Segundo trabalho laboratorial

Mestrado Integrado em Engenharia Informática e Computação

Redes de Computadores

Grupo 8 Turma 2
João Romão - up201806779
Tiago Alves - up201603820

Faculdade de Engenharia da Universidade do Porto
Rua Roberto Frias, sn, 4200-465 Porto, Portugal

23 de dezembro de 2020

Conteúdo

1 Introdução

Este trabalho está dividido em duas partes: Desenvolver uma aplicação de download e Configurar uma rede de computadores, que com o auxilio da aplicação consiga fazer download de ficheiros de qualquer servidor FTP na Internet. A aplicação de download deve seguir o modelo standard de FTP, sendo esta explicada na secção 2, e a configuração da rede é feita ao longo de seis experiências presentes na secção 3. Durante a realização deste trabalho foram consolidados vários conceitos abordados nas aulas teóricas, tendo sido necessário utilizar estes conhecimentos para completar o trabalho com sucesso.

2 Primeira parte - Aplicação de download

Na realização deste trabalho foi realizada uma aplicação de download por ftp, onde o utilizador poderia indicar qual o ficheiro e site de onde quer fazer o download, assim como o utilizador e respectiva password caso seja necessário. De modo a correr o programa é necessário utilizar o seguinte comando: ***ftp://[<user>:<password>@]<host>/<url-path>***.

2.1 Arquitetura

Em primeiro lugar, a aplicação faz o parsing dos argumentos, atribuindo os valores dados a todas as variáveis de uma struct ***arguments***. Caso não tenha sido dado nos argumentos o utilizador e a respetiva password, é atribuído o valor de "anonymous"ao user, entrando assim em modo anónimo e não sendo necessária uma password. De seguida foi obtido o IP do host dado, usando a função ***getHostIP(char* hostName)*** já previamente fornecida. Depois disto, é então aberta a conecção com o socket usando a função ***ftp_open_connection(char *serverAddr, int serverPort)***, sendo utilizado o port 21, default.

No caso de a ligação ter sido estabelida com sucesso, é lida a mensagem de boas vindas, através da função ***ftp_poll_read(int fd, const char *ready_state, char *buff)***, que usando polling, lê e dá parse desta mensagem até que seja interpretado o código de sucesso passado nos argumentos, 220 neste caso. De modo a autenticarmos o utilizador, é então preciso fazer login usando a função ***ftp_login(int sockFd, char * user, char * pass);*** que envia os dados de user e pass contidos na struct arguments, fazendo sempre a verificação de erros e se foi autenticado com sucesso. Com o objetivo de os ficheiros serem obtidos usando uma stream binária, foi chamada a função ***ftp_set_binary_mode(int sock_fd);***, que envia o comando "TYPE I", informando que se quer fazer o download com este formato.

De seguida, inicializámos a struct ***pasv_info*** que tem duas variáveis, o ***port_number***, representando o port de onde vai ser feita a troca dos dados, assim como o ***ip_adress***, representando o endereço IP. Esta struct é assim passado como argumento à função ***ftp_set_passive_mode(int sock_fd, pasv_info *pasv)***, que informa o servidor que se pretende entrar no modo passivo, sendo a struct populada com o novo port e endereço IP, sendo logo de seguida aberta uma nova conexão para transmitir os dados do ficheiro. No caso de a mensagem recebida ser "227 Entering Passive Mode (192,168,109,136,172,172)", por exemplo, o port é calculado fazendo 256*172+172, obtendo o port 44204 e o IP 192.168.109.136.

Para pedir o ficheiro é chamada a função ***ftp_send_retr(int sock_fd, char *path)*** que envia o comando RETR para o novo socket, e depois fazer o download efetivo do ficheiro chamando a função ***ftp_retr_file(int sock_fd, char *path)***. Por fim é chamada a funçao ***ftp_close_connection(int sock_fd);*** que faz o fecho da ligação com o servidor com o comando "quit".

2.2 Resultado

A aplicação funcionou como esperado, tendo funcionado para diferentes tipos de ficheiros e diferentes domínios FTP. Foram realizados testes com ficheiros de texto, imagem e vídeo nos seguintes domínios: netlab1.fe.up.pt, ftp.up.pt e ftp.gnu.org, tendo em todos os casos obtido o

resultado esperado. Na figura 28 da secção de anexos é possível observar um resultado exemplo da execução da aplicação de download.

3 Segunda parte - Configuração e análise de redes

3.1 Experiência 1

Esta experiência tinha como objetivo a comunicação de duas máquinas, tux43 e tux44. Para isto, foi necessário configurar os IP's das portas dos dois computadores utilizando o comando ***ifconfig eth0 <ip>/<máscara>*** para os ips 172.16.40.1/24 e 172.16.40.254/24 respectivamente. De seguida foram adicionadas as rotas necessárias à Arp table e foi enviado o sinal ping, ***ping <ipDestino>*** de modo a verificarmos a transmissão de mensagens entre eles.

A ligação entre tux44 e tux43 foi reaizada da seguinte maneira: tux44S0 -> T3; tux43E0 -> Switch; tux44E0 -> Switch; T4 -> Switch console

Os **pacotes ARP, Adress Resolution Protocol**, têm como função mapear um endereço da camada internet, Ip Adress, a endreços da camada de Network, MAC Adress. De modo a testar o processo completo, desde a criação das entradas nesta tabela, foi utilizado o comando ***arp −d <ipAdress>*** de modo a apagar as entradas pré existentes. De seguida, de maneira a conseguir enviar os pacotes de um computador para o outro, o computador envia um pacote ARP de ***broadcast*** para todas as máquinas da rede local, perguntando qual das máquinas tem um endereço MAC correspondente ao endereço IP do destinatário. Como resposta o destinatário envia um pacote ARP, indicando de qual o endereço MAC da máquina correspondente, sendo assim possível adicionar à ARP table a entrada e fazer a comunicação.

```
Broadcast           ARP          42 Who has 172.16.40.254? Tell 172.16.40.1
HewlettP_61:2f:d4   ARP          60 172.16.40.254 is at 00:21:5a:5a:7b:ea
```

Figura 1: Broadcast Arp Packet

Caso as entradas da ARP table tenham sido apagadas no tux 43, é necessário enviar um pacote de Broadcast, perguntando a todas as máquinas da rede local qual tem o endereço MAC relativo ao endereço IP *172.16.40.254*, sendo que este pacote contêm o endereço IP do tux43, de modo às outras máquinas saberem para onde enviar o pacote de resposta e o endereço MAC de 00:00:00:00:00:00. Como resposta, o tux 44 envia um pacote ARP para o tux1, enviando o seu endereço MAC *00:21:5a:5a:7b:ea* e o seu endereço IP *172.16.40.254*.

O comando ping primeiramente gera pacotes ARP, caso as entradas tenham sido apagadas, e de seguida gera pacotes ICMP, Internet Control Message Protocol. Sendo que existem dois tipos de pacotes ICMP, o pedido, que é enviado do IP 172.16.40.1, MAC (00:21:5a:61:2f:d4) para o endereço IP 172.16.40.254, MAC (00:21:5a:5a:7b:ea), e a resposta, que tem os mesmos valores mas na direção oposta. O protocolo da trama recebida, assim como o tamanho do pacote pode ser vizualizado a partir do WireShark.

A interface de rede virtual desta experiência é ***loopback***, utilizada para um computador comunicar na própria rede/computador, de modo a conseguir aceder a servidores na própria máquina ou verificar a correta montagem de rede.

3.2 Experiência 2

Nesta experiência foram criadas 2 VLANs (Virtual Local Area Network):

- **VLAN 40**: à qual se ligaram os tux43 e tux44;
- **VLAN 41**: à qual se ligou o tux42.

A experiência tem assim como principal objetivo entender como é feita a configuração de uma VLAN e como é realizada a troca de informação entre máquinas.

Percebeu-se também que o tux42 é inatingível por tux43 e tux44, uma vez que não há qualquer rota possível entre eles, estando ligados a VLANs distintas sem qualquer ligação intermediária.

A partir dos logs retirados nesta experiência, mais concretamente aqueles relacionados com a execução de ping -b 172.16.40.255 no tux43, ping -b 172.16.41.255 no tux42 existem **dois domínios de transmissão**, um associado a cada VLAN:

- VLAN40: 172.168.40.255
- VLAN41: 172.168.41.255

O modo de **configuração das VLANs** está descrito nos anexos na secção de configurações.

3.3 Experiência 3

Nesta experiência o tux44 foi ligado a ambas as VLANs (40 e 41) de modo a funcionar como **router**, com ***IP forwarding ativo*** . Deste modo o tux44 consegue direcionar pacotes de uma VLAN para a outra. Para isto foi configurada uma nova rota entre o tux44 e a VLAN41:

1. ifconfig eth1 172.16.41.253/24 (ligação tux44 eth1 a 0/5)
2. Adição da porta 0/5 à VLAN41
3. tux42: route add -net 172.16.41.0/24 gw 172.16.40.254
4. tux43: route add -net 172.16.40.0/24 gw 172.16.41.253
5. echo 1 > /proc/sys/net/ipv4/ip_forward
6. echo 0 > /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts

No inicio da experiência o tux apenas tem as rotas que são geradas automaticamente quando ligado à respectiva VLAN, sendo que o tux 43 tem uma rota para a VLAN 40, o tux42 para a VLAN 41 e finalmente o tux44 que tem uma rota para ambas as VLAN. Estas rotas têm um gateway com um valor de 0.0.0.0

No entanto, no passo 3, onde se pretende que o tux 43 consiga comunicar com o tux 42, adicionam-se duas novas rotas, uma usando o comando 3, que adiciona uma rota do tux42 para a VLAN com endereço 172.16.41.0/24 tendo como gateway o IP 72.16.40.254, e o comando 4, que indica ao tux43 que para chegar à VLAN com endereço 172.16.40.0/24 deve usar como gateway o IP 72.16.41.253. Ao adicionar estas rotas, quando se quer enviar um ping do tux 42, por exemplo, para a VLAN 40, envia-se primeiro para o router 172.16.41.253

Uma **entrada da forwarding table** tem as seguintes informações associadas:

- **Destination** - IP da rede de destino
- **Gateway** - IP do router ao qual se deve encaminhar a mensagem, de modo a ele transmiti-la para a rede destino.
- **Netmask** - Utilizado para determinar o ID da rede a partir do endereço destino.
- **Flags** - Informações sobre a rota.
- **Metric** - Utilizada para escolher a melhor rota, caso várias estejam disponíveis.
- **Use** - Contador de pesquisas pela rota.
- **Interface** - Indica que interface está localmente responsável por chegar ao gateway. (eth0, eth1).

Figura 2: Exemplo de mensagem ARP trocada na comunicação entre tux43 e tux42

Quando o tux43 envia um ping ao tux42, o tux43 não conhece o endereço MAC do tux42, sendo deste modo necessário o envio de uma mensagem ***ARP***.

Uma vez que ambos os computadores se econtram associados a subredes diferentes, no caso do tux43, que se encontra associado à VLAN40, o pedido será enviado para a interface eth0 do tux44. O tux44 por sua vez possui uma rota direta para o tux42, estando a interface eth1 do tux44 e o tux42 na mesma subrede.

O processo repete-se quando o tux42 tenta responder, mas no sentido inverso, sendo que o endereço MAC registado desta vez não será o de tux44 eth0 mas sim tux44 eth1.

Isto faz com que os endereços MAC associados aos pacotes ICMP serão os do tux43 e da interface eth0 do tux44. Os endereços IP presentes nos pacotes ICMP continuarão a ser os endereços do tux43 (172.16.40.1) e do tux42 (172.16.41.1).

Antes de as rotas serem adicionadas, os ***pacotes ICMP*** são do tipo ***Host unreachable***, visto que não conseguem chegar à rede destino. No entanto, após configurar as rotas corretamente, passamos a ter pacotes do tipo reply e request, tal como explicado na secção 3.1.

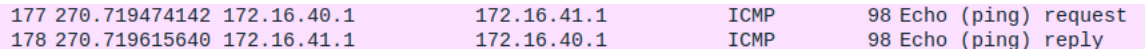


Figura 3: Visualização de pacotes ICMP e respetivos endereços IP

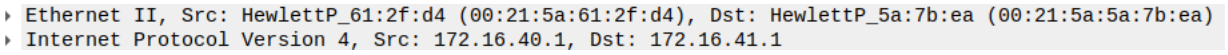


Figura 4: Endereços MAC src: tux43 (00:21:5a:61:2f:d4) dst: tux44 eth0 (00:21:5a:5a:7b:ea)

3.4 Experiência 4

Com esta experiência, pretende-se configurar um router comercial, ligando-o à rede do laboratório 172.16.1.0/24. De seguida dá-se então a implementação do NAT, de modo a ser possível conectar-se com a Internet .

De modo a configurar a rede para esta experiência, foi necessário fazer algumas configurações adicionais às feitas anteriormente:

- tux42 - route del default gw 172.16.41.253, que remove a rota pré definida do tux42 para o tux44
- tux42 - route add default gw 172.16.41.254, que adiciona a rota default do tux42 para o Router comercial
- tux44 - route add default gw 172.16.41.254, que adiciona a rota default do tux44 para o Router comercial

De modo a definir as duas rotas no router foi usado o comando

- ***ip route <ipDestino><máscara><gateway>***

Com os argumentos

- ***ip route 0.0.0.0 0.0.0.0 172.16.1.254***
- ***ip route 172.16.40.0 255.255.255.0 172.16.41.253***.

É importante salientar que para correr estes comandos deve-se iniciar sessão no router através do ***gkterm*** e fazer a correta configuração dos cabos.

As rotas default dos tux42 e 44 foram alteradas do modo a que as máquinas se conseguissem conectar através da Internet, pelo router comercial. Após termos feito a configuração já referida, as mensagens já são encaminhadas pelo Rc por default, a não ser que a outra máquina esteja na mesma sub rede, como é o caso do tux43 e tux44, ambos ligados à VLAN40.

