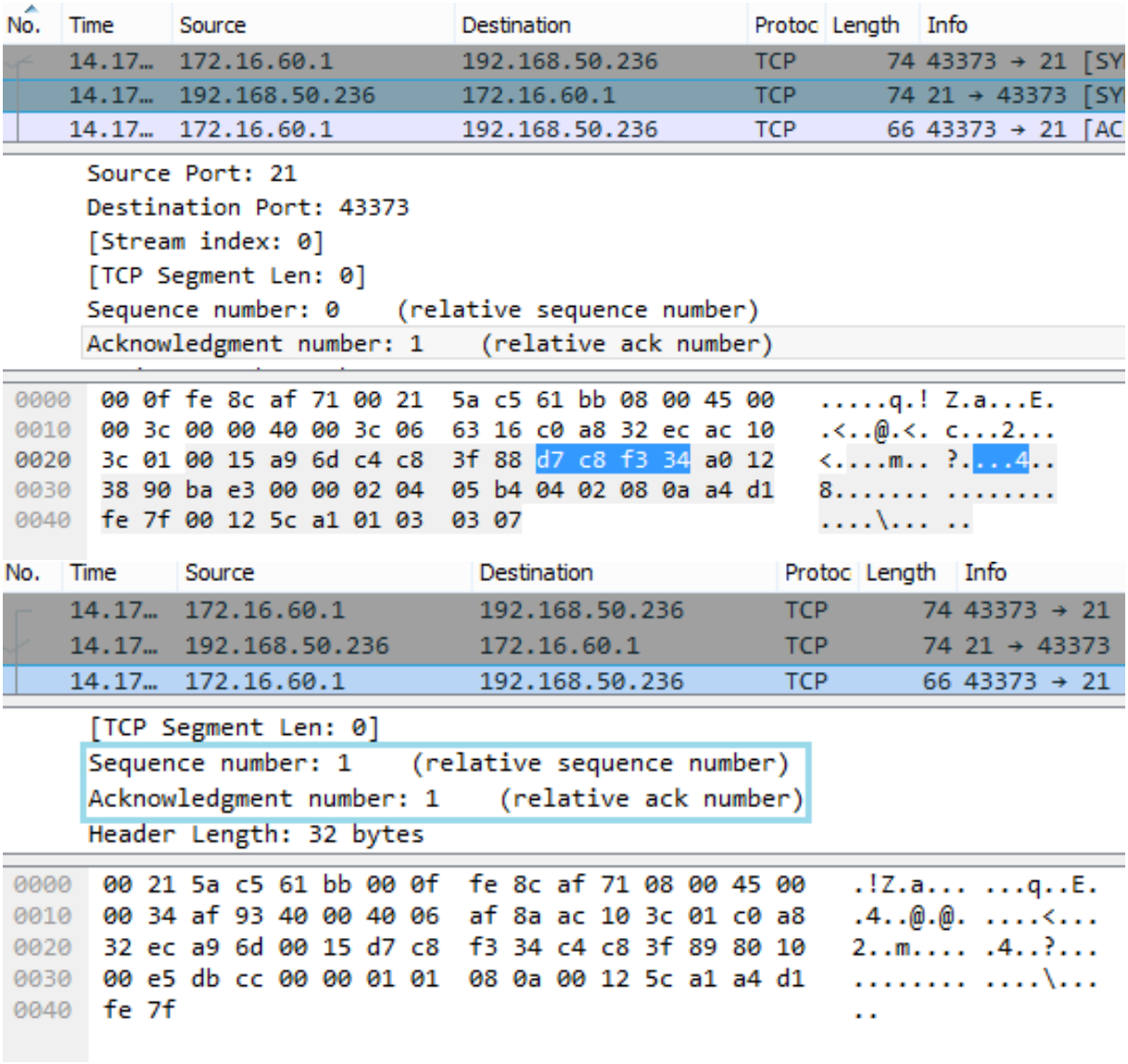
C.6 Experiência 6

C.6.1 Capturas Wireshark dos 'hanshakes' no tux1

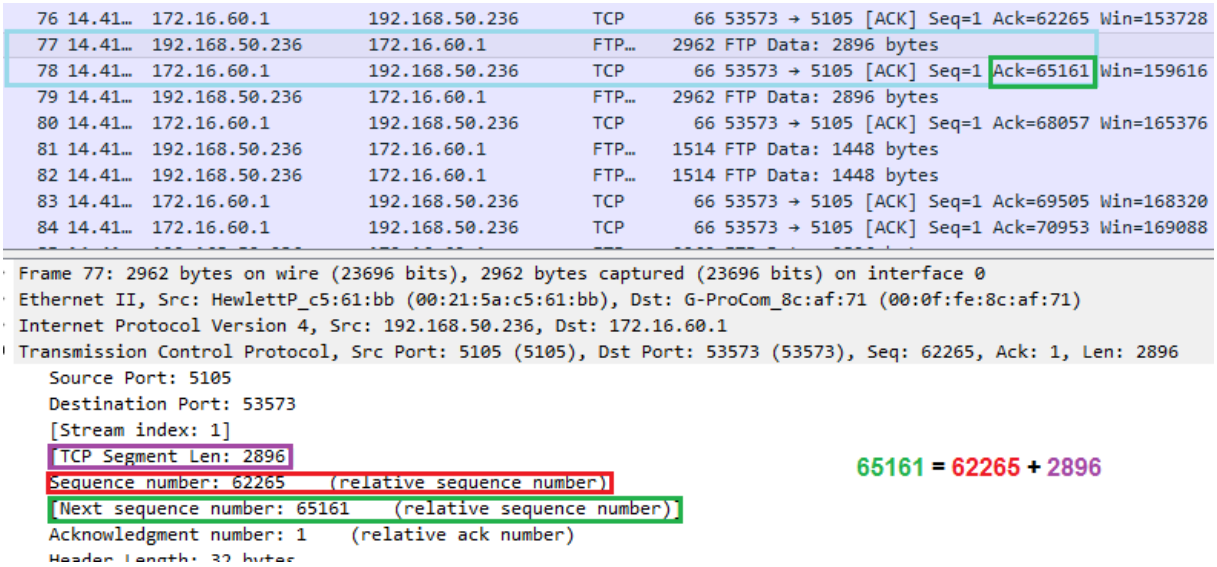


C.6.2 Primeiro 'Handshake' no tux1 em Detalhe

