Configuração de uma rede e desenvolvimento de uma aplicação de download

Redes de Computadores

2º trabalho laboratorial - T6G3

23/12/2019

António Dantas up201703878@fe.up.pt
Eduardo Macedo up201703658@fe.up.pt
Vítor Gonçalves up201703917@fe.up.pt

Sumário

No âmbito da unidade curricular Redes de Computadores, foi-nos proposto o desenvolvimento de uma aplicação de download de acordo com o protocolo FTP e a configuração de uma rede, bem como o estudo da mesma.

No fim da implementação de tudo o que nos foi pedido no enunciado, é possível concluir que a aplicação faz download de um ficheiro de um server FTP, com sucesso e sem perdas de conteúdo, através de uma rede configurada por nós, pelo que o desenvolvimento foi efetuado com sucesso.

Introdução

O objetivo deste 2º trabalho laboratorial de Redes de Computadores passa pelo desenvolvimento de uma aplicação de download de acordo com o protocolo ***FTP(File Transfer Protocol)*** com ajuda de ligações ***TCP(Transmission Control Protocol),*** através de ***sockets***.

Numa 2º parte foi pedido a configuração e análise de uma rede de computadores que permitisse a execução da aplicação referida em cima, a partir de duas ***VLAN's*** dentro de um ***switch***, sendo que numa delas foi implementado o ***NAT***.

A informação presente no relatório estará distribuída da seguinte forma:

- **Parte 1 - Aplicação de download**
  - Arquitetura da aplicação e resultados obtidos.
- **Parte 2 - Configuração e estudo de uma rede**
  - Análise das diversas experiências efetuadas.
- **Conclusão**
  - Análise final do trabalho prático e reflexão sobre os objetivos alcançados.
- **Referências**
  - Fontes utilizadas durante todo o trabalho prático 2 e escrita do relatório.
- **Anexos**
  - Secção onde estarão presentes as diversas figuras que vão servir de suporte para a escrita do relatório, como por exemplo imagens do código da aplicação, comandos de configuração e logs gravados ao longos das experiências realizadas.

Parte 1 - Aplicação de download

A parte inicial do 2º trabalho prático consistia no desenvolvimento de uma pequena aplicação de download , que recebe como argumento um link que contém vária informação necessária para descarregar um ficheiro de um servidor FTP.

Uma vez que foi a primeira vez que todos os elementos do grupo tiveram contacto com este tipo de aplicação, foi necessário a consulta a dois documentos que falam sobre o **FTP** e a informação contido nos **URL's, RFC959** e **RFC1738** respetivamente.

A linguagem de programação usada foi **C**.

Arquitetura

Para a implementação da aplicação é **inicialmente** feita a análise e extração da informação presente no argumento de entrada, **um link**, para variavéis criadas.

O argumento vem com o seguinte formato:

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

A aplicação pode ser usada em dois modos, o normal e o anónimo. Para ambos os modos o utilizador tem de passar ao URL a nome do **user** e a sua **password**. Em modo **anónimo** o user será **anonymous** e a password poderá ser uma qualquer. ***(ver figura 1, presente nos Anexos)***

Após ser obtida toda a informação proveniente do **URL,** é necessário obter o **ip** do servidor em questão. Para tal, foi utilizado o código fornecido pelos docentes no início do trabalho prático. O código em questão recebe um ***hostname*** e retorna uma estrutura de dados do tipo **hostent**, que sofrerá um cast para uma estrutura do tipo **in_addr** na função **inet_ntoa** que retorna um **char*** que representa o **ip** do servidor.

De seguida, é necessário ligar o cliente **FTP** ao servidor passado no URL usando para isso a ajuda de um ***socket TCP.*** Para tal, a porta usada é sempre a 21.

Após a abertura do socket são enviados ao servidor dois comandos para efetuar o login do utilizador, **user user e pass password.**

Imediatamente após, é enviado o comando **pasv,** que permite entrada em modo passivo. Em **resposta** a este comando, o servidor responde com informação necessária para obter o **número da porta** para **abertura de um novo socket** que servirá apenas para a **transferência de dados**.

Por fim, é mandado ao servidor o comando **retr filename** onde é pedido ao servidor o download do ficheiro passado como argumento.

Após serem percorridos todos estes comandos, é efetuada a transferência do ficheiro que o utilizador especifica no link, aquando da chamada à aplicação.

Por fim, ambas as conexões são encerradas.

Resultados

Para testar a aplicação desenvolvida foram efetuados diversos testes onde foram feitos downloads de diversos ficheiros, tanto em modo anónimo como em modo normal, com diferentes tamanhos. Foram também introduzidos erros no nosso URL de modo a verificar a qualidade do nosso parser. ***(ver figuras 2, 3, 4, 5 do Anexo)***

De salientar que durante estes testes nos deparamos com um erro na nossa aplicação. Quando tentamos fazer a transferência de um ficheiro do qual não temos permissões a nossa aplicação não consegue verificar isso mesmo, ou seja, não nos permite fazer download do ficheiro mas também não termina, ficando bloqueada e apenas encerra passado um determinado tempo.

Tirando este pormenor a aplicação tem o comportamento esperado nos restantes casos.

Parte 2 - Configuração e estudo de uma rede

Experiência 1 - Configurar um IP de rede

Nesta primeira experiência foi ligado o tux1 ao tux4 recorrendo ao switch.

- **O que são pacotes ARP e para que servem?**
  O ARP, ou *Address Resolution Protocol*, é um protocolo de comunicação usado para a conversão de endereços de rede em endereços físicos. Permite assim mapear o endereço IP a um endereço MAC.