Quando se faz ping para o tux43 do tux42, caso os **redirects estejam desativados** no tux 42, os pacotes ICMP de pedido vão seguir a rota tux42 → Rc → tux44 → tux43 e os pacotes de resposta vão seguir a rota oposta.

No entanto, quando os **redirects estão ativos**, ao tentar fazer ping do tux42 para o tux43, o tux42 enviou o pacote ICMP para o Rc, sendo que depois de ter consultado a tabela das rotas, descobriu que o próximo *salto* para chegar ao tux43 era usando o tux44, na sua ligação à VLAN41. Consequentemente, o router reencaminhou o pacote para o tux44 e enviou uma mensagem de ICMP redireccionamento para o tux42, informado-o que esta era a melhor rota para chegar ao tux43. Desta maneira as próximas mensagens enviadas do tux42 para o tux43, seriam todas enviadas pelo tux 44 sem ser necessário recorrer ao router.

O **NAT, Network Adress Translation** é um processo específico que envolve remapear um único endereço IP, muitas vezes público, para o endereço IP destino dentro de uma rede privada, mascarando assim o remetente/destinatário dos pacotes enviados, assegurando a privacidade deste último. Isto é conseguido alterando as informações de rede e informações de endereço encontradas no cabeçalho IP dos pacotes de dados. Além disto, sem o NAT não é possível fazer esta comunicação, visto que ele permite que as redes privadas que usam endereços não registados se conectem e comuniquem com redes públicas.
O modo de **configuração do router com NAT** está descrito nos anexos na secção de configurações.

3.5 Experiência 5

Na experiência 5, o objetivo era configurar o **Domain Name System**, de forma a traduzir os nomes de domínios em endereços IP, permitindo assim a ligação dos computadores à Internet usando hostnames.

Para se fazer a configuração do DNS, procedeu-se à alteração do ficheiro resolv.conf, situado no diretório etc/ do tux em questão. Foram então adicionados os comandos ***search netlab.fe.up.pt***, que representa o nome do servidor DNS e ***nameserver 172.16.1.1***, com o respetivo endereço IP. É importante salientar ainda que sem o DNS o ping ao servidor ftp.up.pt, por exemplo, falhava pois não tinha meio de conseguir descobrir qual o endereço IP do servidor.

São trocados ***dois pacotes pelo DNS***, um de pedido, enviado para o servidor, que contêm o nome de domínio, e um de resposta, enviado pelo servidor que contêm o IP do hostname em causa.

3.6 Experiência 6

Nesta experiência foi utilizada a **aplicação de download** explicada na *secção 2* deste relatório, para se observar o comportamento do protocolo TCP.
É necessário abrir duas ligações TCP, uma para a transmissão de comandos FTP, estabelecida quando se entra em contacto com o servidor e outra para a troca de dados, o canal onde é transferido o ficheiro. O controlo de informação FTP é transportado na primeira conecção.

Existem três fases associadas a uma ligação TCP, a do estabelecimento da conecção, a de troca de dados e finalmente o encerramento da mesma.

O **mecanismo ARQ**, Autmatic Repeat Request, consiste no controlo de erros durante a transmissão de dados, onde o recetor não deixa de processar os frames recebidos quando deteta um erro, continuando a receber frames e enviando no acknowlegment number o número da frame que falhou. Este mecanismo funciona através do método da janela deslizante.

O TCP usa um controlo de congestionamento end to end, fazendo com que o remetente limite ou aumente a taxa de entrega de dados para a conexão em função do congestionamento percebido por ele, sendo assim ***auto-regulado***. A conexão TCP é composta de um ***buffer de recepção***, um ***buffer de envio*** e de diversas variáveis, sendo que dentre essas variáveis, temos a ***janela de congestionamento***, que limita a taxa de envio de pacotes de um remetente TCP.

No início de cada ***RTT***, "Round Trip Time", o remetente envia os seus pacotes de acordo com o tamanho da janela de congestionamento estabelecido, recebendo no fim um sinal que indica que todos os pacotes foram enviados corretamente. A cada ***ACK*** recebido, o TCP reconhece que não há congestionamento da rede, aumentando assim a janela lentamente a cada tempo de ida e volta. Quando existe algum evento de perda, ou caso sejam detetados três ACKs duplicados, o remetente reduz a sua janela de congestionamento a metade, sendo que o tamanho desta tem um valor mínimo de 1 MSS (maximum segment size). Este comportamento de ser aumentado lentamente sendo depois reduzido para metade pode ser notado no gráfico, ficando com uma aparência em "dentes de serra"como observado na figura 26 presente nos anexos.

Durante o início de uma conexão TCP temos a fase de partida lenta, quando o remetente transmite a uma taxa lenta (normalmente 1 MSS) e depois aumenta sua taxa exponencialmente, duplicando o valor de janela de congestionamento a cada tempo de ida e volta até acontecer um evento de perda. O remetente TCP também pode entrar em fase de partida lenta após um evento de esgotamento de temporização, ajustando a janela de congestionamento para 1 MSS e aumentando exponencialmente até que a janela alcance metade do valor que tinha antes do evento (Threshold, em português, patamar).

Na **segunda parte da experiência** pretendia-se começar uma nova transferência num novo computador enquanto a transferência inicial não tivesse terminado.

Com o **aparecimento de uma segunda conexão TCP** (na figura 26 presente por volta dos 15s) a taxa de transmissão da conexão inicial diminuí devido a limitações da largura de banda. Isto acontece pois a largura de banda é distribuída por cada ligação existente.

4 Conclusões

Apesar do reduzido acesso aos laboratórios, pensamos que o trabalho foi desenvolvido com sucesso, sendo que atingimos todos os objetivos pretendidos. Durante a realização deste tivemos a oportunidade de consolidar vários conceitos, como os protocolos FTP e TCP, assim como de perceber as diversas máquinas e dispositivos utilizados, como os routers e o switch assim como as diversas técnicas e protocolos.

5 Referências

Este trabalho foi desenvolvido usando todos os recursos dados pelos professores, tanto pelos slides das teóricas, como pelo livro, pelos RFC's e finalmente, pelo guião do trabalho.

6 Anexos

6.1 Tabela de endereços

Máquina	Endereço IP	Endereço MAC
tux42	172.16.41.1	00:1f:29:d7:45:c4
tux43	172.16.40.1	00:21:5a:61:2f:d4
tux44 eth0	172.16.40.254	00:21:5a:5a:7b:ea
tux44 eth1	172.16.41.253	00:c0:df:25:1a:f4
Router	172.16.41.254	68:ef:bd:e3:df:10

6.2 Configurações

6.2.1 Configuração das VLANs

- Criação da VLAN40
  - configure terminal
  - vlan 40
  - end
  - show vlan id 40 (passo de verificação)
- Adicionar a porta 0/1 à VLAN40
  - configure terminal
  - interface fastethernet 0/1, sendo 1 o número da porta do switch que estamos a configurar
  - switchport mode access
  - switchport access vlan 40
  - end

Na realização da experiência optou-se pela seguinte configuração de cabos:

- tux42 Eth0 ligado a 0/2 [VLAN41]
- tux43 Eth0 ligado a 0/1 [VLAN40]
- tux44 Eth0 ligado a 0/3 [VLAN40]

6.2.2 Configuração de NAT num router comercial

Configuração NAT inside:

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

Configuração NAT outside:

- interface gigabitethernet 0/1
- ip address 172.16.1.y9 255.255.255.0
- no shutdown

- ip nat outside
- exit

Configuração de propriedades NAT:

- ip nat pool ovrld 172.16.1.y9 172.16.1.49 prefix 24
- ip nat inside source list 1 pool ovrld overload

Declaração da lista de acessos válidos:

- access-list 1 permit 172.16.40.0 0.0.0.7
- access-list 1 permit 172.16.41.0 0.0.0.7
- ip route 0.0.0.0 0.0.0.0 172.16.1.254
- ip route 172.16.40.0 255.255.255.0 172.16.41.253
- end

Configuração das rotas IP

- ip route 0.0.0.0 0.0.0.0 172.16.1.254
- ip route 172.16.40.0 255.255.255.0 172.16.41.253
- end

6.3 Experiência 1

```
    15 17.279342098  HewlettP_61:2f:d4  Broadcast           ARP      42 Who has 172.16.40.254? Tell 172.16.40.1
    16 17.279478076  HewlettP_5a:7b:ea  HewlettP_61:2f:d4   ARP      60 172.16.40.254 is at 00:21:5a:5a:7b:ea
    17 17.279486806  172.16.40.1        172.16.40.254       ICMP     98 Echo (ping) request  id=0x1dd3, seq=1/256, ttl=64 (reply in 1…
    18 17.279620759  172.16.40.254      172.16.40.1         ICMP     98 Echo (ping) reply    id=0x1dd3, seq=1/256, ttl=64 (request in…
    19 18.044011837  Cisco_d4:1c:03     Spanning-tree-(for-… STP      60 Conf. Root = 32768/1/30:37:a6:d4:1c:00  Cost = 0  Port = 0x80…
    20 18.295101272  172.16.40.1        172.16.40.254       ICMP     98 Echo (ping) request  id=0x1dd3, seq=2/512, ttl=64 (reply in 2…
    21 18.295232082  172.16.40.254      172.16.40.1         ICMP     98 Echo (ping) reply    id=0x1dd3, seq=2/512, ttl=64 (request in…
    22 19.319100787  172.16.40.1        172.16.40.254       ICMP     98 Echo (ping) request  id=0x1dd3, seq=3/768, ttl=64 (reply in 2…
    23 19.319235298  172.16.40.254      172.16.40.1         ICMP     98 Echo (ping) reply    id=0x1dd3, seq=3/768, ttl=64 (request in…
    24 20.048960637  Cisco_d4:1c:03     Spanning-tree-(for-… STP      60 Conf. Root = 32768/1/30:37:a6:d4:1c:00  Cost = 0  Port = 0x80…
    25 20.343102327  172.16.40.1        172.16.40.254       ICMP     98 Echo (ping) request  id=0x1dd3, seq=4/1024, ttl=64 (reply in …
    26 20.343235442  172.16.40.254      172.16.40.1         ICMP     98 Echo (ping) reply    id=0x1dd3, seq=4/1024, ttl=64 (request i…
    27 21.367100865  172.16.40.1        172.16.40.254       ICMP     98 Echo (ping) request  id=0x1dd3, seq=5/1280, ttl=64 (reply in …
    28 21.367235865  172.16.40.254      172.16.40.1         ICMP     98 Echo (ping) reply    id=0x1dd3, seq=5/1280, ttl=64 (request i…
    29 22.053823953  Cisco_d4:1c:03     Spanning-tree-(for-… STP      60 Conf. Root = 32768/1/30:37:a6:d4:1c:00  Cost = 0  Port = 0x80…
    30 22.391101218  172.16.40.1        172.16.40.254       ICMP     98 Echo (ping) request  id=0x1dd3, seq=6/1536, ttl=64 (reply in …
▸ Frame 15: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface eth0, id 0
▸ Ethernet II, Src: HewlettP_61:2f:d4 (00:21:5a:61:2f:d4), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
▾ Address Resolution Protocol (request)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (1)
    Sender MAC address: HewlettP_61:2f:d4 (00:21:5a:61:2f:d4)
    Sender IP address: 172.16.40.1
    Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
    Target IP address: 172.16.40.254
```

Figura 5: Pergunta pelo endereço MAC

```
    15 17.279342098  HewlettP_61:2f:d4  Broadcast           ARP      42 Who has 172.16.40.254? Tell 172.16.40.1
    16 17.279478076  HewlettP_5a:7b:ea  HewlettP_61:2f:d4   ARP      60 172.16.40.254 is at 00:21:5a:5a:7b:ea
    17 17.279486806  172.16.40.1        172.16.40.254       ICMP     98 Echo (ping) request  id=0x1dd3, seq=1/256, ttl=64 (reply in 1…
    18 17.279620759  172.16.40.254      172.16.40.1         ICMP     98 Echo (ping) reply    id=0x1dd3, seq=1/256, ttl=64 (request in…
    19 18.044011837  Cisco_d4:1c:03     Spanning-tree-(for-… STP      60 Conf. Root = 32768/1/30:37:a6:d4:1c:00  Cost = 0  Port = 0x80…
    20 18.295101272  172.16.40.1        172.16.40.254       ICMP     98 Echo (ping) request  id=0x1dd3, seq=2/512, ttl=64 (reply in 2…
    21 18.295232082  172.16.40.254      172.16.40.1         ICMP     98 Echo (ping) reply    id=0x1dd3, seq=2/512, ttl=64 (request in…
    22 19.319100787  172.16.40.1        172.16.40.254       ICMP     98 Echo (ping) request  id=0x1dd3, seq=3/768, ttl=64 (reply in 2…
    23 19.319235298  172.16.40.254      172.16.40.1         ICMP     98 Echo (ping) reply    id=0x1dd3, seq=3/768, ttl=64 (request in…
    24 20.048960637  Cisco_d4:1c:03     Spanning-tree-(for-… STP      60 Conf. Root = 32768/1/30:37:a6:d4:1c:00  Cost = 0  Port = 0x80…
    25 20.343102327  172.16.40.1        172.16.40.254       ICMP     98 Echo (ping) request  id=0x1dd3, seq=4/1024, ttl=64 (reply in …
    26 20.343235442  172.16.40.254      172.16.40.1         ICMP     98 Echo (ping) reply    id=0x1dd3, seq=4/1024, ttl=64 (request i…
    27 21.367100865  172.16.40.1        172.16.40.254       ICMP     98 Echo (ping) request  id=0x1dd3, seq=5/1280, ttl=64 (reply in …
    28 21.367235865  172.16.40.254      172.16.40.1         ICMP     98 Echo (ping) reply    id=0x1dd3, seq=5/1280, ttl=64 (request i…
    29 22.053823953  Cisco_d4:1c:03     Spanning-tree-(for-… STP      60 Conf. Root = 32768/1/30:37:a6:d4:1c:00  Cost = 0  Port = 0x80…
    30 22.391101218  172.16.40.1        172.16.40.254       ICMP     98 Echo (ping) request  id=0x1dd3, seq=6/1536, ttl=64 (reply in …
▸ Frame 16: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface eth0, id 0
▸ Ethernet II, Src: HewlettP_5a:7b:ea (00:21:5a:5a:7b:ea), Dst: HewlettP_61:2f:d4 (00:21:5a:61:2f:d4)
▾ Address Resolution Protocol (reply)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: reply (2)
    Sender MAC address: HewlettP_5a:7b:ea (00:21:5a:5a:7b:ea)
    Sender IP address: 172.16.40.254
    Target MAC address: HewlettP_61:2f:d4 (00:21:5a:61:2f:d4)
    Target IP address: 172.16.40.1
```