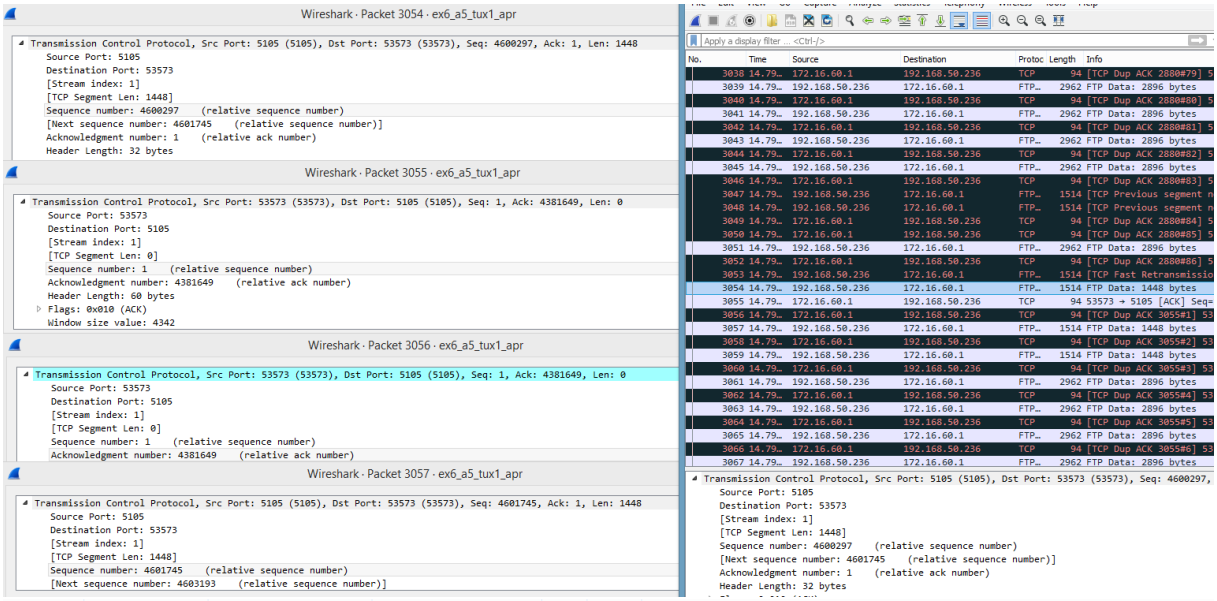




C.6.3 Capturas Wireshark no tux1 de ACKs



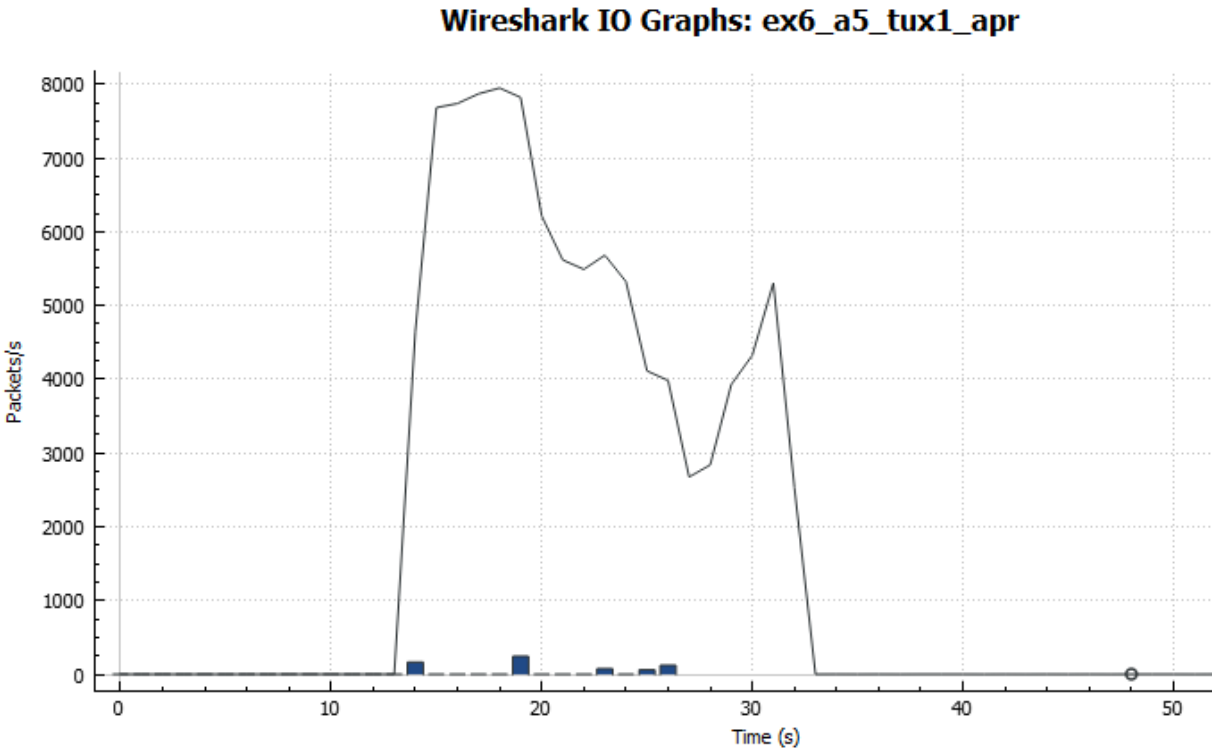
C.6.4 Capturas Wireshark no tux1 de Dup ACK, Fast Retransmission e Retransmission