- **Quais são os endereços IP e MAC dos pacotes ARP e porquê?**
  Quando é feito ping ao tux4 a partir do tux1, o tux1 envia um **pacote de pedido ARP** a perguntar que dispositivo na rede é que tem aquele IP específico, contendo os endereços IP e MAC do dispositivo que fez o pedido (tux1 - 172.12.50.1 e 00:c0:df:25:40:81) e o IP do dispositivo que se pretende estabelecer comunicação (tux4- 172.12.50.254).
  O **pacote de envio ARP** contém o endereço MAC do dispositivo associado ao IP solicitado, neste caso o MAC do tux4 (00:21:5a:c3:78:70) para que a comunicação possa ser estabelecida.

- **Quais são os pacotes gerados pelo comando "ping"?**
  O comando ping gera inicialmente pacotes ARP para determinar os endereços MAC e seguidamente gera pacotes ICMP ou *Internet Control Message Protocol.*

- **Quais são os endereços MAC e IP dos pacotes "ping"?**
  Executando o comando "ping" a partir do tux1 com o objetivo de alcançar o tux4, os endereços MAC e IP serão os seguintes:

  No pacote de pedido: (consultar figura 12)

Endereço MAC de origem:	00:c0:df:25:40:81
Endereço IP de origem:	172.16.50.1
Endereço MAC de destino:	00:21:5a:c3:78:70
Endereço IP de destino:	172.16.50.254

  No pacote de resposta: (consultar figura 13)

Endereço MAC de origem:	00:21:5a:c3:78:70
Endereço IP de origem:	172.16.50.254
Endereço MAC de destino:	00:c0:df:25:40:81
Endereço IP de destino:	172.16.50.1

- **Como determinar se um *Ethernet* frame é ARP, ICMP, ou IP?**
  O *header* de um pacote de Ethernet contém o tipo de trama com que nos encontramos permitindo-nos assim essa distinção.
    - Se o valor do tipo for 0x0806 significa que o trama é do tipo ARP.
    - Se o valor do tipo for 0x0800 trata-se de um trama do tipo IP (ver figura 13 para exemplo), sendo depois possível analisar o IP *header*. Caso o valor deste *header* seja 1, podemos concluir que o tipo de protocolo é ICMP.

- **Como se determina o tamanho de um trama recebido?**
  O tamanho do comprimento do trama é algo que pode ser verificado no *wireshark.* Para um exemplo consultar a imagem 14.

- **Em que consiste a interface *loopback* e qual a sua importância?**
  A interface *loopback* é uma interface virtual da rede que permite ao computador enviar e receber respostas para/de si mesmo. É importante pois permite que esta funcionalidade seja usada para testar se a configuração da rede está correta. (Figura 15)

Experiência 2 - Implementar duas LAN's virtuais no *switch*

Nesta experiência foram criadas duas LAN's virtuais, vlan 60 e 61 (consultar figuras 16 e 17), onde foram associados à vlan 60 o tux1 e tux4 e à vlan 61 o tux2.

- **Como configurar vlany0?**
  Deverão ser introduzidos no GTKTerminal os seguintes comandos:

  → configure terminal

  → vlan y0

  → end

E depois é necessário configurar as portas dos tux 1 e 4:

→ configure terminal

→ interface fastethernet 0/<nº da porta pretendida>

→switchport mode access

→ switchport access vlan y0

→ end

- **Quantos domínios de *broadcast* existem? Como se pode concluir isto através dos *logs*?**

  Após a conclusão desta experiência foram estabelecidos dois domínios de broadcast. Um associado aos tux 1 e 4 e outro associado ao tux2. Isto é verificável através dos *logs* pois ao executar o comando *ping broadcast* no tux1 podemos verificar que é obtida uma resposta do tux4. no entanto nenhuma do tux2. Por outro lado, ao realizar o mesmo comando no tux2, este não obtém nenhuma resposta (Figura 18).

Experiência 3 - Configurar um *Router* em *Linux*

Nesta experiência, o tux4 foi configurado como um router que funciona como uma “ponte” entre ambas as *vlans* criadas na experiência2.

- **Que rotas há nos tuxes e qual o seu significado?**
  No nosso caso, as rotas das vlans:
  - Tux1 tem uma rota para a vlan 60 (172.16.60.0) através da *gateway* 172.16.60.1
  - Tux4 tem uma rota para a vlan 60 (172.16.60.0) através da *gateway* 172.16.60.254 e outra rota para a vlan 61 (172.16.61.0) através da *gateway* 172.16.61.253
  - Tux1 tem uma rota para a vlan 60 (172.16.60.0) através da *gateway* 172.16.61.1

  Nos tuxes foram criadas as seguintes rotas:
  - Tux1 tem uma rota para a vlan 61 (172.16.61.0) pela gateway 172.16.60.254
  - Tux2 tem uma rota para a vlan 61 (172.16.60.0) pela gateway 172.16.61.253

- **Que informações contém uma entrada da tabela de *forwarding*?**
  As seguintes informações podem ser encontradas neste tipo de tabelas:

  Destination: o destino da rota.

  Gateway: o IP do próximo ponto por onde passará a rota.

  Netmask: que permite determinar o ID da rede a partir do endereço IP do destino.

  Flags: dá-nos mais informações sobre a rota.

  *Interface***:** qual a placa de rede responsável pela *gateway* (eth0/eth1).