Figura 6: Confirmação do endereço MAC

6.4 Experiência 2

```
    66 82.087591605  172.16.40.1          172.16.40.255        ICMP       98 Echo (ping) request  id=0x2b9e, seq=7/1792, ttl=64 (no respon…
    67 82.087740504  172.16.40.254         172.16.40.1          ICMP       98 Echo (ping) reply    id=0x2b9e, seq=7/1792, ttl=64
    68 82.205582158  Cisco_d4:1c:03        Spanning-tree-(for-… STP        60 Conf. Root = 32768/40/30:37:a6:d4:1c:00  Cost = 0  Port = 0x8…
    69 83.111594403  172.16.40.1          172.16.40.255        ICMP       98 Echo (ping) request  id=0x2b9e, seq=8/2048, ttl=64 (no respon…
    70 83.111760971  172.16.40.254         172.16.40.1          ICMP       98 Echo (ping) reply    id=0x2b9e, seq=8/2048, ttl=64
    71 84.135592242  172.16.40.1          172.16.40.255        ICMP       98 Echo (ping) request  id=0x2b9e, seq=9/2304, ttl=64 (no respon…
    72 84.135756366  172.16.40.254         172.16.40.1          ICMP       98 Echo (ping) reply    id=0x2b9e, seq=9/2304, ttl=64
    73 84.205513379  Cisco_d4:1c:03        Spanning-tree-(for-… STP        60 Conf. Root = 32768/40/30:37:a6:d4:1c:00  Cost = 0  Port = 0x8…
    74 85.159593573  172.16.40.1          172.16.40.255        ICMP       98 Echo (ping) request  id=0x2b9e, seq=10/2560, ttl=64 (no respo…
    75 85.159761817  172.16.40.254         172.16.40.1          ICMP       98 Echo (ping) reply    id=0x2b9e, seq=10/2560, ttl=64
    76 86.183605590  172.16.40.1          172.16.40.255        ICMP       98 Echo (ping) request  id=0x2b9e, seq=11/2816, ttl=64 (no respo…
    77 86.183775370  172.16.40.254         172.16.40.1          ICMP       98 Echo (ping) reply    id=0x2b9e, seq=11/2816, ttl=64
    78 86.210372365  Cisco_d4:1c:03        Spanning-tree-(for-… STP        60 Conf. Root = 32768/40/30:37:a6:d4:1c:00  Cost = 0  Port = 0x8…
    79 87.207592534  172.16.40.1          172.16.40.255        ICMP       98 Echo (ping) request  id=0x2b9e, seq=12/3072, ttl=64 (no respo…
    80 87.207757705  172.16.40.254         172.16.40.1          ICMP       98 Echo (ping) reply    id=0x2b9e, seq=12/3072, ttl=64
```

Figura 7: Ping broadcast a partir de tux43

```
   246 393.477487615 172.16.41.1          172.16.41.255        ICMP       98 Echo (ping) request  id=0x30a1, seq=1/256, ttl=64 (no respons…
   247 394.482223832 172.16.41.1          172.16.41.255        ICMP       98 Echo (ping) request  id=0x30a1, seq=2/512, ttl=64 (no respons…
   248 395.010460977 Cisco_d4:1c:04        Spanning-tree-(for-… STP        60 Conf. Root = 32768/41/30:37:a6:d4:1c:00  Cost = 0  Port = 0x8…
   249 395.506230097 172.16.41.1          172.16.41.255        ICMP       98 Echo (ping) request  id=0x30a1, seq=3/768, ttl=64 (no respons…
   250 396.530234686 172.16.41.1          172.16.41.255        ICMP       98 Echo (ping) request  id=0x30a1, seq=4/1024, ttl=64 (no respon…
   251 397.013960445 Cisco_d4:1c:04        Spanning-tree-(for-… STP        60 Conf. Root = 32768/41/30:37:a6:d4:1c:00  Cost = 0  Port = 0x8…
   252 397.554233478 172.16.41.1          172.16.41.255        ICMP       98 Echo (ping) request  id=0x30a1, seq=5/1280, ttl=64 (no respon…
   253 398.578229267 172.16.41.1          172.16.41.255        ICMP       98 Echo (ping) request  id=0x30a1, seq=6/1536, ttl=64 (no respon…
   254 399.015142718 Cisco_d4:1c:04        Spanning-tree-(for-… STP        60 Conf. Root = 32768/41/30:37:a6:d4:1c:00  Cost = 0  Port = 0x8…
   255 399.606227730 172.16.41.1          172.16.41.255        ICMP       98 Echo (ping) request  id=0x30a1, seq=7/1792, ttl=64 (no respon…
   256 400.626227480 172.16.41.1          172.16.41.255        ICMP       98 Echo (ping) request  id=0x30a1, seq=8/2048, ttl=64 (no respon…
   257 401.025241768 Cisco_d4:1c:04        Spanning-tree-(for-… STP        60 Conf. Root = 32768/41/30:37:a6:d4:1c:00  Cost = 0  Port = 0x8…
   258 401.650228646 172.16.41.1          172.16.41.255        ICMP       98 Echo (ping) request  id=0x30a1, seq=9/2304, ttl=64 (no respon…
   259 402.409496358 Cisco_d4:1c:04        Cisco_d4:1c:04       LOOP       60 Reply
   260 402.674212841 172.16.41.1          172.16.41.255        ICMP       98 Echo (ping) request  id=0x30a1, seq=10/2560, ttl=64 (no respo…
```

Figura 8: Ping broadcast a partir de tux42

6.5 Experiência 3

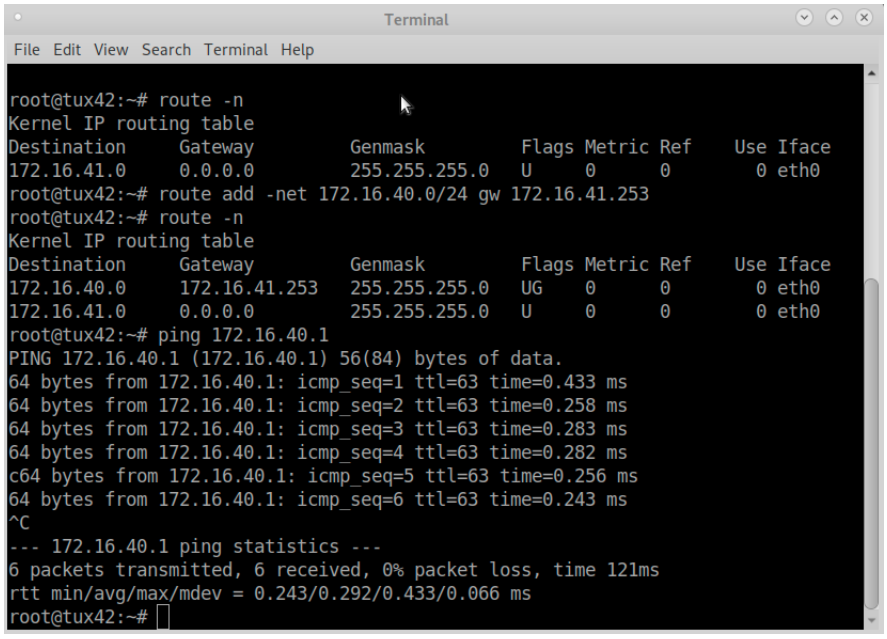


Figura 9: Verificação de rotas no tux42

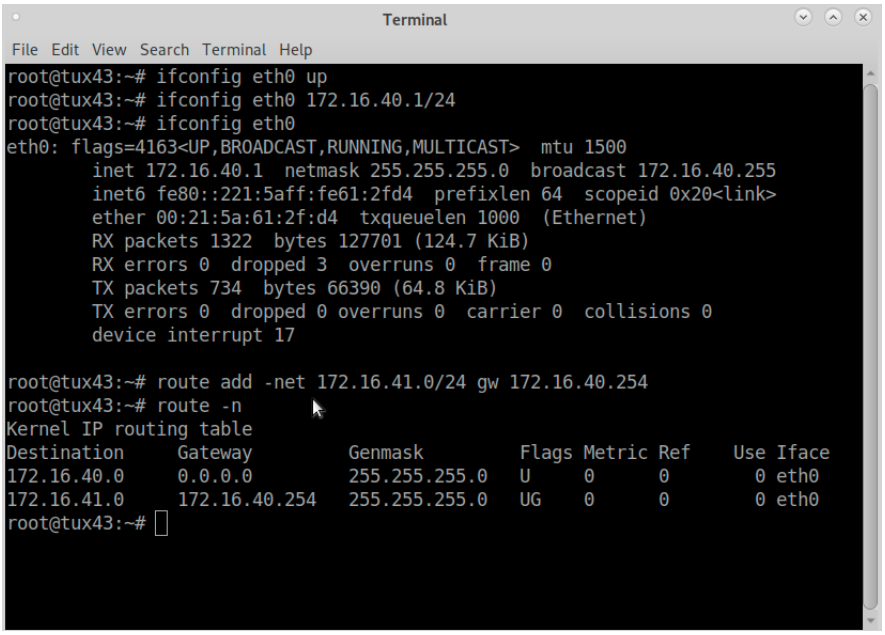


Figura 10: Verificação de rotas no tux43

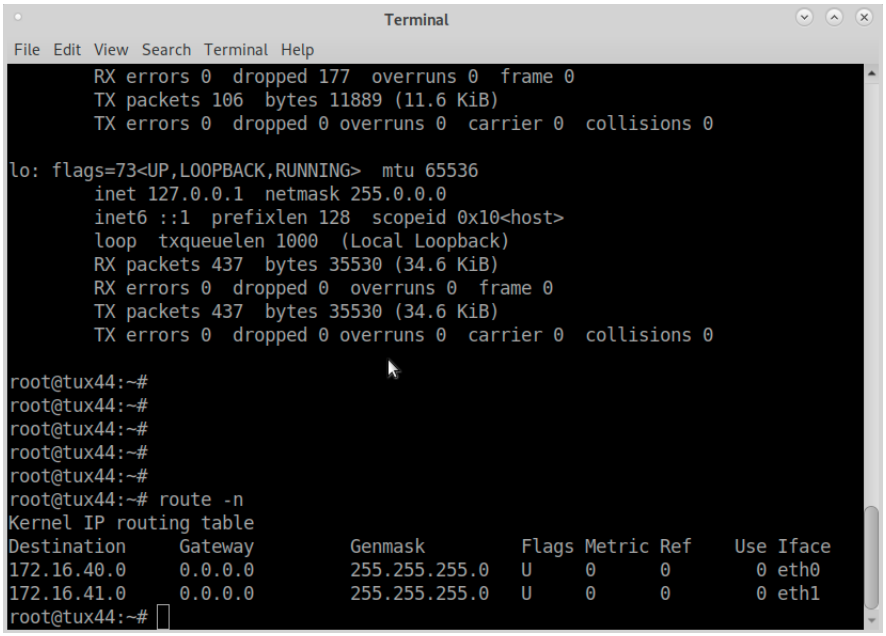


Figura 11: Verificação de rotas no tux44

1	0.000000000	172.16.40.1	172.16.41.1	ICMP	98	Echo (ping) request id=0x1bec, seq=9/2304, ttl=64 (reply in …
2	0.000265814	172.16.41.1	172.16.40.1	ICMP	98	Echo (ping) reply id=0x1bec, seq=9/2304, ttl=63 (request i…
3	1.024000701	172.16.40.1	172.16.41.1	ICMP	98	Echo (ping) request id=0x1bec, seq=10/2560, ttl=64 (reply in…
4	1.024301645	172.16.41.1	172.16.40.1	ICMP	98	Echo (ping) reply id=0x1bec, seq=10/2560, ttl=63 (request …
5	1.850679521	Cisco_d4:1c:03	Spanning-tree-(for-…	STP	60	Conf. Root = 32768/40/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8…
6	2.047994418	172.16.40.1	172.16.41.1	ICMP	98	Echo (ping) request id=0x1bec, seq=11/2816, ttl=64 (reply in…
7	2.048256740	172.16.41.1	172.16.40.1	ICMP	98	Echo (ping) reply id=0x1bec, seq=11/2816, ttl=63 (request …
8	3.072002802	172.16.40.1	172.16.41.1	ICMP	98	Echo (ping) request id=0x1bec, seq=12/3072, ttl=64 (reply in…
9	3.072278253	172.16.41.1	172.16.40.1	ICMP	98	Echo (ping) reply id=0x1bec, seq=12/3072, ttl=63 (request …
10	3.855566674	Cisco_d4:1c:03	Spanning-tree-(for-…	STP	60	Conf. Root = 32768/40/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8…
11	4.095991630	172.16.40.1	172.16.41.1	ICMP	98	Echo (ping) request id=0x1bec, seq=13/3328, ttl=64 (reply in…
12	4.096251437	172.16.41.1	172.16.40.1	ICMP	98	Echo (ping) reply id=0x1bec, seq=13/3328, ttl=63 (request …
13	4.426186146	Cisco_d4:1c:03	Cisco_d4:1c:03	LOOP	60	Reply
14	5.120000293	172.16.40.1	172.16.41.1	ICMP	98	Echo (ping) request id=0x1bec, seq=14/3584, ttl=64 (reply in…
15	5.120278468	172.16.41.1	172.16.40.1	ICMP	98	Echo (ping) reply id=0x1bec, seq=14/3584, ttl=63 (request …
16	5.860564175	Cisco_d4:1c:03	Spanning-tree-(for-…	STP	60	Conf. Root = 32768/40/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8…
17	6.143999806	172.16.40.1	172.16.41.1	ICMP	98	Echo (ping) request id=0x1bec, seq=15/3840, ttl=64 (reply in…
18	6.144292089	172.16.41.1	172.16.40.1	ICMP	98	Echo (ping) reply id=0x1bec, seq=15/3840, ttl=63 (request …
19	7.167998901	172.16.40.1	172.16.41.1	ICMP	98	Echo (ping) request id=0x1bec, seq=16/4096, ttl=64 (reply in…
20	7.168277565	172.16.41.1	172.16.40.1	ICMP	98	Echo (ping) reply id=0x1bec, seq=16/4096, ttl=63 (request …