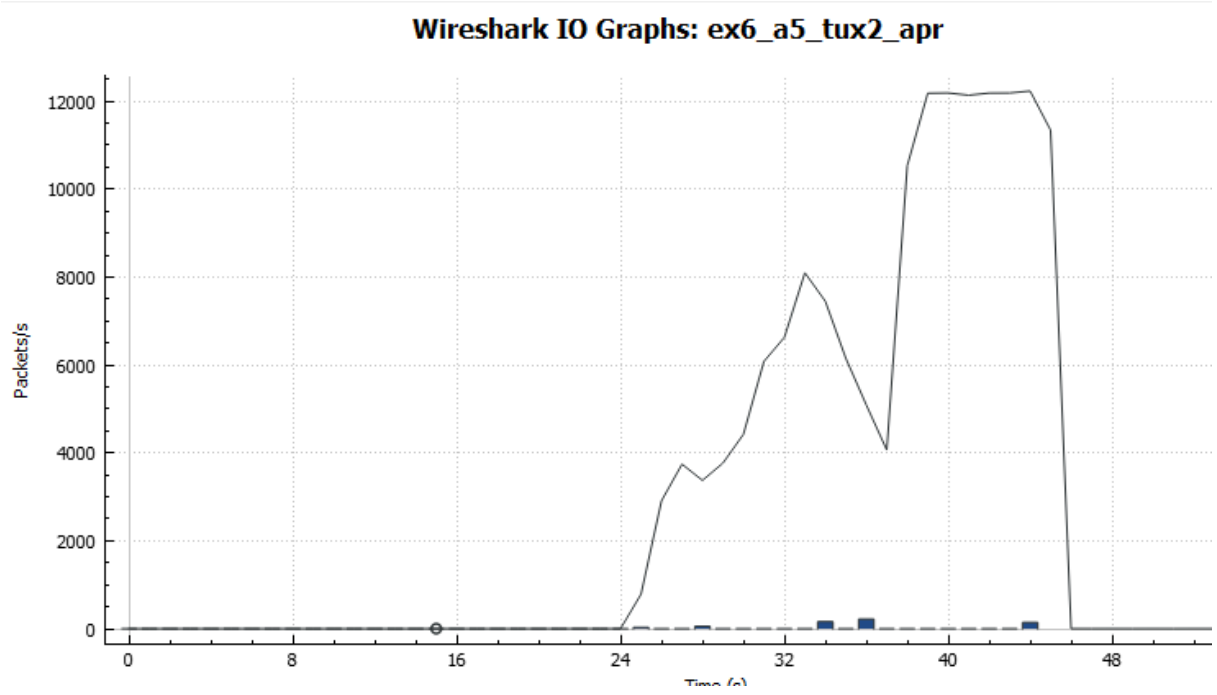
No.	Time	Source	Destination	Protoc	Length	Info
3056	14.797013	172.16.60.1	192.168.50.236	TCP	94	[TCP Dup ACK 3055#1] 53573 → 5105 [ACK] Seq=1 Ack=4381649 Win=555776 L
3058	14.797242	172.16.60.1	192.168.50.236	TCP	94	[TCP Dup ACK 3055#2] 53573 → 5105 [ACK] Seq=1 Ack=4381649 Win=555776 L
3060	14.797252	172.16.60.1	192.168.50.236	TCP	94	[TCP Dup ACK 3055#3] 53573 → 5105 [ACK] Seq=1 Ack=4381649 Win=555776 L
3062	14.797504	172.16.60.1	192.168.50.236	TCP	94	[TCP Dup ACK 3055#4] 53573 → 5105 [ACK] Seq=1 Ack=4381649 Win=555776 L
3064	14.797750	172.16.60.1	192.168.50.236	TCP	94	[TCP Dup ACK 3055#5] 53573 → 5105 [ACK] Seq=1 Ack=4381649 Win=555776 L
3066	14.797999	172.16.60.1	192.168.50.236	TCP	94	[TCP Dup ACK 3055#6] 53573 → 5105 [ACK] Seq=1 Ack=4381649 Win=555776 L
3068	14.798249	172.16.60.1	192.168.50.236	TCP	94	[TCP Dup ACK 3055#7] 53573 → 5105 [ACK] Seq=1 Ack=4381649 Win=555776 L
3070	14.798500	172.16.60.1	192.168.50.236	TCP	94	[TCP Dup ACK 3055#8] 53573 → 5105 [ACK] Seq=1 Ack=4381649 Win=555776 L
3072	14.798748	172.16.60.1	192.168.50.236	TCP	94	[TCP Dup ACK 3055#9] 53573 → 5105 [ACK] Seq=1 Ack=4381649 Win=555776 L
3074	14.798998	172.16.60.1	192.168.50.236	TCP	94	[TCP Dup ACK 3055#10] 53573 → 5105 [ACK] Seq=1 Ack=4381649 Win=555776
3076	14.799250	172.16.60.1	192.168.50.236	TCP	94	[TCP Dup ACK 3055#11] 53573 → 5105 [ACK] Seq=1 Ack=4381649 Win=555776
3078	14.799498	172.16.60.1	192.168.50.236	TCP	94	[TCP Dup ACK 3055#12] 53573 → 5105 [ACK] Seq=1 Ack=4381649 Win=555776
3080	14.799737	192.168.50.236	172.16.60.1	FTP…	1514	[TCP Fast Retransmission] FTP Data: 1448 bytes
3081	14.799750	172.16.60.1	192.168.50.236	TCP	94	[TCP Dup ACK 3055#13] 53573 → 5105 [ACK] Seq=1 Ack=4381649 Win=555776