- **Quais mensagens ARP e endereços MAC são observados e porquê?**

  Ao executar o comando *ping* a partir do tux1 para o tux2 (sendo este agora possível devido ao tux4 funcionar como router), tux1 tenta comunicar com tux2. Para tal, o tux1 tenta comunicar com o tux4, sendo que depois este permite que comunique com o tux2. Inicialmente o tux1 envia um pedido ARP com os seus endereços IP e MAC (no nosso caso 172.16.60.1 e 00:08:54:71:71:2e) e o endereço IP de tux4 (172.16.60.254) sendo que o este último envia uma resposta com o seu MAC (00:21:5a:5a:79:97) para o tux1 o que possibilita a comunicação entre estes dois.

```
57 91.437517039  Netronix_71:71:2e    Broadcast           ARP         60 Who has 172.16.60.254? Tell 172.16.60.1
58 91.437546933  HewlettP_5a:79:97    Netronix_71:71:2e   ARP         42 172.16.60.254 is at 00:21:5a:5a:79:97
```

Seguidamente, é enviado um pacote ARP à procura do endereço IP 172.16.61.253 obtendo como resposta o MAC associado a este endereço.

```
77 99.258706733  HewlettP_5a:74:3e    Netronix_71:73:da    ARP         60 Who has 172.16.61.253? Tell 172.16.61.1
78 99.258716511  Netronix_71:73:da    HewlettP_5a:74:3e    ARP         42 172.16.61.253 is at 00:08:54:71:73:da
```

E por fim, o tux 1 envia novamente um pedido com os seus endereços e o endereço do tux2. O tux2, envia o seu MAC adress e assim já estão mapeados todos os dados necessário para o estabelecimento de comunicação entre o tux1 e o tux2.

```
81 47.131901341  HewlettP_5a:79:97    Netronix_71:71:2e    ARP         60 Who has 172.16.60.1? Tell 172.16.60.254
82 47.131912306  Netronix_71:71:2e    HewlettP_5a:79:97    ARP         42 172.16.60.1 is at 00:08:54:71:71:2e
```

- **Que ICMP packets são observados e porquê?**

  São observados pacotes ICMP de *request* e *reply*, pois depois de serem adicionadas as rotas todos os tux's, estes conseguem ser alcançados por todos os outros. Caso isto nao fosse possível, seriam enviados pacotes ICMP - *Host Unreachable*.

- **Quais são os endereços IP e MAC associados a um ICMP packet e porquê?**

  Os endereços IP e MAC associados com os pacotes ICMP são os endereços IP e MAC dos tux's de origem e destino. Por exemplo, quando se faz *ping* do tux 1 para o tux 4 (.253) os endereços de origem vão ser 172.16.60.1 (IP) e 00:08:54:71:71:2e (MAC) e o de destino 172.16.61.254 (IP) e 00:21:5a:5a:79:97 (MAC).

Experiência 4 - Configurar um router comercial e implementar NAT

Nesta experiência tivemos que ligar à rede do laboratório um router comercial, configurando-o primeiramente sem NAT. Depois, tivemos de o reconfigurar (desta vez com NAT) para que fosse possível o acesso à internet dos computadores da rede.

- **Como configurar uma rota estática num router comercial?**
  Primeiramente é necessário fazermos a configuração do router. Para isto é preciso mudarmos a porta do **GTKTerm** do Switch para o Router e invocar os seguintes comandos de forma a criarmos a VLAN:
  → *configure terminal*
  → *ip route { ip rota de de destino | máscara [ip gateway] }*
  → *exit*
  Por fim, para definir uma rota estática em que todos os computadores da rede conseguissem comunicar com o router, foi necessário adicionar uma *default route* do **tux1** para o **tux4** e do **tux4** e **tux2** para o router (**ver figura 9**).

- **Quais são as rotas seguidas pelos pacotes durante a experiência? Porquê?**
  Caso exista uma rota específica para o destino em questão, eles vão segui-la. Senão, os pacotes vão à rota default à procura do destino pretendido. Tomando o exemplo do **tux1** que tem uma rota default para o **tux4** que por sua vez tem uma rota para o router, assim os pacotes vão do **tux1** para o **tux4** e do **tux4** para o router (**ver figura 10**).

- **Como se configura o NAT no router comercial?**
  Para se configurar o NAT no router foi necessário programar novamente o **GTKTerm**. Apenas foi necessário invocar os comandos dados no enunciado do trabalho (***Configuração do Router Cisco com NAT* - ver figura 11**).

- **O que faz o NAT?**
  NAT significa *Network Address Translation* e tem como objetivo a transformação de endereços IP privados em públicos, de maneira a que estes tenham acesso à internet.
  Visto que o número de endereços IP públicos (IPv4) são um recurso limitado, o NAT ajuda a poupar esse espaço de endereçamento. Isto acontece porque apesar de termos vários endereços privados dentro da nossa rede de computadores, o NAT transforma-os num único endereço público que os deixa aceder ao exterior.
  O NAT pode funcionar também como uma *firewall* uma vez que apenas deixa passar pacotes vindos de fora que tenham sido gerados em resposta a um pedido da rede, impossibilitando a entrada de conexões indesejadas.

Experiência 5 - DNS

Nesta experiência foi preciso configurar o DNS (*Domain Name System*) nos computadores se encontravam ligados ao router (**tux1**, **tux2** e **tux4**). A função do DNS é traduzir endereços IP para os seus respectivos *hostnames*.

- **Como configurar o serviço DNS?**
  Para configurar o DNS apenas é necessário alterar o ficheiro *resolv.conf*. É preciso que este ficheiro contenha a seguinte informação:
  → *search netlab.fe.up.pt*
  → *nameserver 172.16.2.1*
  netlab.fe.up.pt contém uma base de dados de endereços de IP e os seus respectivos *hostnames*. Após esta experiência já nos é possível dar *ping* aos domínios da internet (por exemplo, *google.com*), em vez de o fazer sempre com endereços IP.

- **Que pacotes são trocados pelo DNS e que informações são transportadas?**
  Um pacote é enviado do computador para o servidor que contém o *hostname* desejado. O servidor responde com outro pacote que contém o endereço de IP desse *hostname*.

Experiência 6 - Conexões TCP

Nesta experiência final foi executada a aplicação de download, desenvolvida na parte 1, dentro da rede configurada e foram analisadas as conexões e o protocolo TCP.