Figura 12: Ping apartir de tux43 para tux42

170	268.692205466	HewlettP_61:2f:d4	Broadcast	ARP	60	Who has 172.16.40.254? Tell 172.16.40.1
171	268.692228723	HewlettP_5a:7b:ea	HewlettP_61:2f:d4	ARP	42	172.16.40.254 is at 00:21:5a:5a:7b:ea
172	268.692356323	172.16.40.1	172.16.41.1	ICMP	98	Echo (ping) request id=0x1bec, seq=1/256, ttl=64 (reply in 1…
173	268.692602722	172.16.41.1	172.16.40.1	ICMP	98	Echo (ping) reply id=0x1bec, seq=1/256, ttl=63 (request in…
174	269.695490615	172.16.40.1	172.16.41.1	ICMP	98	Echo (ping) request id=0x1bec, seq=2/512, ttl=64 (reply in 1…
175	269.695632951	172.16.41.1	172.16.40.1	ICMP	98	Echo (ping) reply id=0x1bec, seq=2/512, ttl=63 (request in…
176	270.677792575	Cisco_d4:1c:05	Spanning-tree-(for-…	STP	60	Conf. Root = 32768/40/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8…
177	270.719474142	172.16.40.1	172.16.41.1	ICMP	98	Echo (ping) request id=0x1bec, seq=3/768, ttl=64 (reply in 1…
178	270.719615640	172.16.41.1	172.16.40.1	ICMP	98	Echo (ping) reply id=0x1bec, seq=3/768, ttl=63 (request in…
179	271.282063978	Cisco_d4:1c:05	Cisco_d4:1c:05	LOOP	60	Reply
180	271.743470869	172.16.40.1	172.16.41.1	ICMP	98	Echo (ping) request id=0x1bec, seq=4/1024, ttl=64 (reply in …
181	271.743611180	172.16.41.1	172.16.40.1	ICMP	98	Echo (ping) reply id=0x1bec, seq=4/1024, ttl=63 (request i…
182	272.703297275	Cisco_d4:1c:05	Spanning-tree-(for-…	STP	60	Conf. Root = 32768/40/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8…
183	272.767456073	172.16.40.1	172.16.41.1	ICMP	98	Echo (ping) request id=0x1bec, seq=5/1280, ttl=64 (reply in …
184	272.767618733	172.16.41.1	172.16.40.1	ICMP	98	Echo (ping) reply id=0x1bec, seq=5/1280, ttl=63 (request i…
185	273.791459994	172.16.40.1	172.16.41.1	ICMP	98	Echo (ping) request id=0x1bec, seq=6/1536, ttl=64 (reply in …
186	273.791603238	172.16.41.1	172.16.40.1	ICMP	98	Echo (ping) reply id=0x1bec, seq=6/1536, ttl=63 (request i…
187	273.866190074	HewlettP_5a:7b:ea	HewlettP_61:2f:d4	ARP	42	Who has 172.16.40.1? Tell 172.16.40.254
188	273.866303915	HewlettP_61:2f:d4	HewlettP_5a:7b:ea	ARP	60	172.16.40.1 is at 00:21:5a:61:2f:d4
189	274.709585346	Cisco_d4:1c:05	Spanning-tree-(for-…	STP	60	Conf. Root = 32768/40/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8…
190	274.815437933	172.16.40.1	172.16.41.1	ICMP	98	Echo (ping) request id=0x1bec, seq=7/1792, ttl=64 (reply in …
191	274.815573495	172.16.41.1	172.16.40.1	ICMP	98	Echo (ping) reply id=0x1bec, seq=7/1792, ttl=63 (request i…
192	275.839437734	172.16.40.1	172.16.41.1	ICMP	98	Echo (ping) request id=0x1bec, seq=8/2048, ttl=64 (reply in …
193	275.839577765	172.16.41.1	172.16.40.1	ICMP	98	Echo (ping) reply id=0x1bec, seq=8/2048, ttl=63 (request i…

Figura 13: Captura apartir de tux44 (interface eth0)

165	266.251312641	Kye_25:1a:f4	Broadcast	ARP	42	Who has 172.16.41.1? Tell 172.16.41.253
166	266.251429276	HewlettP_d7:45:c4	Kye_25:1a:f4	ARP	60	172.16.41.1 is at 00:1f:29:d7:45:c4
167	266.251436120	172.16.40.1	172.16.41.1	ICMP	98	Echo (ping) request id=0x1bec, seq=1/256, ttl=63 (reply in 1…
168	266.251538717	172.16.41.1	172.16.40.1	ICMP	98	Echo (ping) reply id=0x1bec, seq=1/256, ttl=64 (request in…
169	266.685188743	Cisco_d4:1c:07	Spanning-tree-(for-…	STP	60	Conf. Root = 32768/41/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8…
170	267.254452940	172.16.40.1	172.16.41.1	ICMP	98	Echo (ping) request id=0x1bec, seq=2/512, ttl=63 (reply in 1…
171	267.254566292	172.16.41.1	172.16.40.1	ICMP	98	Echo (ping) reply id=0x1bec, seq=2/512, ttl=64 (request in…
172	268.278436886	172.16.40.1	172.16.41.1	ICMP	98	Echo (ping) request id=0x1bec, seq=3/768, ttl=63 (reply in 1…
173	268.278549819	172.16.41.1	172.16.40.1	ICMP	98	Echo (ping) reply id=0x1bec, seq=3/768, ttl=64 (request in…
174	268.689703829	Cisco_d4:1c:07	Spanning-tree-(for-…	STP	60	Conf. Root = 32768/41/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8…
175	268.841136442	Cisco_d4:1c:07	Cisco_d4:1c:07	LOOP	60	Reply
176	269.302433194	172.16.40.1	172.16.41.1	ICMP	98	Echo (ping) request id=0x1bec, seq=4/1024, ttl=63 (reply in …
177	269.302544940	172.16.41.1	172.16.40.1	ICMP	98	Echo (ping) reply id=0x1bec, seq=4/1024, ttl=64 (request i…
178	270.326415325	172.16.40.1	172.16.41.1	ICMP	98	Echo (ping) request id=0x1bec, seq=5/1280, ttl=63 (reply in …
179	270.326552842	172.16.41.1	172.16.40.1	ICMP	98	Echo (ping) reply id=0x1bec, seq=5/1280, ttl=64 (request i…
180	270.696410457	Cisco_d4:1c:07	Spanning-tree-(for-…	STP	60	Conf. Root = 32768/41/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8…
181	271.350422179	172.16.40.1	172.16.41.1	ICMP	98	Echo (ping) request id=0x1bec, seq=6/1536, ttl=63 (reply in …
182	271.350537067	172.16.41.1	172.16.40.1	ICMP	98	Echo (ping) reply id=0x1bec, seq=6/1536, ttl=64 (request i…
183	271.498012206	HewlettP_d7:45:c4	Kye_25:1a:f4	ARP	60	Who has 172.16.41.253? Tell 172.16.41.1
184	271.498028689	Kye_25:1a:f4	HewlettP_d7:45:c4	ARP	42	172.16.41.253 is at 00:c0:df:25:1a:f4
185	272.374396417	172.16.40.1	172.16.41.1	ICMP	98	Echo (ping) request id=0x1bec, seq=7/1792, ttl=63 (reply in …
186	272.374507744	172.16.41.1	172.16.40.1	ICMP	98	Echo (ping) reply id=0x1bec, seq=7/1792, ttl=64 (request i…
187	272.699531096	Cisco_d4:1c:07	Spanning-tree-(for-…	STP	60	Conf. Root = 32768/41/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8…
188	273.398399709	172.16.40.1	172.16.41.1	ICMP	98	Echo (ping) request id=0x1bec, seq=8/2048, ttl=63 (reply in …
189	273.398512084	172.16.41.1	172.16.40.1	ICMP	98	Echo (ping) reply id=0x1bec, seq=8/2048, ttl=64 (request i…
190	274.422387217	172.16.40.1	172.16.41.1	ICMP	98	Echo (ping) request id=0x1bec, seq=9/2304, ttl=63 (reply in …
191	274.422496519	172.16.41.1	172.16.40.1	ICMP	98	Echo (ping) reply id=0x1bec, seq=9/2304, ttl=64 (request i…

Figura 14: Captura apartir de tux44 (interface eth1)