No.	Time	Source	Destination	Protoc	Length	Info
3080	14.79…	192.168.50.236	172.16.60.1	FTP…	1514	[TCP Fast Retransmission] FTP Data: 1448 bytes
3081	14.79…	172.16.60.1	192.168.50.236	TCP	94	[TCP Dup ACK 3055#13] 53573 → 5105 [ACK] Seq=1 Ack=4381649
3082	14.79…	172.16.60.1	192.168.50.236	TCP	94	53573 → 5105 [ACK] Seq=1 Ack=4384545 Win=552960 Len=0 TSval
3083	14.79…	192.168.50.236	172.16.60.1	TCP	2962	[TCP Out-Of-Order] 5105 → 53573 [ACK] Seq=4384545 Ack=1 Win
3084	14.80…	172.16.60.1	192.168.50.236	TCP	94	53573 → 5105 [ACK] Seq=1 Ack=4387441 Win=550144 Len=0 TSval
3085	14.80…	192.168.50.236	172.16.60.1	TCP	1514	[TCP Out-Of-Order] 5105 → 53573 [ACK] Seq=4387441 Ack=1 Win
3086	14.80…	192.168.50.236	172.16.60.1	TCP	1514	[TCP Out-Of-Order] 5105 → 53573 [ACK] Seq=4390337 Ack=1 Win
3087	14.80…	172.16.60.1	192.168.50.236	TCP	94	53573 → 5105 [ACK] Seq=1 Ack=4390337 Win=547328 Len=0 TSval
3088	14.80…	172.16.60.1	192.168.50.236	TCP	94	53573 → 5105 [ACK] Seq=1 Ack=4391785 Win=545920 Len=0 TSval
3089	14.80…	192.168.50.236	172.16.60.1	TCP	2962	[TCP Out-Of-Order] 5105 → 53573 [ACK] Seq=4391785 Ack=1 Win
3090	14.80…	172.16.60.1	192.168.50.236	TCP	94	53573 → 5105 [ACK] Seq=1 Ack=4396129 Win=541696 Len=0 TSval
3091	14.80…	192.168.50.236	172.16.60.1	TCP	2962	[TCP Out-Of-Order] 5105 → 53573 [ACK] Seq=4396129 Ack=1 Win
3092	14.80…	172.16.60.1	192.168.50.236	TCP	94	53573 → 5105 [ACK] Seq=1 Ack=4399025 Win=538880 Len=0 TSval
3093	14.80…	192.168.50.236	172.16.60.1	TCP	1514	[TCP Retransmission] 5105 → 53573 [ACK] Seq=4399025 Ack=1 W