- **Quantas conexões TCP são abertas pela aplicação FTP?**
  A aplicação FTP estabeleceu 2 conexões TCP. Uma para a ligação do cliente com o servidor, mandando os comandos necessários para tal e recepção das respostas do servidor. Uma segunda conexão foi aberta para transferência de dados(ficheiro) enviados pelo servidor FTP.

- **Em qual das conexões é transportado o controlo de informação?**
  O controlo de informação é transportado na primeira conexão TCP, aquela responsável pela ligação onde há troca de comandos e resposta do cliente com o servidor.

- **Quais são as fases da conexão TCP?**
  Uma conexão TCP enfrenta 3 fases. A conexão ao servidor, a troca de informação e o encerramento da conexão.

- **Como é que funciona o mecanismo ARQ TCP? Quais são os campos TCP relevantes? Que informação importante pode ser observada nos logs?**
  O mecanismo ***ARQ(Automatic Repeat Request) TCP,*** é semelhante ao **Go-Back-N ARQ** com a diferença que o receptor não deixa de processar os frames quando detecta um erro. Quando o erro acontece, o receptor continua a receber as frames seguintes enviando no ACK o número da frame que falhou até ser recebida corretamente. Quando o emissor verificar os ACK, vai enviar as frames que foram perdidas.
  Para tal mecanismo, é necessário ter em consideração nos campos das mensagens enviadas, os **números ack** e o **sequence number**(número de pacote a ser enviado).
  Ver figura 6 dos Anexos.

- **Como é que o mecanismo de controlo de congestionamento TCP funciona? Quais são os campos relevantes? Como é que o fluxo de dados da conexão evolui ao longo do tempo? Está de acordo com o mecanismo de controlo de congestionamento TCP?**
  De modo a controlar o congestionamento da rede, o TCP utiliza um mecanismo que limita ou aumenta a taxa de envio de dados consoante o congestionamento da rede no momento.
  Verificamos que inicialmente a taxa se mantém muito perto de 0, e passado alguns segundos esta taxa cresce bastante, sendo observável o maior valor da taxa perto dos 6.5s com valor de 170 pacotes/10ms.
  A partir do momento que começou a ser efetuada a transferência do segundo ficheiro o valor da taxa nunca mais teve valores tão alto. É possível ver que aquando do início do segundo download a taxa baixa de valor drasticamente seguida de uma subida a pique, acabando depois por estabilizar até ao final.
  O mecanismo de controlo de congestionamento TCP pode ser verificado na **Figura 7 e 8**, uma vez que quando a rede está mais congestionada a taxa de transferência acaba por ser menor.

- **A taxa de transferência de uma conexão de dados TCP é perturbada pelo aparecimento de uma segunda conexão TCP? Como?**
  Com o aparecimento da segunda conexão TCP a taxa de transferência é afetada uma vez que esta taxa é distribuída de igual forma pelas duas conexões. Com a diminuição da taxa, a transferência irá demorar mais tempo.

Conclusão

No final do segundo trabalho prático, é possível afirmar que o grupo adquiriu conhecimento suficiente para a configuração de uma rede de computadores e o desenvolvimento de uma aplicação de download.

Apesar de inicialmente termos enfrentado diversas dificuldades, principalmente por ser a primeira vez de todos os membros do grupo neste paradigma das redes de computadores, podemos concluir que todo o esforço valeu a pena e que o resultado final deixa-nos bastante orgulhoso.

Em suma, o trabalho foi concluído com sucesso uma vez que fizemos com que a nossa aplicação conseguisse transferir um ficheiro de um servidor através de uma rede configurada pelo grupo.

Observações

A meio do segundo trabalho prático o grupo mudou de bancada, tendo sido desenvolvido a maior parte das experiências e a apresentação na bancada 6 da sala I320 (as experiências iniciais foram feitas na bancada 5)

Por esta mesma razão, é de salientar que certas configurações presentes nos prints e logs do Anexo das experiências 1 e 2 são diferentes das restantes experiências

Referências

- slides das aulas teóricas
- documentos FTP
    - https://tools.ietf.org/html/rfc959
    - https://tools.ietf.org/html/rfc1738
    - https://en.wikipedia.org/wiki/File_Transfer_Protocol
- https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/ipaddr_nat/configuration/15-mt/nat-15-mt-book/iadnat-addr-consv.html