6.6 Experiência 4

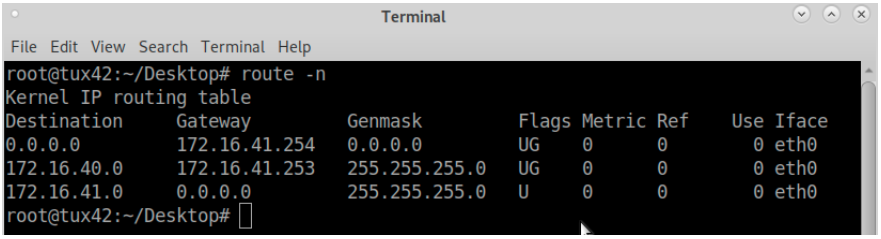


Figura 15: Verificação de rotas no tux42

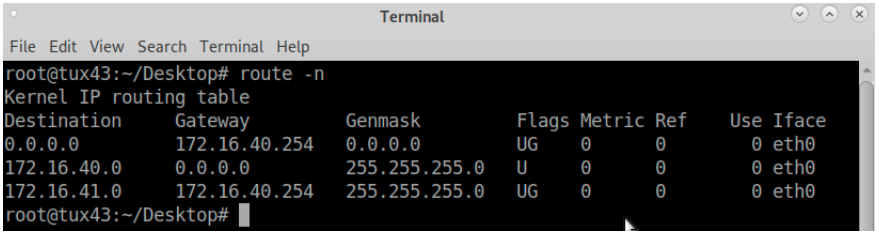


Figura 16: Verificação de rotas no tux43

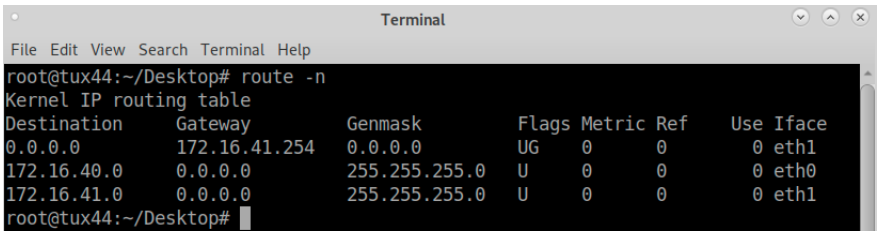


Figura 17: Verificação de rotas no tux44

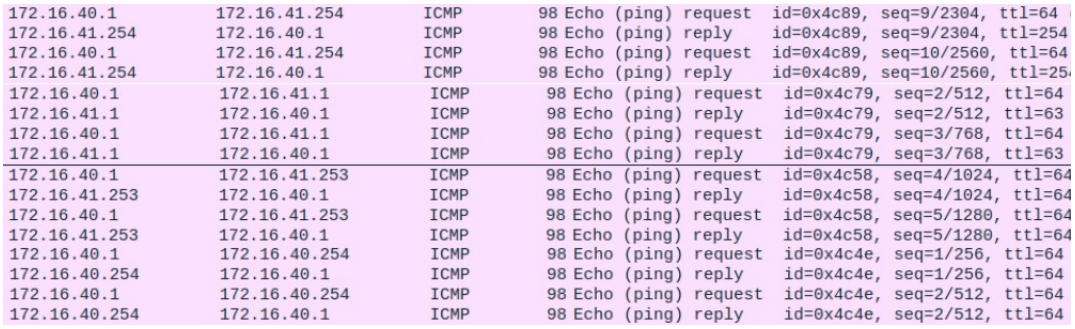


Figura 18: Ping para todas as interfaces de tux44, tux42 e Rc a partir de tux43

16	22.090015668	HewlettP_d7:45:c4	Broadcast	ARP	42	Who has 172.16.41.254? Tell 172.16.41.1
17	22.090430184	Cisco_e3:df:10	HewlettP_d7:45:c4	ARP	60	172.16.41.254 is at 68:ef:bd:e3:df:10
18	22.090444641	172.16.41.1	172.16.40.1	ICMP	98	Echo (ping) request id=0x2443, seq=1/256, ttl=64 (no respons…
19	22.090919850	172.16.41.254	172.16.41.1	ICMP	70	Redirect (Redirect for host)
20	22.090942549	172.16.41.1	172.16.40.1	ICMP	98	Echo (ping) request id=0x2443, seq=1/256, ttl=63 (reply in 2…
21	22.091160807	172.16.40.1	172.16.41.1	ICMP	98	Echo (ping) reply id=0x2443, seq=1/256, ttl=63 (request in…
22	22.129996028	Cisco_d4:1c:04	Cisco_d4:1c:04	LOOP	60	Reply
23	23.091240075	172.16.41.1	172.16.40.1	ICMP	98	Echo (ping) request id=0x2443, seq=2/512, ttl=64 (reply in 2…
24	23.091552411	172.16.41.254	172.16.41.1	ICMP	70	Redirect (Redirect for host)
25	23.091747412	172.16.40.1	172.16.41.1	ICMP	98	Echo (ping) reply id=0x2443, seq=2/512, ttl=63 (request in…
26	24.059176651	Cisco_d4:1c:04	Spanning-tree-(for-…	STP	60	Conf. Root = 32768/41/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8…
27	24.107032046	172.16.41.1	172.16.40.1	ICMP	98	Echo (ping) request id=0x2443, seq=3/768, ttl=64 (reply in 2…
28	24.107315747	172.16.41.254	172.16.41.1	ICMP	70	Redirect (Redirect for host)
29	24.107510468	172.16.40.1	172.16.41.1	ICMP	98	Echo (ping) reply id=0x2443, seq=3/768, ttl=63 (request in…
30	25.131025287	172.16.41.1	172.16.40.1	ICMP	98	Echo (ping) request id=0x2443, seq=4/1024, ttl=64 (reply in …
31	25.131319883	172.16.41.254	172.16.41.1	ICMP	70	Redirect (Redirect for host)
32	25.131537513	172.16.40.1	172.16.41.1	ICMP	98	Echo (ping) reply id=0x2443, seq=4/1024, ttl=63 (request i…
33	26.064056478	Cisco_d4:1c:04	Spanning-tree-(for-…	STP	60	Conf. Root = 32768/41/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8…
34	26.155032007	172.16.41.1	172.16.40.1	ICMP	98	Echo (ping) request id=0x2443, seq=5/1280, ttl=64 (reply in …
35	26.155387576	172.16.41.254	172.16.41.1	ICMP	70	Redirect (Redirect for host)
36	26.155587885	172.16.40.1	172.16.41.1	ICMP	98	Echo (ping) reply id=0x2443, seq=5/1280, ttl=63 (request i…
37	27.179039915	172.16.41.1	172.16.40.1	ICMP	98	Echo (ping) request id=0x2443, seq=6/1536, ttl=64 (reply in …
38	27.179338981	172.16.41.254	172.16.41.1	ICMP	70	Redirect (Redirect for host)
39	27.179564713	172.16.40.1	172.16.41.1	ICMP	98	Echo (ping) reply id=0x2443, seq=6/1536, ttl=63 (request i…
40	27.201800438	Kye_25:1a:f4	HewlettP_d7:45:c4	ARP	60	Who has 172.16.41.1? Tell 172.16.41.253
41	27.201808470	HewlettP_d7:45:c4	Kye_25:1a:f4	ARP	42	172.16.41.1 is at 00:1f:29:d7:45:c4

Figura 19: Ping tux43 a partir de tux42 após ser removida a rota para 172.16.40.0/24 via tux44

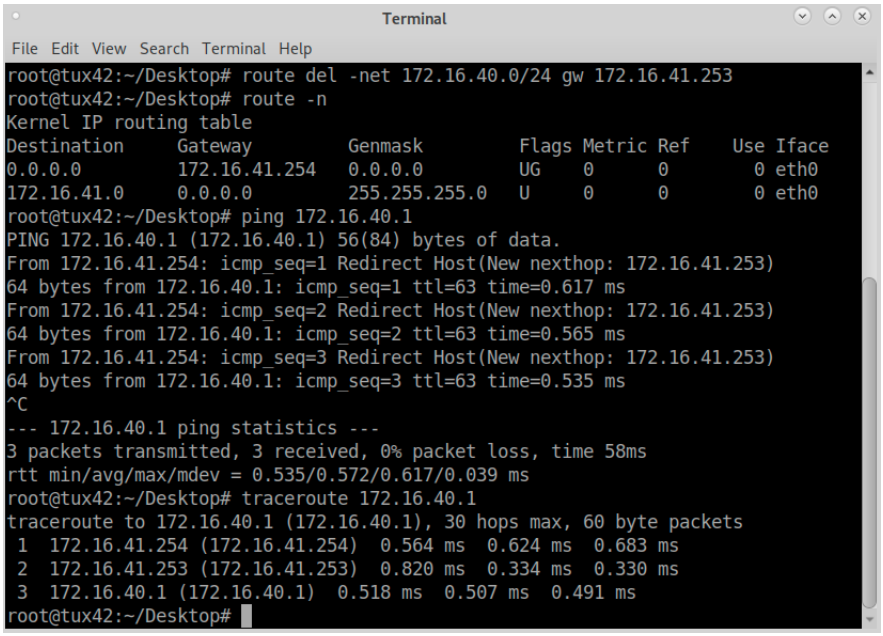


Figura 20: Execução de traceroute em tux 42 após ser removida a rota para 172.16.40.0/24 via tux44

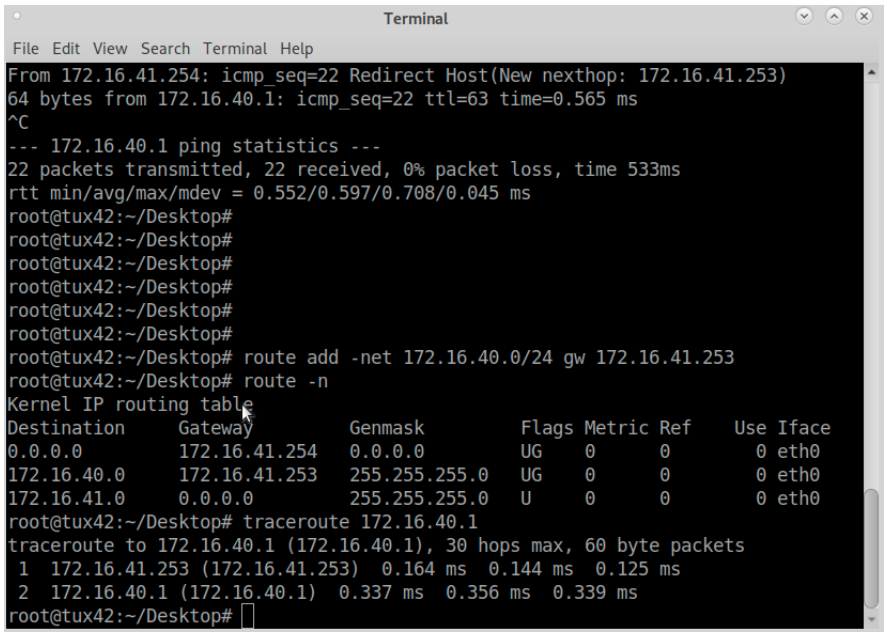
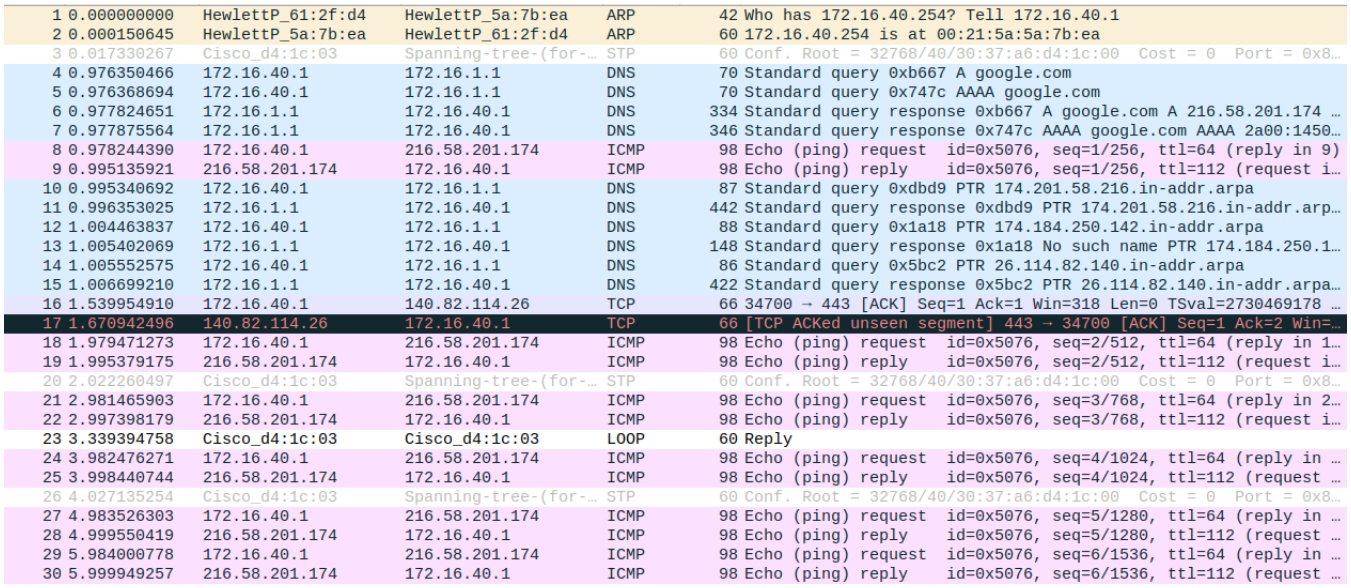


Figura 21: Execução de traceroute em tux 42 adicionando novamente a rota para 172.16.40.0/24 via tux44

1	0.000000000	Cisco_d4:1c:03	Spanning-tree-(for-…	STP	60	Conf. Root = 32768/40/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8…
2	1.416508416	172.16.40.1	172.16.1.254	ICMP	98	Echo (ping) request id=0x4f9c, seq=1/256, ttl=64 (reply in 3)
3	1.417241040	172.16.1.254	172.16.40.1	ICMP	98	Echo (ping) reply id=0x4f9c, seq=1/256, ttl=62 (request in…
4	1.550935384	172.16.40.1	172.16.1.1	DNS	86	Standard query 0xa9b0 PTR 26.114.82.140.in-addr.arpa
5	1.552405171	172.16.1.1	172.16.40.1	DNS	422	Standard query response 0xa9b0 PTR 26.114.82.140.in-addr.arpa…
6	1.671026461	HewlettP_5a:7b:ea	HewlettP_61:2f:d4	ARP	60	Who has 172.16.40.1? Tell 172.16.40.254
7	1.671047064	HewlettP_61:2f:d4	HewlettP_5a:7b:ea	ARP	42	172.16.40.1 is at 00:21:5a:61:2f:d4
8	1.999894436	Cisco_d4:1c:03	Spanning-tree-(for-…	STP	60	Conf. Root = 32768/40/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8…
9	2.422465272	172.16.40.1	172.16.1.254	ICMP	98	Echo (ping) request id=0x4f9c, seq=2/512, ttl=64 (reply in 1…
10	2.423080355	172.16.1.254	172.16.40.1	ICMP	98	Echo (ping) reply id=0x4f9c, seq=2/512, ttl=62 (request in…
11	3.446468062	172.16.40.1	172.16.1.254	ICMP	98	Echo (ping) request id=0x4f9c, seq=3/768, ttl=64 (reply in 1…
12	3.447052346	172.16.1.254	172.16.40.1	ICMP	98	Echo (ping) reply id=0x4f9c, seq=3/768, ttl=62 (request in…
13	3.777987516	Cisco_d4:1c:03	Cisco_d4:1c:03	LOOP	60	Reply
14	4.004447602	Cisco_d4:1c:03	Spanning-tree-(for-…	STP	60	Conf. Root = 32768/40/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8…
15	4.470463302	172.16.40.1	172.16.1.254	ICMP	98	Echo (ping) request id=0x4f9c, seq=4/1024, ttl=64 (reply in …
16	4.471055128	172.16.1.254	172.16.40.1	ICMP	98	Echo (ping) reply id=0x4f9c, seq=4/1024, ttl=62 (request i…
17	5.494464120	172.16.40.1	172.16.1.254	ICMP	98	Echo (ping) request id=0x4f9c, seq=5/1280, ttl=64 (reply in …
18	5.495090028	172.16.1.254	172.16.40.1	ICMP	98	Echo (ping) reply id=0x4f9c, seq=5/1280, ttl=62 (request i…
19	6.009777571	Cisco_d4:1c:03	Spanning-tree-(for-…	STP	60	Conf. Root = 32768/40/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8…
20	6.518468423	172.16.40.1	172.16.1.254	ICMP	98	Echo (ping) request id=0x4f9c, seq=6/1536, ttl=64 (reply in …
21	6.519029729	172.16.1.254	172.16.40.1	ICMP	98	Echo (ping) reply id=0x4f9c, seq=6/1536, ttl=62 (request i…
22	6.554921170	172.16.40.1	172.16.1.1	DNS	86	Standard query 0x33cb PTR 26.114.82.140.in-addr.arpa
23	6.556347935	172.16.1.1	172.16.40.1	DNS	422	Standard query response 0x33cb PTR 26.114.82.140.in-addr.arpa…
24	7.542466363	172.16.40.1	172.16.1.254	ICMP	98	Echo (ping) request id=0x4f9c, seq=7/1792, ttl=64 (reply in …
25	7.543080468	172.16.1.254	172.16.40.1	ICMP	98	Echo (ping) reply id=0x4f9c, seq=7/1792, ttl=62 (request i…