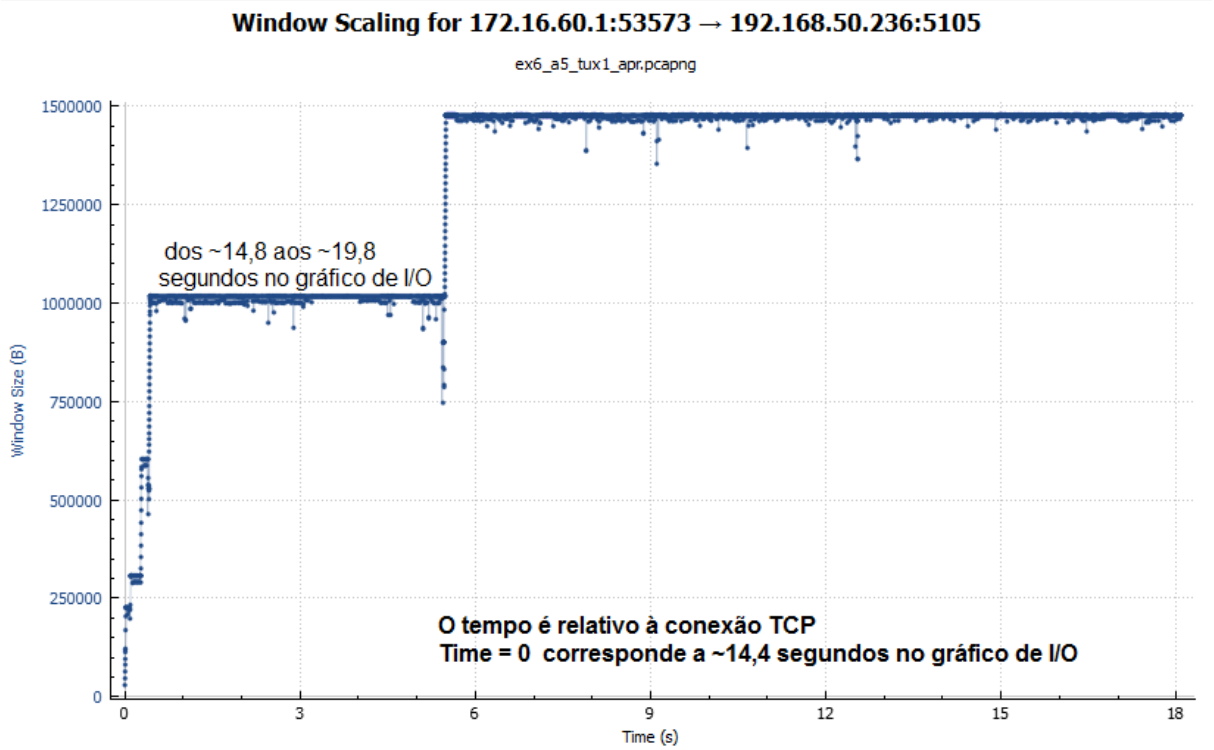
C.6.5 Gráfico de Tráfego no tux1



C.6.6 Gráfico de Tráfego no tux2

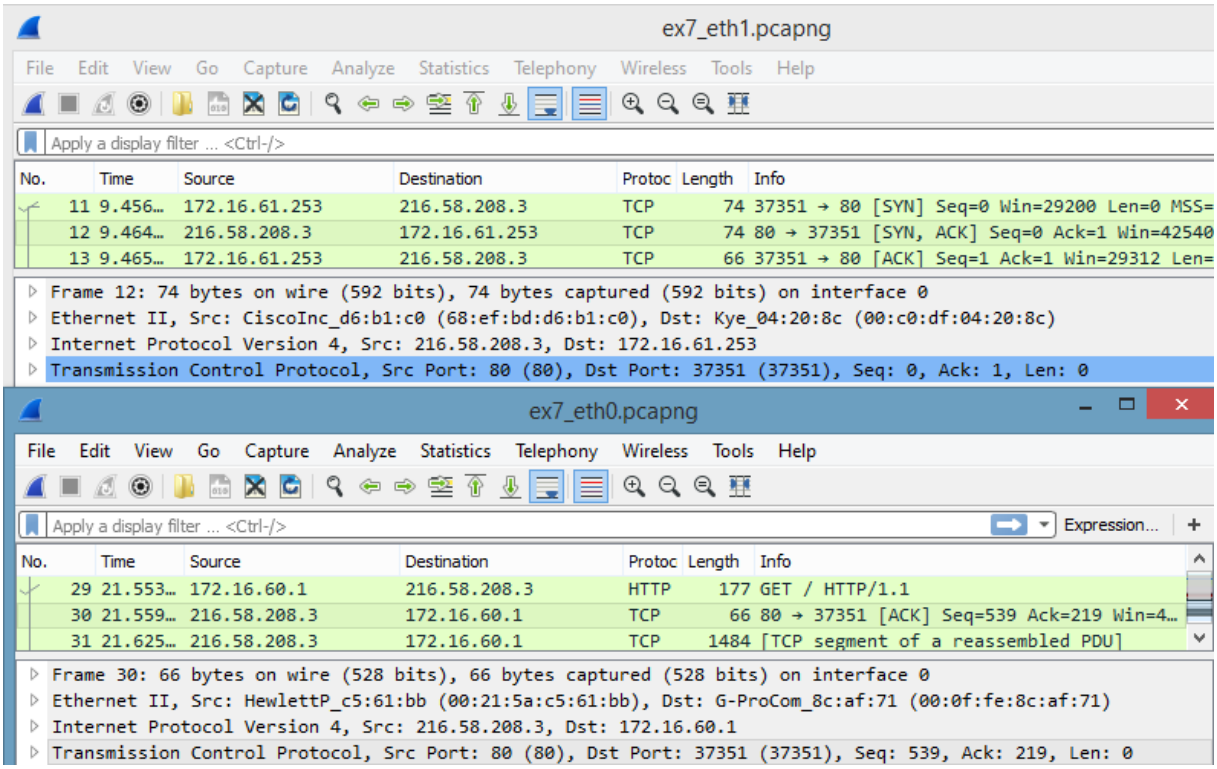


C.6.7 Gráfico de Window Size no tux1 de pacotes TCP recebidos na porta de dados

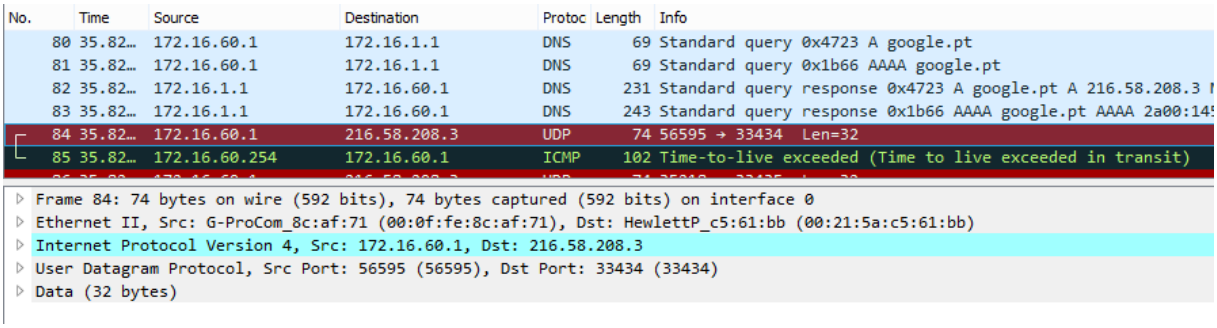


C.7 Experiência 7

C.7.1 Capturas Wireshark TCP no tux4.eth0 e tux4.eth1



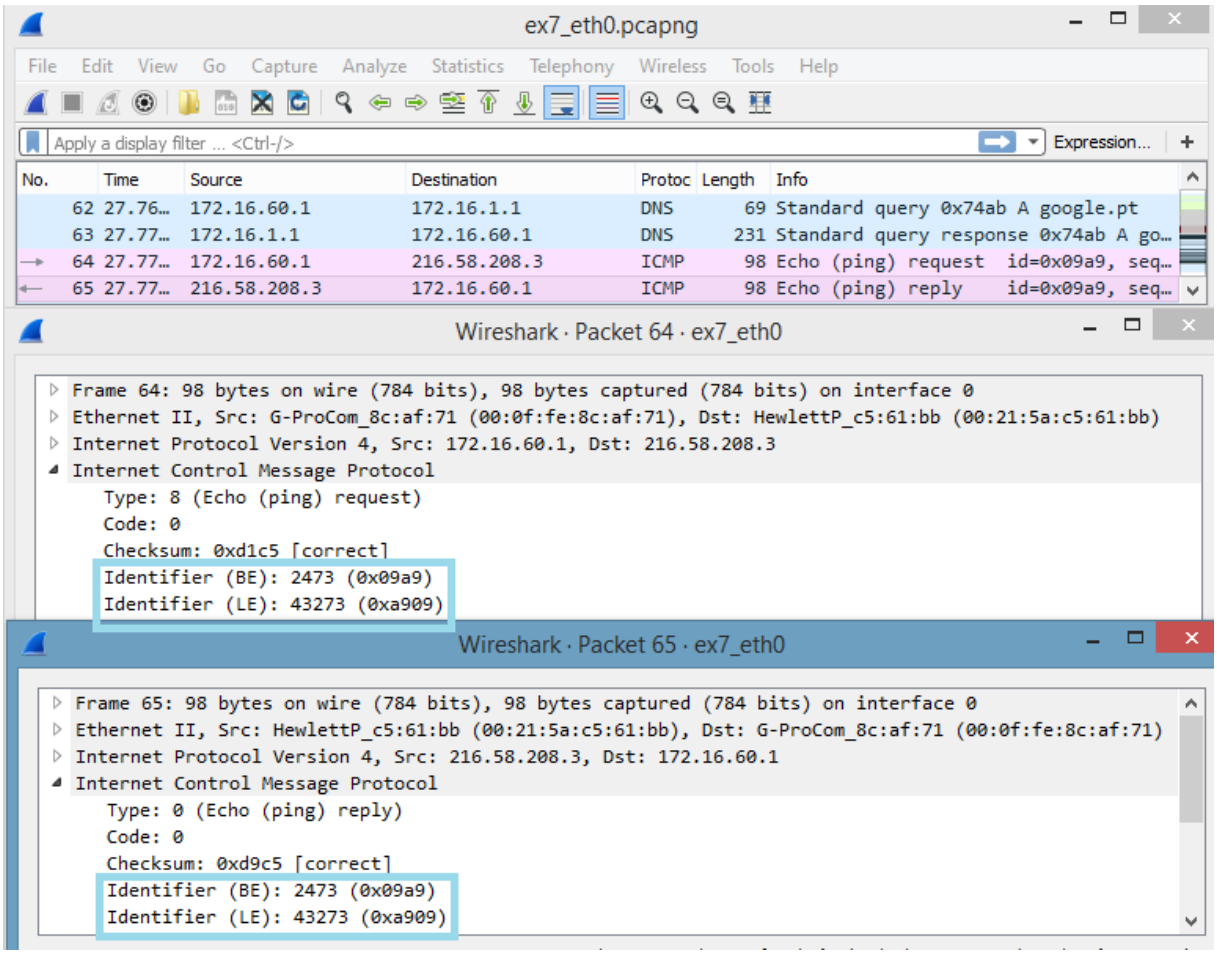
C.7.2 Captura Wireshark UDP e timeout no tux4.eth0