Anexos

Figuras

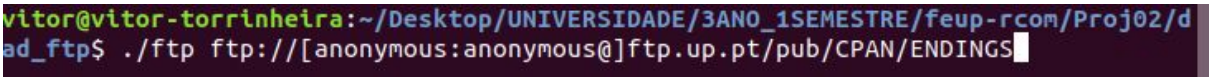

Figura 1: chamada da aplicação de download

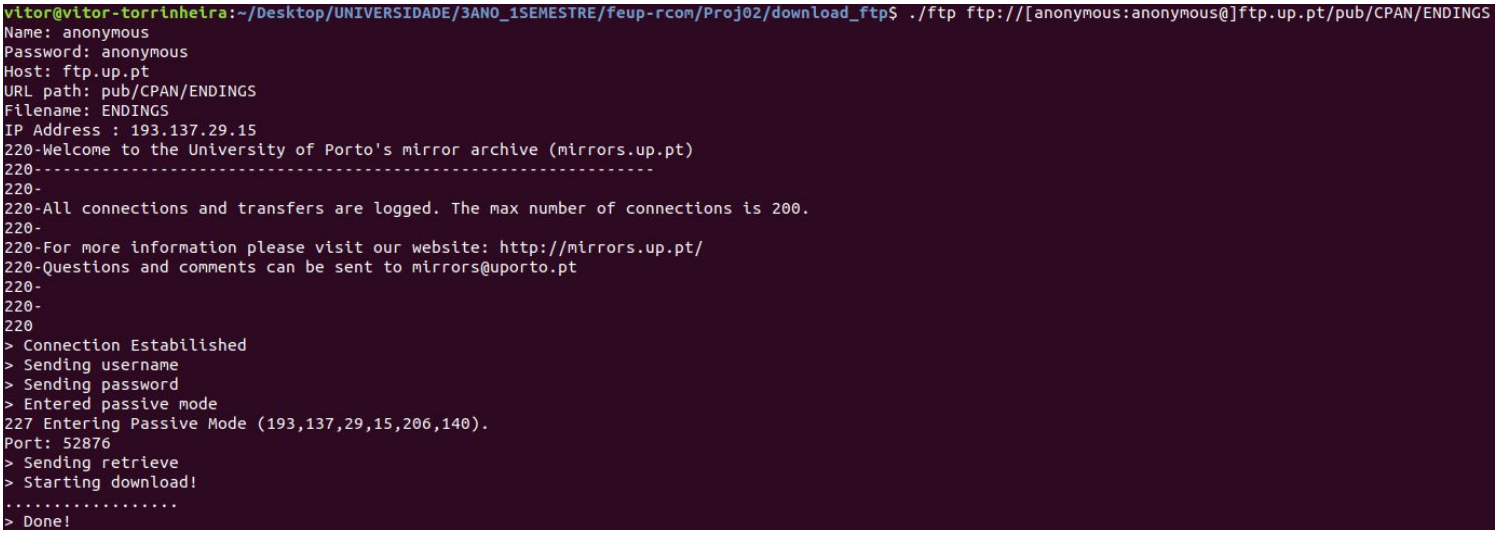

Figura 2: sucesso de download do ficheiro ENDINGS

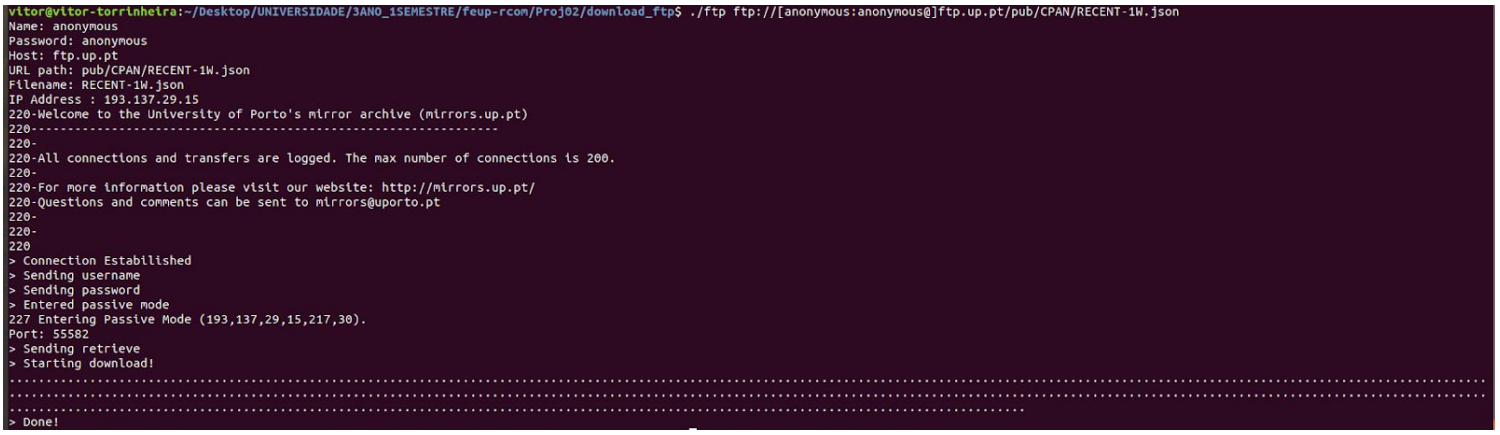

Figura 3: sucesso de download do ficheiro RECENT-1W.json

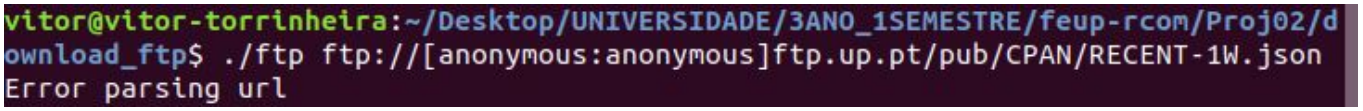

Figura 4: argumento passado com erro(falta @)

Figura 5: argumento passado com erro (falta /)