Figura 22: Ping 172.16.1.254 a partir de tux43 com NAT

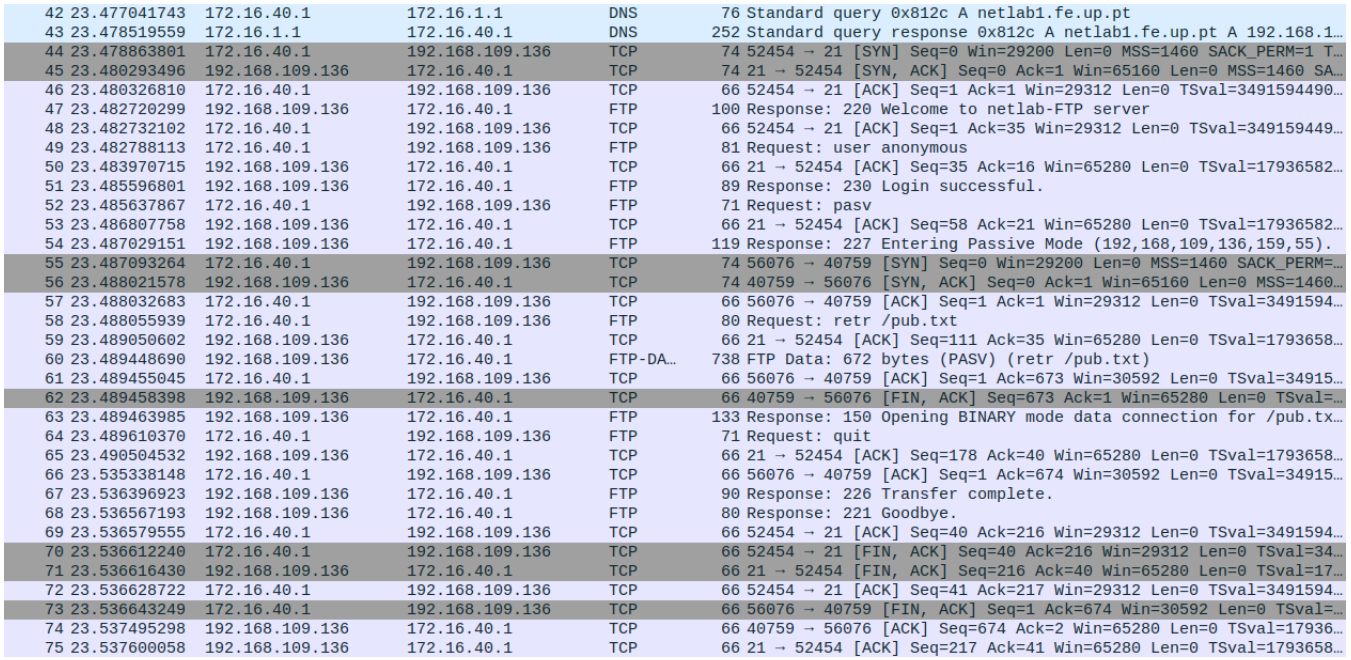
6.7 Experiência 5



1	0.000000000	HewlettP_61:2f:d4	HewlettP_5a:7b:ea	ARP	42	Who has 172.16.40.254? Tell 172.16.40.1
2	0.000150645	HewlettP_5a:7b:ea	HewlettP_61:2f:d4	ARP	60	172.16.40.254 is at 00:21:5a:5a:7b:ea
3	0.017330267	Cisco_d4:1c:03	Spanning-tree-(for-…	STP	60	Conf. Root = 32768/40/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8…
4	0.976350466	172.16.40.1	172.16.1.1	DNS	70	Standard query 0xb667 A google.com
5	0.976368694	172.16.40.1	172.16.1.1	DNS	70	Standard query 0x747c AAAA google.com
6	0.977824651	172.16.1.1	172.16.40.1	DNS	334	Standard query response 0xb667 A google.com A 216.58.201.174 …
7	0.977875564	172.16.1.1	172.16.40.1	DNS	346	Standard query response 0x747c AAAA google.com AAAA 2a00:1450…
8	0.978244390	172.16.40.1	216.58.201.174	ICMP	98	Echo (ping) request id=0x5076, seq=1/256, ttl=64 (reply in 9)
9	0.995135921	216.58.201.174	172.16.40.1	ICMP	98	Echo (ping) reply id=0x5076, seq=1/256, ttl=112 (request i…
10	0.995340692	172.16.40.1	172.16.1.1	DNS	87	Standard query 0xdbd9 PTR 174.201.58.216.in-addr.arpa
11	0.996353025	172.16.1.1	172.16.40.1	DNS	442	Standard query response 0xdbd9 PTR 174.201.58.216.in-addr.arp…
12	1.004463837	172.16.40.1	172.16.1.1	DNS	88	Standard query 0x1a18 PTR 174.184.250.142.in-addr.arpa
13	1.005402069	172.16.1.1	172.16.40.1	DNS	148	Standard query response 0x1a18 No such name PTR 174.184.250.1…
14	1.005552575	172.16.40.1	172.16.1.1	DNS	86	Standard query 0x5bc2 PTR 26.114.82.140.in-addr.arpa
15	1.006699210	172.16.1.1	172.16.40.1	DNS	422	Standard query response 0x5bc2 PTR 26.114.82.140.in-addr.arpa…
16	1.539954910	172.16.40.1	140.82.114.26	TCP	66	34700 → 443 [ACK] Seq=1 Ack=1 Win=318 Len=0 TSval=2730469178 …
17	1.670942496	140.82.114.26	172.16.40.1	TCP	66	[TCP ACKed unseen segment] 443 → 34700 [ACK] Seq=1 Ack=2 Win=…
18	1.979471273	172.16.40.1	216.58.201.174	ICMP	98	Echo (ping) request id=0x5076, seq=2/512, ttl=64 (reply in 1…
19	1.995379175	216.58.201.174	172.16.40.1	ICMP	98	Echo (ping) reply id=0x5076, seq=2/512, ttl=112 (request i…
20	2.022260497	Cisco_d4:1c:03	Spanning-tree-(for-…	STP	60	Conf. Root = 32768/40/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8…
21	2.981465903	172.16.40.1	216.58.201.174	ICMP	98	Echo (ping) request id=0x5076, seq=3/768, ttl=64 (reply in 2…
22	2.997398179	216.58.201.174	172.16.40.1	ICMP	98	Echo (ping) reply id=0x5076, seq=3/768, ttl=112 (request i…
23	3.339394758	Cisco_d4:1c:03	Cisco_d4:1c:03	LOOP	60	Reply
24	3.982476271	172.16.40.1	216.58.201.174	ICMP	98	Echo (ping) request id=0x5076, seq=4/1024, ttl=64 (reply in …
25	3.998440744	216.58.201.174	172.16.40.1	ICMP	98	Echo (ping) reply id=0x5076, seq=4/1024, ttl=112 (request …
26	4.027135254	Cisco_d4:1c:03	Spanning-tree-(for-…	STP	60	Conf. Root = 32768/40/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8…
27	4.983526303	172.16.40.1	216.58.201.174	ICMP	98	Echo (ping) request id=0x5076, seq=5/1280, ttl=64 (reply in …
28	4.999550419	216.58.201.174	172.16.40.1	ICMP	98	Echo (ping) reply id=0x5076, seq=5/1280, ttl=112 (request …
29	5.984000778	172.16.40.1	216.58.201.174	ICMP	98	Echo (ping) request id=0x5076, seq=6/1536, ttl=64 (reply in …
30	5.999949257	216.58.201.174	172.16.40.1	ICMP	98	Echo (ping) reply id=0x5076, seq=6/1536, ttl=112 (request …

Figura 23: Captura da execução de ping google.com a partir de tux43

6.8 Experiência 6



42	23.477041743	172.16.40.1	172.16.1.1	DNS	76	Standard query 0x812c A netlab1.fe.up.pt
43	23.478519559	172.16.1.1	172.16.40.1	DNS	252	Standard query response 0x812c A netlab1.fe.up.pt A 192.168.1…
44	23.478863801	172.16.40.1	192.168.109.136	TCP	74	52454 → 21 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 T…
45	23.480293496	192.168.109.136	172.16.40.1	TCP	74	21 → 52454 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SA…
46	23.480326810	172.16.40.1	192.168.109.136	TCP	66	52454 → 21 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=3491594490…
47	23.482720299	192.168.109.136	172.16.40.1	FTP	100	Response: 220 Welcome to netlab-FTP server
48	23.482732102	172.16.40.1	192.168.109.136	TCP	66	52454 → 21 [ACK] Seq=1 Ack=35 Win=29312 Len=0 TSval=349159449…
49	23.482788113	172.16.40.1	192.168.109.136	FTP	81	Request: user anonymous
50	23.483970715	192.168.109.136	172.16.40.1	TCP	66	21 → 52454 [ACK] Seq=35 Ack=16 Win=65280 Len=0 TSval=17936582…
51	23.485596801	192.168.109.136	172.16.40.1	FTP	89	Response: 230 Login successful.
52	23.485637867	172.16.40.1	192.168.109.136	FTP	71	Request: pasv
53	23.486807758	192.168.109.136	172.16.40.1	TCP	66	21 → 52454 [ACK] Seq=58 Ack=21 Win=65280 Len=0 TSval=17936582…
54	23.487029151	192.168.109.136	172.16.40.1	FTP	119	Response: 227 Entering Passive Mode (192,168,109,136,159,55).
55	23.487093264	172.16.40.1	192.168.109.136	TCP	74	56076 → 40759 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=…
56	23.488021578	192.168.109.136	172.16.40.1	TCP	74	40759 → 56076 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460…
57	23.488032683	172.16.40.1	192.168.109.136	TCP	66	56076 → 40759 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=3491594…
58	23.488055939	172.16.40.1	192.168.109.136	FTP	80	Request: retr /pub.txt
59	23.489050602	192.168.109.136	172.16.40.1	TCP	66	21 → 52454 [ACK] Seq=111 Ack=35 Win=65280 Len=0 TSval=1793658…
60	23.489448690	192.168.109.136	172.16.40.1	FTP-DA…	738	FTP Data: 672 bytes (PASV) (retr /pub.txt)
61	23.489455045	172.16.40.1	192.168.109.136	TCP	66	56076 → 40759 [ACK] Seq=1 Ack=673 Win=30592 Len=0 TSval=34915…
62	23.489458398	192.168.109.136	172.16.40.1	TCP	66	40759 → 56076 [FIN, ACK] Seq=673 Ack=1 Win=65280 Len=0 TSval=…
63	23.489463985	192.168.109.136	172.16.40.1	FTP	133	Response: 150 Opening BINARY mode data connection for /pub.tx…
64	23.489610370	172.16.40.1	192.168.109.136	FTP	71	Request: quit
65	23.490504532	192.168.109.136	172.16.40.1	TCP	66	21 → 52454 [ACK] Seq=178 Ack=40 Win=65280 Len=0 TSval=1793658…
66	23.535338148	172.16.40.1	192.168.109.136	TCP	66	56076 → 40759 [ACK] Seq=1 Ack=674 Win=30592 Len=0 TSval=34915…
67	23.536396923	192.168.109.136	172.16.40.1	FTP	90	Response: 226 Transfer complete.
68	23.536567193	192.168.109.136	172.16.40.1	FTP	80	Response: 221 Goodbye.
69	23.536579555	172.16.40.1	192.168.109.136	TCP	66	52454 → 21 [ACK] Seq=40 Ack=216 Win=29312 Len=0 TSval=3491594…
70	23.536612240	172.16.40.1	192.168.109.136	TCP	66	52454 → 21 [FIN, ACK] Seq=40 Ack=216 Win=29312 Len=0 TSval=34…
71	23.536616430	192.168.109.136	172.16.40.1	TCP	66	21 → 52454 [FIN, ACK] Seq=216 Ack=40 Win=65280 Len=0 TSval=17…
72	23.536628722	172.16.40.1	192.168.109.136	TCP	66	52454 → 21 [ACK] Seq=41 Ack=217 Win=29312 Len=0 TSval=3491594…
73	23.536643249	172.16.40.1	192.168.109.136	TCP	66	56076 → 40759 [FIN, ACK] Seq=1 Ack=674 Win=30592 Len=0 TSval=…
74	23.537495298	192.168.109.136	172.16.40.1	TCP	66	40759 → 56076 [ACK] Seq=674 Ack=2 Win=65280 Len=0 TSval=17936…
75	23.537600058	192.168.109.136	172.16.40.1	TCP	66	21 → 52454 [ACK] Seq=217 Ack=41 Win=65280 Len=0 TSval=1793658…

Figura 24: Transferência de um ficheiro com recurso à aplicação de download desenvolvida