No.	Time	Source	Destination	Protoc	Length	Info
80	35.82…	172.16.60.1	172.16.1.1	DNS	69	Standard query 0x4723 A google.pt
81	35.82…	172.16.60.1	172.16.1.1	DNS	69	Standard query 0x1b66 AAAA google.pt
82	35.82…	172.16.1.1	172.16.60.1	DNS	231	Standard query response 0x4723 A google.pt A 216.58.208.3
83	35.82…	172.16.1.1	172.16.60.1	DNS	243	Standard query response 0x1b66 AAAA google.pt AAAA 2a00:145
84	35.82…	172.16.60.1	216.58.208.3	UDP	74	56595 → 33434 Len=32
85	35.82…	172.16.60.254	172.16.60.1	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)

```
Frame 84: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface 0
Ethernet II, Src: G-ProCom_8c:af:71 (00:0f:fe:8c:af:71), Dst: HewlettP_c5:61:bb (00:21:5a:c5:61:bb)
Internet Protocol Version 4, Src: 172.16.60.1, Dst: 216.58.208.3
User Datagram Protocol, Src Port: 56595 (56595), Dst Port: 33434 (33434)
Data (32 bytes)
```

C.7.3 Capturas Wireshark TCP no tux4.eth0



D Código Fonte

D.1 Ficheiro downloader.c

```
1
2  #include <string.h>
3  #include <sys/socket.h>
4  #include <netinet/in.h>
5  #include <arpa/inet.h>
6  #include <netdb.h>
7  #include <stdlib.h>
8  #include <unistd.h>
9  #include <stdio.h>
10 #include <string.h>
11 #include <errno.h>
12 #include <sys/types.h>
13 #include "utilities.h"
14 #include "ftp.h"
15
16 //VARS AND STRUCTS
     ---------------------------------------------------------------------------

17 #define FTP_PORT    21
18 #define MAX_STRING_SIZE 200
19 struct /*???*/Info{
20   char username[MAX_STRING_SIZE];
21   char password[MAX_STRING_SIZE];
22   char host_name[MAX_STRING_SIZE];
23   char url_path[MAX_STRING_SIZE];
24   char filename[MAX_STRING_SIZE];
25   char ip[MAX_STRING_SIZE];
26 };
27
28 //AUX FUNCS CODE
     ---------------------------------------------------------------------------

29 int parse(char *str, struct Info* info) {
30
31   //http://docs.roxen.com/pike/7.0/tutorial/strings/sscanf.xml
32     if(4 != sscanf(str, "ftp://[%[^:]:%[^@]@]%[^/]/%s\n", info->
         username, info->password, info->host_name, info->url_path)) {
33       return 1;
34     }
35
36   //get filename http://stackoverflow.com/questions/32822988/get-the-
       last-token-of-a-string-in-c
37       char *last = strrchr(info->url_path, '/') ;
38       if(last!=NULL)
39       {
40     memcpy(info->filename, last+1, strlen(last)+1);
41     memset(last,0,strlen(last)+1);
42       }
43       else {
44     strcpy(info->filename,info->url_path);
45     memset(info->url_path,0,sizeof(info->url_path));
46       }
47
48     return 0;
49 }
```

```
50
51  int get_ip(struct Info* info) {
52      struct hostent* host;
53
54      if ((host = gethostbyname(info->host_name)) == NULL) {
55          perror("gethostbyname");
56          return 1;
57      }
58
59      char* ip = inet_ntoa(*((struct in_addr *)host->h_addr));
60      strcpy(info->ip, ip);
61
62      printf("Host name  : %s\n", host->h_name);
63      printf("IP Address : %s\n", info->ip);
64
65      return 0;
66  }
67
68
69  //MAIN
    ------------------------------------------------------------------------------
70  #define DEBUG_ALL 1
71  int main(int argc,char **argv)
72  {
73    struct Info info;
74
75    // ftp message composition: ftp://[<user>:<password>@]<host>/<url-
       path>
76
77      // ---- URL stuff ----
78
79      //parse
80      if(parse(argv[1],&info)!=OK)
81      {
82        printf("\nINVALID ARGUMENT! couldn't be parsed properly.\n");
83        return 1;
84      }
85      DEBUG_SECTION(DEBUG_ALL,
86      printf("\nuser:%s\n",info.username);
87      printf("pass:%s\n",info.password);
88      printf("host:%s\n",info.host_name);
89      printf("urlpath:%s\n",info.url_path);
90      printf("filename:%s\n",info.filename);
91      );
92
93      //- - - - - -
94      get_ip(&info);
95
96      // ---- FTP stuff -----
97
98  printf("\n connecting... \n");
99
100     if(ftp_connect(info.ip, FTP_PORT)!=OK)
101 {ftp_abort(); return 1;}
102
103 printf("\n logging in... \n");
104
105     if(ftp_login(info.username, info.password)!=OK)// Send user n pass
106 {ftp_abort(); return 1;}
107
108
109
110     if(strlen(info.url_path)>0) {
111       printf("\n changing dir... \n");
```

```
112
113         if(ftp_changedir(info.url_path)!=OK)// change directory
114         {ftp_abort(); return 1;}
115       }
116
117 printf("\n passive mode... \n");
118
119       if(ftp_pasv()!=OK)// passive mode
120 {ftp_abort(); return 1;}
121
122 printf("\n asking for file... \n");
123
124       if(ftp_retr(info.filename)!=OK)// ask to receive file
125 {ftp_abort(); return 1;}
126
127 printf("\n downloading file... \n");
128
129       if(ftp_download(info.filename)!=OK)// receive file
130 {ftp_abort(); return 1;}
131
132 printf("\n disconecting... \n");
133
134       if(ftp_disconnect()!=OK)// disconnect from server
135 {ftp_abort(); return 1;}
136
137 printf("\n downloader terminated ok! \n");
138
139       return 0;
140 }
```