No.	Time	Source	Destination	Protocol	Length	Info
87206	9.067699710	193.137.29.15	172.16.10.1	FTP-DA…	1434	FTP Data: 1368 bytes (PASV) (retr pub/CPAN/RECENT-Z.json)
87207	9.067816557	193.137.29.15	172.16.10.1	FTP-DA…	1434	FTP Data: 1368 bytes (PASV) (retr pub/CPAN/RECENT-Z.json)
87208	9.067832271	172.16.10.1	193.137.29.15	TCP	66	39189 → 55285 [ACK] Seq=1 Ack=79028425 Win=741376 Len=0 TSval=5699390 TSecr=650961828
87209	9.067932844	193.137.29.15	172.16.10.1	FTP-DA…	1434	FTP Data: 1368 bytes (PASV) (retr pub/CPAN/RECENT-Z.json)
87210	9.068050459	193.137.29.15	172.16.10.1	FTP-DA…	1434	FTP Data: 1368 bytes (PASV) (retr pub/CPAN/RECENT-Z.json)
87211	9.068067710	172.16.10.1	193.137.29.15	TCP	66	39189 → 55285 [ACK] Seq=1 Ack=79031161 Win=741376 Len=0 TSval=5699390 TSecr=650961828
87212	9.068166258	193.137.29.15	172.16.10.1	FTP-DA…	1434	FTP Data: 1368 bytes (PASV) (retr pub/CPAN/RECENT-Z.json)
87213	9.068282336	193.137.29.15	172.16.10.1	FTP-DA…	1434	FTP Data: 1368 bytes (PASV) (retr pub/CPAN/RECENT-Z.json)
87214	9.068298609	172.16.10.1	193.137.29.15	TCP	66	39189 → 55285 [ACK] Seq=1 Ack=79033897 Win=741376 Len=0 TSval=5699390 TSecr=650961828
87215	9.068399601	193.137.29.15	172.16.10.1	FTP-DA…	1434	FTP Data: 1368 bytes (PASV) (retr pub/CPAN/RECENT-Z.json)
87216	9.068516168	193.137.29.15	172.16.10.1	FTP-DA…	1434	FTP Data: 1368 bytes (PASV) (retr pub/CPAN/RECENT-Z.json)
87217	9.068532861	172.16.10.1	193.137.29.15	TCP	66	39189 → 55285 [ACK] Seq=1 Ack=79036633 Win=741376 Len=0 TSval=5699390 TSecr=650961828
87218	9.068633154	193.137.29.15	172.16.10.1	FTP-DA…	1434	FTP Data: 1368 bytes (PASV) (retr pub/CPAN/RECENT-Z.json)
87219	9.068749721	193.137.29.15	172.16.10.1	FTP-DA…	1434	FTP Data: 1368 bytes (PASV) (retr pub/CPAN/RECENT-Z.json)
87220	9.068765366	172.16.10.1	193.137.29.15	TCP	66	39189 → 55285 [ACK] Seq=1 Ack=79039369 Win=741376 Len=0 TSval=5699390 TSecr=650961828
87221	9.068866079	193.137.29.15	172.16.10.1	FTP-DA…	1434	FTP Data: 1368 bytes (PASV) (retr pub/CPAN/RECENT-Z.json)
87222	9.068982925	193.137.29.15	172.16.10.1	FTP-DA…	1434	FTP Data: 1368 bytes (PASV) (retr pub/CPAN/RECENT-Z.json)
87223	9.069000176	172.16.10.1	193.137.29.15	TCP	66	39189 → 55285 [ACK] Seq=1 Ack=79042105 Win=741376 Len=0 TSval=5699390 TSecr=650961828
87224	9.069100191	193.137.29.15	172.16.10.1	FTP-DA…	1434	[TCP Previous segment not captured] FTP Data: 1368 bytes (PASV) (retr pub/CPAN/RECENT-Z.json)
87225	9.069119607	172.16.10.1	193.137.29.15	TCP	78	[TCP Dup ACK 87223#1] 39189 → 55285 [ACK] Seq=1 Ack=79042105 Win=741376 Len=0 TSval=5699390 TSecr=650961828 SLE=79051…
87226	9.069215989	193.137.29.15	172.16.10.1	FTP-DA…	1434	FTP Data: 1368 bytes (PASV) (retr pub/CPAN/RECENT-Z.json)
87227	9.069233660	172.16.10.1	193.137.29.15	TCP	78	[TCP Dup ACK 87223#2] 39189 → 55285 [ACK] Seq=1 Ack=79042105 Win=741376 Len=0 TSval=5699390 TSecr=650961828 SLE=79051…
87228	9.069332696	193.137.29.15	172.16.10.1	FTP-DA…	1434	FTP Data: 1368 bytes (PASV) (retr pub/CPAN/RECENT-Z.json)
87229	9.069349877	172.16.10.1	193.137.29.15	TCP	78	[TCP Dup ACK 87223#3] 39189 → 55285 [ACK] Seq=1 Ack=79042105 Win=741376 Len=0 TSval=5699390 TSecr=650961828 SLE=79051…
87230	9.069450101	193.137.29.15	172.16.10.1	FTP-DA…	1434	FTP Data: 1368 bytes (PASV) (retr pub/CPAN/RECENT-Z.json)