5829	5.184495281	192.168.109.136	172.16.40.1	FTP-DA…	1514 FTP Data: 1448 bytes (PASV) (retr /files/crab.mp4)
5830	5.184552899	172.16.40.1	192.168.109.136	TCP	94 [TCP Dup ACK 5384#223] 34864 → 43596 [ACK] Seq=1 Ack=5086825 …
5831	5.184618059	192.168.109.136	172.16.40.1	FTP-DA…	1514 FTP Data: 1448 bytes (PASV) (retr /files/crab.mp4)
5832	5.184675049	172.16.40.1	192.168.109.136	TCP	94 [TCP Dup ACK 5384#224] 34864 → 43596 [ACK] Seq=1 Ack=5086825 …
5833	5.184741886	192.168.109.136	172.16.40.1	FTP-DA…	1514 FTP Data: 1448 bytes (PASV) (retr /files/crab.mp4)
5834	5.184799504	172.16.40.1	192.168.109.136	TCP	94 [TCP Dup ACK 5384#225] 34864 → 43596 [ACK] Seq=1 Ack=5086825 …
5835	5.184864315	192.168.109.136	172.16.40.1	FTP-DA…	1514 FTP Data: 1448 bytes (PASV) (retr /files/crab.mp4)
5836	5.184922003	172.16.40.1	192.168.109.136	TCP	94 [TCP Dup ACK 5384#226] 34864 → 43596 [ACK] Seq=1 Ack=5086825 …
5837	5.184987652	192.168.109.136	172.16.40.1	FTP-DA…	1514 FTP Data: 1448 bytes (PASV) (retr /files/crab.mp4)
5838	5.185044991	172.16.40.1	192.168.109.136	TCP	94 [TCP Dup ACK 5384#227] 34864 → 43596 [ACK] Seq=1 Ack=5086825 …
5839	5.185110710	192.168.109.136	172.16.40.1	FTP-DA…	1514 FTP Data: 1448 bytes (PASV) (retr /files/crab.mp4)
5840	5.185167421	172.16.40.1	192.168.109.136	TCP	94 [TCP Dup ACK 5384#228] 34864 → 43596 [ACK] Seq=1 Ack=5086825 …
5841	5.185233419	192.168.109.136	172.16.40.1	TCP	1514 [TCP Out-Of-Order] 43596 → 34864 [ACK] Seq=5189633 Ack=1 Win=…
5842	5.185290339	172.16.40.1	192.168.109.136	TCP	94 [TCP Dup ACK 5384#229] 34864 → 43596 [ACK] Seq=1 Ack=5086825 …
5843	5.185355988	192.168.109.136	172.16.40.1	TCP	1514 [TCP Out-Of-Order] 43596 → 34864 [ACK] Seq=5191081 Ack=1 Win=…
5844	5.185412699	172.16.40.1	192.168.109.136	TCP	94 [TCP Dup ACK 5384#230] 34864 → 43596 [ACK] Seq=1 Ack=5086825 …
5845	5.185479815	192.168.109.136	172.16.40.1	FTP-DA…	1514 FTP Data: 1448 bytes (PASV) (retr /files/crab.mp4)
5846	5.185536176	172.16.40.1	192.168.109.136	TCP	94 [TCP Dup ACK 5384#231] 34864 → 43596 [ACK] Seq=1 Ack=5086825 …
5847	5.185602454	192.168.109.136	172.16.40.1	FTP-DA…	1514 FTP Data: 1448 bytes (PASV) (retr /files/crab.mp4)
5848	5.185658954	172.16.40.1	192.168.109.136	TCP	94 [TCP Dup ACK 5384#232] 34864 → 43596 [ACK] Seq=1 Ack=5086825 …
5849	5.185725232	192.168.109.136	172.16.40.1	FTP-DA…	1514 FTP Data: 1448 bytes (PASV) (retr /files/crab.mp4)
5850	5.185781593	172.16.40.1	192.168.109.136	TCP	94 [TCP Dup ACK 5384#233] 34864 → 43596 [ACK] Seq=1 Ack=5086825 …
5851	5.185848989	192.168.109.136	172.16.40.1	FTP-DA…	1514 FTP Data: 1448 bytes (PASV) (retr /files/crab.mp4)
5852	5.185905210	172.16.40.1	192.168.109.136	TCP	94 [TCP Dup ACK 5384#234] 34864 → 43596 [ACK] Seq=1 Ack=5086825 …
5853	5.185972606	192.168.109.136	172.16.40.1	TCP	1514 [TCP Out-Of-Order] 43596 → 34864 [ACK] Seq=5204113 Ack=1 Win=…
5854	5.186029595	172.16.40.1	192.168.109.136	TCP	94 [TCP Dup ACK 5384#235] 34864 → 43596 [ACK] Seq=1 Ack=5086825 …
5855	5.186094476	192.168.109.136	172.16.40.1	FTP-DA…	1514 FTP Data: 1448 bytes (PASV) (retr /files/crab.mp4)
5856	5.186150907	172.16.40.1	192.168.109.136	TCP	94 [TCP Dup ACK 5384#236] 34864 → 43596 [ACK] Seq=1 Ack=5086825 …
5857	5.186217185	192.168.109.136	172.16.40.1	FTP-DA…	1514 FTP Data: 1448 bytes (PASV) (retr /files/crab.mp4)

Figura 25: Transferência simultânea de dois ficheiros em dois computadores diferentes com recurso à aplicação de download desenvolvida

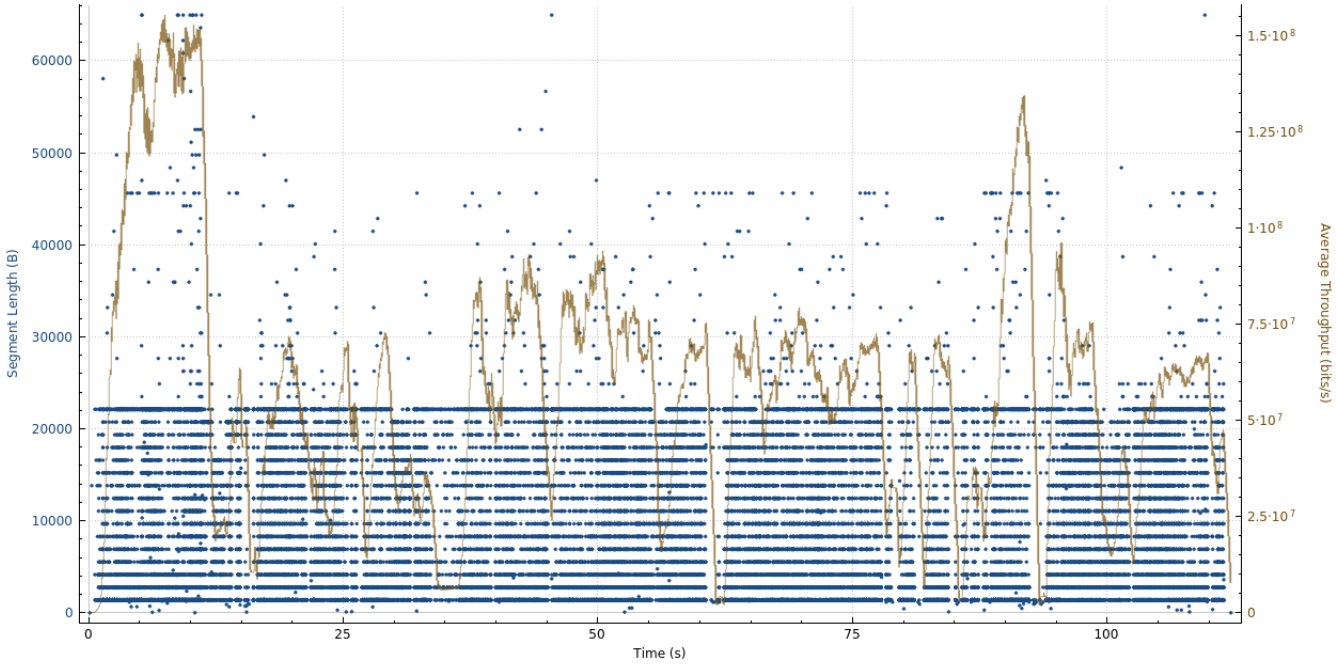


Figura 26: Troughput gerado durante o download simultâneo de dois ficheiros

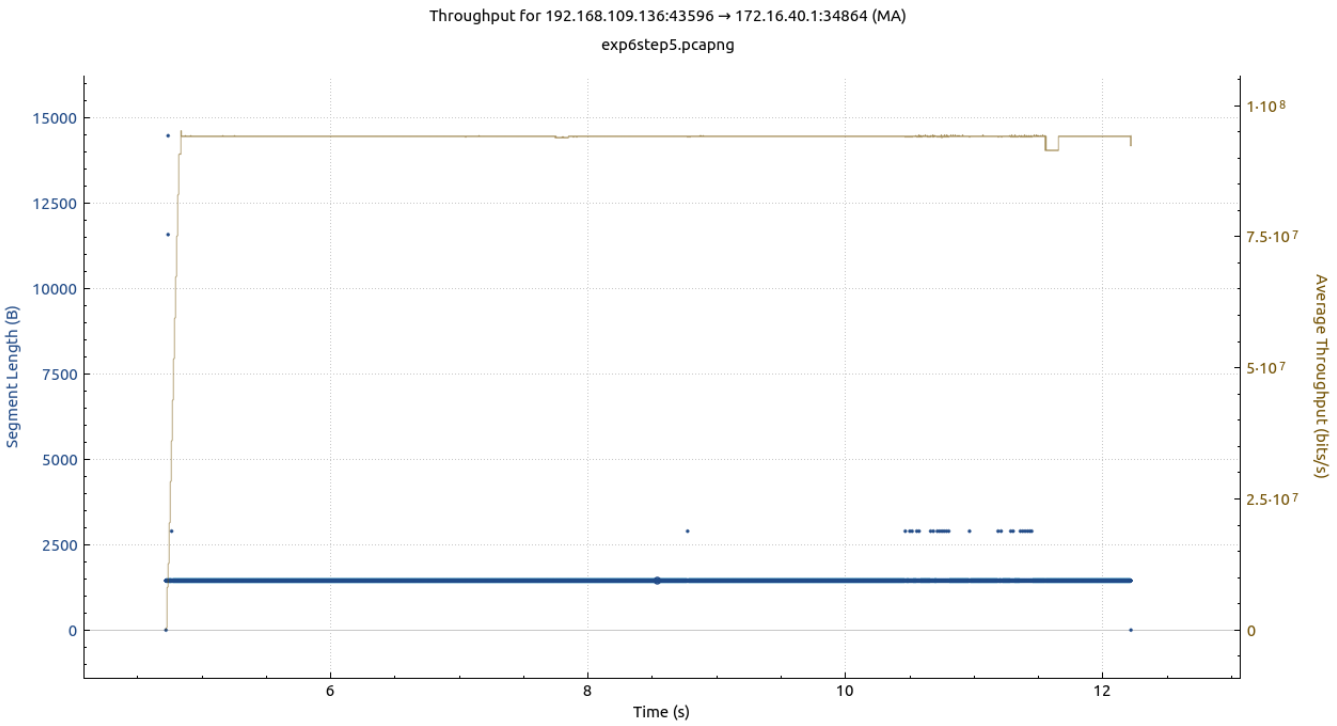


Figura 27: Troughput gerado durante o download simultâneo de dois ficheiros (experiência em laboratório)

Nota: O ficheiro que que foi transferido no tux42 (netlab1.fe.up.pt → pub.txt) tem um tamanho muito reduzido quando comparado com o ficheiro tranferido no tux43 (netlab1.fe.up.pt → files/crab.mp4) fazendo com que a execução simultânea apenas ocorra durante um intervalo de tempo reduzido (um pouco antes dos 12s), gerando uma pequena variação no gráfico que rapidamente regressa ao normal.

6.9 Aplicação Download

```
**************************

Host: netlab1.fe.up.pt
Url: /pub.txt
User: anonymous
Pass: ****

**************************

IP Address : 192.168.109.136

220 Welcome to netlab-FTP server

>user anonymous

230 Login successful.

200 Switching to Binary mode.

>pasv

227 Entering Passive Mode (192,168,109,136,167,132).

IP Address: 192.168.109.136
Port Number: 42884

>retr /pub.txt

150 Opening BINARY mode data connection for /pub.txt (672 bytes).

>quit

226 Transfer complete.
221 Goodbye.

joao@joaodiogo:~/Documents/feup/rcom/TP2/src$
```

Figura 28: Exemplo da execução da aplicação de download

6.9.1 main.c

```
1
2
3  #include "utils.h"
4  #include "stdlib.h"
5  #include "stdio.h"
6  #include "ftp.h"
7  #include "getip.h"
8  #include "string.h"
9  #include <unistd.h>
10
11 #define ARGUMENT_MAX_SIZE 255
12 #define READY_STATE_WELCOME "220 " //space needed for multiple line welcome
       message
13 #define INFO_MSG "230 "
14
15 int main(int argc, char **argv)
16 {
17
18   arguments args;
19   args.user = (char *)malloc(ARGUMENT_MAX_SIZE);
20   args.password = (char *)malloc(ARGUMENT_MAX_SIZE);
21   args.url_path = (char *)malloc(ARGUMENT_MAX_SIZE);
22   args.host = (char *)malloc(ARGUMENT_MAX_SIZE);
23   parse_arguments(&args, argc, argv);
24
25   char *ipAddr = (char *)malloc(20);
26   ipAddr = getHostIP(args.host);
27
28   printf("IP Address : %s\n\n", ipAddr);
29
30   //Establish a TCP connection. This protocol uses port 21 by default
31   int sock_fd = ftp_open_connection(ipAddr, SERVER_PORT); // O que por no sever
       port?
32
33   char buff[MAX_SIZE];
```

```
34
35    // After successful connection, the server sends a line of welcome text, for
        example, 220 welcome.
36    if (ftp_poll_read(sock_fd, READY_STATE_WELCOME, buff) < 0)
37    {
38      printf("Error: Reading welcome message\n");
39      return -1;
40    }
41    //Print Welcome Message
42    printf("%s\n", buff);
43
44    if (ftp_login(sock_fd, args.user, args.password) < 0)
45    {
46      printf("Error: Logging In\n");
47      return -1;
48    }
49
50    if (ftp_set_binary_mode(sock_fd) < 0)
51    {
52      printf("Error setting bynary mode\n");
53      return -1;
54    }
55
56    pasv_info pasv;
57    if (ftp_set_passive_mode(sock_fd, &pasv) < 0)
58    {
59      printf("Error: Setting passive Mode\n");
60      return -1;
61    }
62
63    int recv_fd;
64    if ((recv_fd = ftp_open_connection(pasv.ip_address, pasv.port_number)) < 0)
65    {
66      printf("Error: Opening new connection after passive mode\n");
67      return -1;
68    }
69
70    if (ftp_send_retr(sock_fd, args.url_path) < 0)
71    {
72      printf("Error: Setting passive Mode\n");
73      return -1;
74    }
75
76    if (ftp_retr_file(recv_fd, args.url_path) < 0)
77    {
78      printf("Error: Retrieving file\n");
79      return -1;
80    }
81
82    ftp_close_connection(sock_fd);
83
84    close(sock_fd);
85    close(recv_fd);
86
87    return 0;
88  }
```