D.2 Ficheiro ftp.h

```
1
2  #ifndef FTP
3  #define FTP
4
5  int ftp_connect( const char* ip, int port);
6  int ftp_disconnect();
7
8  int ftp_login( const char* user, const char* password);
9  int ftp_changedir( const char* path);
10 int ftp_pasv();
11 int ftp_retr( const char* filename);
12 int ftp_download( const char* filename);
13
14 void ftp_abort();
15
16 #endif
```

D.3 Ficheiro ftp.c

```
1
2  #include <stdio.h>
3  #include <unistd.h>
4  #include <string.h>
5
6  #include <sys/types.h>
7  #include <sys/socket.h>
8
9
10 #include "ftp.h"
11 #include "socket.h"
12 #include "utilities.h"
13
```

```
14 #define MAX_STRING_SIZE 500
15
16 int control_socket_fd;
17 int data_socket_fd;
18
19
20 //
    ------------------------------------------------------------------------

21 // READ AND SEND
22 #if 1
23
24 int ftp_read(char* str,unsigned long str_total_size)
25 {
26     int bytes = 0;
27     if( (bytes = recv(control_socket_fd,str,str_total_size,0)) < 0  )
28       {
29     perror("ftp_read: recv failed\n");
30     return -1;
31       }
32     return bytes;
33 }
34
35 int ftp_send( const char* str,unsigned long str_size)
36 {
37         int bytes = 0;
38     if( (bytes = send(control_socket_fd,str,str_size,0)) < 0  )
39       {
40     perror("ftp_read: recv failed\n");
41     return -1;
42       }
43     return bytes;
44 }
45
46 #endif
47
48
49 //
    ------------------------------------------------------------------------

50 // CONECT AND DISCONECT
51 #if 1
52
53 int ftp_connect( const char* ip, int port) {
54
55     int socket_fd;
56     char read_bytes[MAX_STRING_SIZE];
57
58     //open control socket
59     if ((socket_fd = connect_socket_TCP(ip, port)) < 0)
60     {
61         printf("ftp_connect: Failed to connect socket\n");
62         return 1;
63     }
64
65     control_socket_fd = socket_fd;
66     data_socket_fd    = 0;
67
68     //Try to read with control socket
69     if (ftp_read(read_bytes, sizeof(read_bytes))<0)
70     {
71         printf("ftp_connect: Failed to read\n");
72         return 1;
73     }
74
```

```
75      return 0;
76  }
77
78  int ftp_disconnect() {
79      char aux[MAX_STRING_SIZE];
80
81      //read discnnect
82          if (ftp_read(aux, sizeof(aux))<0) {
83          printf("ftp_disconnect: Failed to disconnect\n");
84          return 1;
85      }
86      //send disconnect
87      sprintf(aux, "QUIT\r\n");
88      if (ftp_send(aux, strlen(aux))<0) {
89          printf("ftp_disconnect: Failed to output QUIT");
90          return 1;
91      }
92
93      close(control_socket_fd);
94
95      return 0;
96  }
97
98  #endif
99
100 //
    ------------------------------------------------------------------------

101 // MAIN OPERATIONS
102 #if 1
103
104 int ftp_login( const char* user, const char* password) {
105
106     char aux[MAX_STRING_SIZE];
107
108     //send username
109     sprintf(aux, "user %s\r\n", user);
110     if (ftp_send( aux, strlen(aux))< 0) {
111         printf("ftp_login: ftp_send failure.\n");
112         return 1;
113     }
114     //receive answer to username
115     if (ftp_read( aux, sizeof(aux))<0) {
116         printf( "ftp_login:Bad response to user\n");
117         return 1;
118     }
119
120     //send password
121     memset(aux, 0, sizeof(aux));//reuse 2send
122     sprintf(aux, "pass %s\r\n", password);
123     if (ftp_send( aux, strlen(aux))< 0) {
124         printf("ftp_login: failed to send password.\n");
125         return 1;
126     }
127     //receive answer to password
128     if (ftp_read( aux, sizeof(aux))<0)
129     {
130         printf( "ftp_login:Bad response to pass\n");
131         return 1;
132     }
133
134     return 0;
135 }
136
137 int ftp_changedir(const char* path) {
```

```
138
139     char aux[MAX_STRING_SIZE];
140
141     //send cwd command
142     sprintf(aux, "CWD %s\r\n", path);
143     if (ftp_send(aux, strlen(aux))< 0) {
144         printf("ftp_changedir:Failed to send\n");
145         return 1;
146     }
147
148     //get response
149     if (ftp_read(aux, sizeof(aux))< 0) {
150         printf("ftp_changedir:Failed to get a valid response\n");
151         return 1;
152     }
153
154     return 0;
155 }
156
157 #define DEBUG_PASV 1
158 int ftp_pasv() {
159
160     char aux[MAX_STRING_SIZE] = "PASV\r\n";
161
162     //send pasv msg
163     if (ftp_send(aux, strlen(aux))< 0) {
164         printf("ftp_pasv: Failed to enter in passive mode\n");
165         return 1;
166     }
167
168     //receive response
169     if (ftp_read(aux, sizeof(aux))<0) {
170         printf("ftp_pasv: Failed to receive information to enter
               passive mode\n");
171         return 1;
172     }
173
174         DEBUG_SECTION(DEBUG_PASV,printf("pasv():received:%s\n",aux);
175     );
176
177     // info was received. scan it
178     int ip_bytes[4];
179     int ports[2];
180
181     if ((sscanf(aux, "%*[^(](%d,%d,%d,%d,%d,%d)",
182     ip_bytes,&ip_bytes[1], &ip_bytes[2], &ip_bytes[3], ports, &ports
           [1]))
183         !=6 )
184     {
185         printf("ftp_pasv: Cannot process received data, must receive 6
               bytes\n");
186         return 1;
187     }
188
189     // reuse aux and get ip
190     memset(aux, 0, sizeof(aux));
191     if ((sprintf(aux, "%d.%d.%d.%d",
192     ip_bytes[0], ip_bytes[1], ip_bytes[2], ip_bytes[3]))
193         <7)
194     {
195         printf("ftp_pasv: Cannot compose ip address\n");
196         return 1;
197     }
198
199         DEBUG_SECTION(DEBUG_PASV,printf("pasv():ip:%s\n",aux);
```

```
200     );
201
202     // calculate port
203     int portResult = ports[0] * 256 + ports[1];
204
205     printf("IP: %s\n", aux);
206     printf("PORT: %d\n", portResult);
207
208     if ((data_socket_fd = connect_socket_TCP(aux, portResult)) < 0) {
209         printf( "ftp_pasv: Failed to connect data socket\n");
210         return 1;
211     }
212
213     return 0;
214 }
215
216 #define DEBUG_RETR 1
217 int ftp_retr(const char* filename) {
218     char aux[MAX_STRING_SIZE];
219
220     //send retr
221     sprintf(aux, "RETR %s\r\n", filename);
222     //sprintf(aux, "LIST %s\r\n", "");
223     if (ftp_send(aux, strlen(aux))< 0) {
224         printf("ftp_retr: Failed to send \n");
225         return 1;
226     }
227
228     //get respones
229     if (ftp_read(aux, sizeof(aux))< 0) {
230         printf("ftp_retr: Failed to get response\n");
231         return 1;
232     }
233
234     DEBUG_SECTION(DEBUG_PASV,printf("ftp_retr_debug_1:%s\n",aux););
235
236     return 0;
237 }
238
239 #define DEBUG_DOWNLOAD 0
240 int ftp_download(const char* filename) {
241
242   printf("\ndata_%d__cont_%d\n",data_socket_fd,  control_socket_fd);
243
244     FILE* file;
245     int bytes;
246
247     //create n open file
248     if (!(file = fopen(filename, "w"))) {
249         printf("ftp_download: Failed to create/open file\n");
250         return 1;
251     }
252
253
254     char buf[MAX_STRING_SIZE];
255     while ((bytes = recv(data_socket_fd,buf,MAX_STRING_SIZE,0))>0) {
256         if (bytes < 0) {
257             perror("ftp_download: Failed to receive from data socket\n"
                    );
258             fclose(file);
259             return 1;
260         }
261
262         DEBUG_SECTION(DEBUG_DOWNLOAD,
263                   printf("bytes:%d\n",bytes);
```

```
                        printf("rec:%s\n",buf);
                    );

        //output received bytes to file
        if ((bytes = fwrite(buf, bytes, 1, file)) < 0) {
            perror("ftp_download: Failed to write data in file\n");
            return 1;
        }
    }

    //close file and data socket
    fclose(file);
    close(data_socket_fd);

    return 0;
}

void ftp_abort()
{
    printf("\n ABORTED! \n");
    if(data_socket_fd) close(data_socket_fd);
    if(control_socket_fd) close(control_socket_fd);

}

#endif
```