87231	9.069467073	172.16.10.1	193.137.29.15	TCP	78	[TCP Dup ACK 87223#4] 39189 → 55285 [ACK] Seq=1 Ack=79042105 Win=741376 Len=0 TSval=5699390 TSecr=650961828 SLE=79051…
87232	9.069565900	193.137.29.15	172.16.10.1	FTP-DA…	1434	FTP Data: 1368 bytes (PASV) (retr pub/CPAN/RECENT-Z.json)
87233	9.069584618	172.16.10.1	193.137.29.15	TCP	78	[TCP Dup ACK 87223#5] 39189 → 55285 [ACK] Seq=1 Ack=79042105 Win=741376 Len=0 TSval=5699391 TSecr=650961828 SLE=79051…
87234	9.069682467	193.137.29.15	172.16.10.1	FTP-DA…	1434	FTP Data: 1368 bytes (PASV) (retr pub/CPAN/RECENT-Z.json)
87235	9.069699369	172.16.10.1	193.137.29.15	TCP	78	[TCP Dup ACK 87223#6] 39189 → 55285 [ACK] Seq=1 Ack=79042105 Win=741376 Len=0 TSval=5699391 TSecr=650961828 SLE=79051…
87236	9.069800081	193.137.29.15	172.16.10.1	FTP-DA…	1434	FTP Data: 1368 bytes (PASV) (retr pub/CPAN/RECENT-Z.json)
87237	9.069817263	172.16.10.1	193.137.29.15	TCP	78	[TCP Dup ACK 87223#7] 39189 → 55285 [ACK] Seq=1 Ack=79042105 Win=741376 Len=0 TSval=5699391 TSecr=650961828 SLE=79051…
87238	9.069915810	193.137.29.15	172.16.10.1	FTP-DA…	1434	[TCP Previous segment not captured] FTP Data: 1368 bytes (PASV) (retr pub/CPAN/RECENT-Z.json)
87239	9.069932782	172.16.10.1	193.137.29.15	TCP	86	[TCP Dup ACK 87223#8] 39189 → 55285 [ACK] Seq=1 Ack=79042105 Win=741376 Len=0 TSval=5699391 TSecr=650961828 SLE=79062…
87240	9.070034402	193.137.29.15	172.16.10.1	FTP-DA…	1434	[TCP Previous segment not captured] FTP Data: 1368 bytes (PASV) (retr pub/CPAN/RECENT-Z.json)
87241	9.070051374	172.16.10.1	193.137.29.15	TCP	94	[TCP Dup ACK 87223#9] 39189 → 55285 [ACK] Seq=1 Ack=79042105 Win=741376 Len=0 TSval=5699391 TSecr=650961828 SLE=79065…
87242	9.070149991	193.137.29.15	172.16.10.1	FTP-DA…	1434	FTP Data: 1368 bytes (PASV) (retr pub/CPAN/RECENT-Z.json)
87243	9.070166963	172.16.10.1	193.137.29.15	TCP	94	[TCP Dup ACK 87223#10] 39189 → 55285 [ACK] Seq=1 Ack=79042105 Win=741376 Len=0 TSval=5699391 TSecr=650961828 SLE=7906…
87244	9.070266418	193.137.29.15	172.16.10.1	FTP-DA…	1434	FTP Data: 1368 bytes (PASV) (retr pub/CPAN/RECENT-Z.json)
87245	9.070283390	172.16.10.1	193.137.29.15	TCP	94	[TCP Dup ACK 87223#11] 39189 → 55285 [ACK] Seq=1 Ack=79042105 Win=741376 Len=0 TSval=5699391 TSecr=650961828 SLE=7906…
87246	9.070383544	193.137.29.15	172.16.10.1	FTP-DA…	1434	FTP Data: 1368 bytes (PASV) (retr pub/CPAN/RECENT-Z.json)
87247	9.070400935	172.16.10.1	193.137.29.15	TCP	94	[TCP Dup ACK 87223#12] 39189 → 55285 [ACK] Seq=1 Ack=79042105 Win=741376 Len=0 TSval=5699391 TSecr=650961828 SLE=7906…
87248	9.070499552	193.137.29.15	172.16.10.1	FTP-DA…	1434	FTP Data: 1368 bytes (PASV) (retr pub/CPAN/RECENT-Z.json)
87249	9.070516314	172.16.10.1	193.137.29.15	TCP	94	[TCP Dup ACK 87223#13] 39189 → 55285 [ACK] Seq=1 Ack=79042105 Win=741376 Len=0 TSval=5699391 TSecr=650961828 SLE=7906…
87250	9.070615839	193.137.29.15	172.16.10.1	FTP-DA…	1434	FTP Data: 1368 bytes (PASV) (retr pub/CPAN/RECENT-Z.json)
87251	9.070632951	172.16.10.1	193.137.29.15	TCP	94	[TCP Dup ACK 87223#14] 39189 → 55285 [ACK] Seq=1 Ack=79042105 Win=741376 Len=0 TSval=5699391 TSecr=650961828 SLE=7906…
87252	9.070733524	193.137.29.15	172.16.10.1	FTP-DA…	1434	FTP Data: 1368 bytes (PASV) (retr pub/CPAN/RECENT-Z.json)