6.9.2 ftp.c

```
1
2
3  #include <stdio.h>
4  #include <sys/types.h>
5  #include <sys/socket.h>
6  #include <sys/stat.h>
7  #include <netinet/in.h>
8  #include <arpa/inet.h>
9  #include <stdlib.h>
10 #include <unistd.h>
11 #include <signal.h>
12 #include <netdb.h>
13 #include <strings.h>
14 #include <fcntl.h>
15 #include <string.h>
16
17 #include "ftp.h"
18 #include "utils.h"
19
20 int ftp_open_connection(char *serverAddr, int serverPort)
21 {
22
23     int sock_fd;
24     struct sockaddr_in server_addr;
25
26     /*server address handling*/
27     bzero((char *)&server_addr, sizeof(server_addr));
28     server_addr.sin_family = AF_INET;
29     server_addr.sin_addr.s_addr = inet_addr(serverAddr); /*32 bit Internet address
        network byte ordered*/
30     server_addr.sin_port = htons(serverPort);           /*server TCP port must be
        network byte ordered */
31
32     /*open an TCP socket*/
33     if ((sock_fd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
34     {
35         printf("Error: socket()\n");
36         return -1;
37     }
38     /*connect to the server*/
39     if (connect(sock_fd, (struct sockaddr *)&server_addr, sizeof(server_addr)) <
       0)
40     {
41         perror("connect");
42         return -1;
43     }
44     return sock_fd;
45 }
46
47 int ftp_poll_read(int fd, const char *ready_state, char *buff)
48 {
49     memset(buff, 0, MAX_SIZE * sizeof(char));
50
51     char *ret;
52     int read_ret;
53     while ((ret = strstr(buff, ready_state)) == NULL)
54     {
55         memset(buff, 0, MAX_SIZE * sizeof(char));
56
57         if ((read_ret = read(fd, buff, MAX_SIZE)) == -1)
58         {
59             printf("Error: Reading from socket\n");
60             return -1;
61         }
62
63         if (read_ret == 0)
64         {
65             printf("Error: Could not reach ready state in ftp_read\n");
66             return -1;
67         }
68     }
```

```
69
70      return strlen(buff);
71  }
72
73  int ftp_read(int fd, char *buff)
74  {
75      FILE *socket = fdopen(fd, "r");
76      if (socket == NULL)
77          return -1;
78
79      memset(buff, 0, MAX_SIZE * sizeof(char));
80      size_t n_bytes_read;
81      while (getline(&buff, &n_bytes_read, socket) > 0)
82      {
83          if (strlen(buff) != 0)
84              printf("%s", buff);
85
86          if (buff[3] == ' ')
87          {
88              break;
89          }
90      }
91      printf("\n");
92      return 0;
93  }
94
95  int ftp_write(int sock_fd, char *buf)
96  {
97      int n_bytes = write(sock_fd, buf, strlen(buf));
98      if (n_bytes < 0)
99      {
100         printf("Error: Error write socket message\n");
101         return -1;
102     }
103     return n_bytes;
104 }
105
106 int ftp_login(int sock_fd, char *user, char *pass)
107 {
108     char userMsg[strlen(USER) + strlen(user) + 2];
109     char passMsg[strlen(PASS) + strlen(pass) + 2];
110     sprintf(userMsg, "%s%s\n", USER, user);
111     sprintf(passMsg, "%s%s\n", PASS, pass);
112
113     printf(">%s\n", userMsg);
114
115     // SEND USER
116     if (ftp_write(sock_fd, userMsg) < 0)
117     {
118         printf("Error: Sending user\n");
119         return -1;
120     }
121
122     char buff[MAX_SIZE];
123     // RECEIVE ANSWER
124     if (ftp_read(sock_fd, buff) < 0)
125     {
126         printf("Error: Error receiving answer after sending user message\n");
127         return -1;
128     }
129
130     if (strstr(buff, LOGIN_SUCCESSFUL) != NULL)
131         return 0;
132
133     if (strstr(buff, USER_SUCCESSFUL) == NULL)
134     {
135         printf("Error: Received wrong message after user input\n");
136         return -1;
137     }
138
139     printf(">%s%s\n\n", PASS, hiddenPass(pass));
140
141     // SEND PASS
```

```
142     if (ftp_write(sock_fd, passMsg) < 0)
143     {
144         printf("Error: Sending Password\n");
145         return -1;
146     }
147 
148     // RECEIVE ANSWER
149     if (ftp_read(sock_fd, buff) < 0)
150     {
151         printf("Error: Error receiving answer after sending pass message\n\n");
152         return -1;
153     }
154     if (strstr(buff, INCORRECT_PASS) != NULL)
155     {
156         printf("Error: Incorrect Password\n");
157         return -1;
158     }
159     else if (strstr(buff, LOGIN_SUCCESSFUL) == NULL)
160     {
161         printf("Error: Error Logging In\n");
162         return -1;
163     }
164 
165     return 0;
166 }
167 
168 int ftp_set_passive_mode(int sock_fd, pasv_info *pasv)
169 {
170     char pasvMsg[strlen(PASSIVE_MODE_CMD) + 1];
171     sprintf(pasvMsg, "%s\n", PASSIVE_MODE_CMD);
172 
173     printf(">%s\n", pasvMsg);
174 
175     // SEND USER
176     if (ftp_write(sock_fd, pasvMsg) < 0)
177     {
178         printf("Error: Sending Passive command\n");
179         return -1;
180     }
181     char buff[MAX_SIZE];
182     if (ftp_read(sock_fd, buff) < 0)
183     {
184         printf("Error reading passive mode response\n");
185         return -1;
186     }
187 
188     if (strstr(buff, PASSIVE_MODE_SUCC_CODE) == NULL)
189     {
190         printf("Error setting passive mode\n");
191         return -1;
192     }
193 
194     int ip1, ip2, ip3, ip4, portHigh, portLow;
195 
196     if (sscanf(buff, "227 Entering Passive Mode (%d,%d,%d,%d,%d,%d)", &ip1, &ip2,
    &ip3, &ip4, &portHigh, &portLow) < 0)
197     {
198         printf("Error parsing passive mode msg\n");
199         return -1;
200     }
201     int portNumber = portHigh * 256 + portLow;
202 
203     char *ipAdress = malloc(MAX_SIZE);
204     sprintf(ipAdress, "%d.%d.%d.%d", ip1, ip2, ip3, ip4);
205     printf("IP Address: %s\n", ipAdress);
206     printf("Port Number: %d\n\n", portNumber);
207 
208     pasv->ip_address = ipAdress;
209     pasv->port_number = portNumber;
210 
211     return 0;
212 }
213 
```

```
214 int ftp_send_retr(int sock_fd, char *path)
215 {
216     char retrMsg[strlen(RETR_CMD) + strlen(path) + 1];
217     sprintf(retrMsg, "%s%s\n", RETR_CMD, path);
218 
219     printf(">%s\n", retrMsg);
220 
221     // SEND RETR
222     if (ftp_write(sock_fd, retrMsg) < 0)
223     {
224         printf("Error: Sending RETR Command\n");
225         return -1;
226     }
227     char buff[MAX_SIZE];
228     if (ftp_read(sock_fd, buff) < 0)
229     {
230         printf("Error reading RETR Response\n");
231         return -1;
232     }
233     return 0;
234 }
235 
236 int ftp_retr_file(int sock_fd, char *path)
237 {
238 
239     char reversed[strlen(path) + 1];
240     memset(reversed, 0, strlen(path) + 1);
241 
242     int counter = 0;
243     for (int i = strlen(path) - 1; i >= 0; i--)
244     {
245         if (path[i] == '/')
246             break;
247         reversed[counter++] = path[i];
248     }
249     char *path_copy = strrev(reversed);
250 
251     int fd;
252     if ((fd = open(path_copy, O_WRONLY | O_CREAT, 0660)) < 0)
253     {
254         perror("Error creating new file");
255         return -1;
256     }
257 
258     char buff[MAX_SIZE];
259     int bytes_read = 0;
260     while ((bytes_read = read(sock_fd, buff, MAX_SIZE)) > 0)
261     {
262         if (write(fd, buff, bytes_read) < 0)
263         {
264             perror("Error writing to new file");
265             return -1;
266         }
267     }
268 
269     if (bytes_read < 0)
270     {
271         printf("Error reading file\n");
272         return -1;
273     }
274 
275     if (close(fd) == -1)
276     {
277         perror("Error closing new file descriptor");
278         return -1;
279     }
280     return 0;
281 }
282 
283 int ftp_set_binary_mode(int sock_fd)
284 {
285 
286     char bynaryCmd[strlen(BINARY_COMMAND) + 1];
```

```
287     sprintf(bynaryCmd, "%s\n", BINARY_COMMAND);
288 
289     // SEND USER
290     if (ftp_write(sock_fd, bynaryCmd) < 0)
291     {
292         printf("Error: Sending Binary command\n");
293         return -1;
294     }
295     char buff[MAX_SIZE];
296     if (ftp_read(sock_fd, buff) < 0)
297     {
298         printf("Error reading Bynary mode response\n");
299         return -1;
300     }
301 
302     if (strstr(buff, BYNARY_SUCCESS) == NULL)
303     {
304         printf("Error setting binary mode\n");
305         return -1;
306     }
307     return 0;
308 }
309 
310 int ftp_close_connection(int sock_fd)
311 {
312     char closeMsg[MAX_SIZE];
313     sprintf(closeMsg, "%s\n", QUIT_CMD);
314 
315     printf(">%s\n", closeMsg);
316 
317     if (write(sock_fd, closeMsg, strlen(closeMsg)) < 0)
318     {
319         perror("Closing connection");
320         return -1;
321     }
322 
323     char buff[MAX_SIZE];
324     ftp_poll_read(sock_fd, QUIT_SUCCESSFUL, buff);
325     printf("%s\n", buff);
326 
327     return 0;
328 }
```

6.9.3 getip.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <errno.h>
4  #include <netdb.h>
5  #include <sys/types.h>
6  #include <netinet/in.h>
7  #include<arpa/inet.h>
8 
9  char * getHostIP(char * hostName)
10 {
11   struct hostent *h;
12 
13   if ((h=gethostbyname(hostName)) == NULL) {
14     herror("gethostbyname");
15     exit(1);
16   }
17 
18   return inet_ntoa(*((struct in_addr *)h->h_addr));
19 }
```

6.9.4 utils.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include "utils.h"
#include <assert.h>

#define ARGUMENT_FTP "ftp://"
#define ARGUMENT_POSITION 1
#define ANONYMOUS_USER "anonymous"
#define ANONYMOUS_PASS "something"

void parse_arguments(arguments *args, int argc, char *argv[])
{
    if (!check_arg(argc, argv))
    {
        printf("Usage: ftp://[<user>:<password>@]<host>/<url-path>\n");
        exit(1);
    }
    printf("Parsed arguments correctly\n");

    int indexUserPasswordEnd = hasUserPassword(argv[ARGUMENT_POSITION]);
    int hostIndex = strlen(ARGUMENT_FTP);
    if (indexUserPasswordEnd > 0)
        hostIndex = parseUserPassword(argv[ARGUMENT_POSITION], args,
    indexUserPasswordEnd);
    else
    {
        args->user = ANONYMOUS_USER;
        args->password = ANONYMOUS_PASS;
    }

    int hostSize = 0;
    for (int i = hostIndex; i < strlen(argv[1]); i++)
    {
        if (argv[ARGUMENT_POSITION][i] == '/')
        {
            strncpy(args->host, argv[ARGUMENT_POSITION] + hostIndex, hostSize);
            break;
        }
        hostSize++;
    }
    strcpy(args->url_path, argv[ARGUMENT_POSITION] + hostIndex + hostSize);

    printSeparator();
    printf("Host: %s\n", args->host);
    printf("Url: %s\n", args->url_path);
    printf("User: %s\n", args->user);
    printf("Pass: %s\n", hiddenPass(args->password));
    printSeparator();
}

bool check_arg(int argc, char *argv[])
{
    int initialFtp = strncmp(argv[ARGUMENT_POSITION], ARGUMENT_FTP, strlen(
    ARGUMENT_FTP) - 1);
    return (argc == 2) && (initialFtp == 0); //Ver se    igual a ftp://
}

int hasUserPassword(char *str)
{
    for (int i = strlen(ARGUMENT_FTP); i < strlen(str); i++)
        if (str[i] == '@')
            return i;

    return -1;
}

int parseUserPassword(char *str, arguments *args, int userPasswordEnd)
{
    int userSize = 0;
```

```
70      int passwordSize = 0;
71      int passwordBeggining = 0;
72      for (int i = strlen(ARGUMENT_FTP); i < userPasswordEnd; i++)
73      {
74          if (str[i] == ':')
75          {
76              strncpy(args->user, str + strlen(ARGUMENT_FTP), userSize);
77              userSize++;
78              break;
79          }
80          userSize++;
81      }
82
83      passwordBeggining = strlen(ARGUMENT_FTP) + userSize;
84      for (int i = passwordBeggining; i < userPasswordEnd + passwordBeggining; i++)
85      {
86
87          if (str[i] == '@')
88          {
89              strncpy(args->password, str + passwordBeggining, passwordSize);
90              break;
91          }
92          passwordSize++;
93      }
94      return passwordBeggining + passwordSize + 1;
95  }
96
97  void printSeparator()
98  {
99      printf("\n**************************\n\n");
100 }
101
102 char* strrev(char* str)
103 {
104     char *p1, *p2;
105
106     if (!str || !*str)
107         return str;
108     for (p1 = str, p2 = str + strlen(str) - 1; p2 > p1; ++p1, --p2)
109     {
110         *p1 ^= *p2;
111         *p2 ^= *p1;
112         *p1 ^= *p2;
113     }
114     return str;
115 }
116
117 char* hiddenPass(char* pass)
118 {
119     char* buff = malloc(strlen(pass) + 1);
120     for (int i = 0; i < strlen(pass); i++)
121         buff[i] = '*';
122     return buff;
123 }
```