D.4 Ficheiro socket.h

```

#ifndef SOCKET
#define SOCKET

/*return socket fd*/
int connect_socket_TCP(const char* ip, int port);

#endif
```

D.5 Ficheiro socket.c

```

#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
//#include <netdb.h>
#include <strings.h>
#include <stdio.h>


#include "socket.h"

int connect_socket_TCP(const char* ip, int port)
{
    //adapted from clientTCP.c

    int socket_fd;
    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*/
```

```
23      server_addr.sin_port = htons(port); /*server TCP port must be
           network byte ordered */
24
25      // open an TCP socket
26      if ((socket_fd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
27          perror("connect_socket:socket()");
28          return -1;
29      }
30
31      // connect to the server
32      if (connect(socket_fd, (struct sockaddr *) &server_addr, sizeof(
           server_addr)) < 0) {
33          perror("connect_socket:connect()");
34          return -1;
35      }
36
37      return socket_fd;
38  }
```

D.6 Ficheiro Utilities.h

```
1
2   #ifndef UTILITIES
3   #define UTILITIES
4
5   // section: should be a definition created by the programmer that must
       be equal to zero to avoid running the debug code.
6
7   #define DEBUG_SECTION(SECT,CODE) {\
8   if (SECT != 0)\
9   {\
10  CODE\
11  }\
12  }
13
14  #ifndef TYPEDEF_BOOLEAN_DECLARED_
15  #define TYPEDEF_BOOLEAN_DECLARED_
16  typedef int bool;
17  #endif /* TYPEDEF_BOOLEAN_DECLARED_*/
18
19  #define TRUE  1
20  #define YES   1
21  #define FALSE 0
22  #define NO    0
23  #define OK    0
24
25  #define PRINTBYTETOBINARY "%d%d%d%d%d%d%d%d"
26  #define BYTETOBINARY(byte)\
27  (byte & 0x80 ? 1 : 0),\
28  (byte & 0x40 ? 1 : 0),\
29  (byte & 0x20 ? 1 : 0),\
30  (byte & 0x10 ? 1 : 0),\
31  (byte & 0x08 ? 1 : 0),\
32  (byte & 0x04 ? 1 : 0),\
33  (byte & 0x02 ? 1 : 0),\
34  (byte & 0x01 ? 1 : 0)
35
36  #endif /* UTILITIES */
```