▸ Frame 1: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0

Figura 6: Pacotes ACK dup

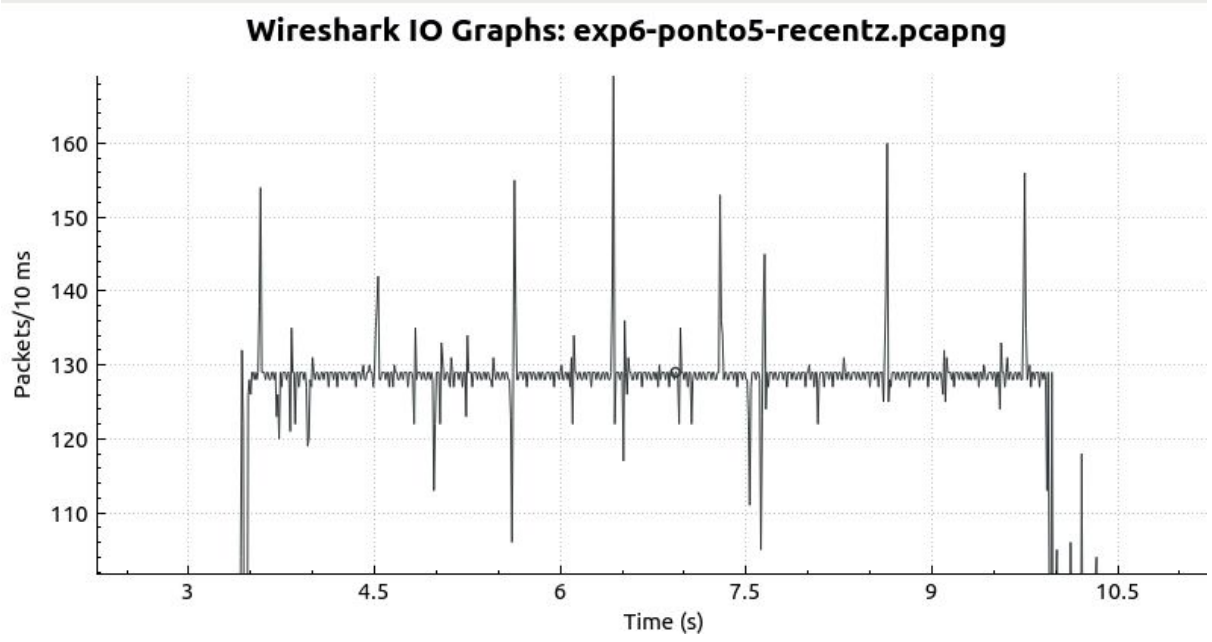

Figura 7: variação fluxo de dados

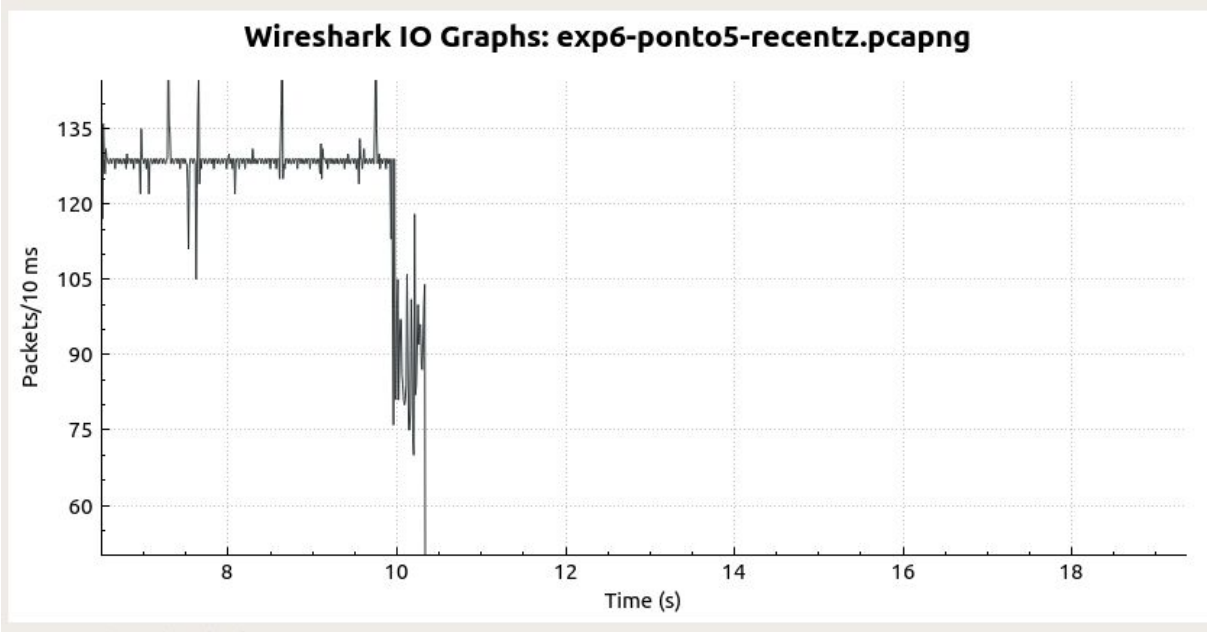

Figura 8: variação fluxo de dados aquando do início da 2º transferência

```
64 bytes from 172.16.2.254: icmp_seq=5 ttl=62 time=0.659 ms
64 bytes from 172.16.2.254: icmp_seq=6 ttl=62 time=0.625 ms
64 bytes from 172.16.2.254: icmp_seq=7 ttl=62 time=0.642 ms
64 bytes from 172.16.2.254: icmp_seq=8 ttl=62 time=0.658 ms
^C
--- 172.16.2.254 ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 6998ms
rtt min/avg/max/mdev = 0.625/0.709/1.078/0.143 ms
gnu11:~# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         172.16.10.254   0.0.0.0         UG    0      0        0 eth0
172.16.10.0     0.0.0.0         255.255.255.0   U     0      0        0 eth0
172.16.11.0     172.16.10.254   255.255.255.0   UG    0      0        0 eth0
gnu11:~#
```

```
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         172.16.11.254   0.0.0.0         UG    0      0        0 eth1
172.16.10.0     0.0.0.0         255.255.255.0   U     0      0        0 eth0
172.16.11.0     0.0.0.0         255.255.255.0   U     0      0        0 eth1
gnu14:~#
```

```
gnu12:~# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         172.16.11.254   0.0.0.0         UG    0      0        0 eth0
172.16.10.0     172.16.11.253   255.255.255.0   UG    0      0        0 eth0
172.16.11.0     0.0.0.0         255.255.255.0   U     0      0        0 eth0
gnu12:~#
```

Figura 9: rotas dos computadores para acesso ao router

```
gnu12:~# traceroute 172.16.10.1
traceroute to 172.16.10.1 (172.16.10.1), 30 hops max, 60 byte packets
 1  172.16.11.254 (172.16.11.254)  1.023 ms  1.307 ms  1.584 ms
 2  172.16.11.253 (172.16.11.253)  2.279 ms  0.385 ms  0.396 ms
 3  172.16.10.1 (172.16.10.1)  0.554 ms  0.550 ms  0.542 ms
gnu12:~#
```

Figura 10: rotas seguidas pelos pacotes durante a experiência

Configuração do Router Cisco com NAT

- Cisco NAT
  http://www.cisco.com/en/US/tech/tk648/tk361/technologies_tech_note09186a0080094e77.shtml

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

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

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

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

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

* In room I320 use interface fastethernet

Figura 11: configuração de NAT num router comercial

```
   45 35.190793817  172.16.50.1      172.16.50.254     ICMP      98 Echo (ping) request  id=0x085a, seq=9/2304, ttl=64 (reply in 46)
   46 35.190906610  172.16.50.254     172.16.50.1       ICMP      98 Echo (ping) reply    id=0x085a, seq=9/2304, ttl=64 (request in 45)

> Frame 45: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface eth0, id 0
> Ethernet II, Src: Kye_25:40:81 (00:c0:df:25:40:81), Dst: HewlettP_c3:78:70 (00:21:5a:c3:78:70)
> Internet Protocol Version 4, Src: 172.16.50.1, Dst: 172.16.50.254
> Internet Control Message Protocol
```

Figura 12: log de pacote de pedido do ping do tux1 para t

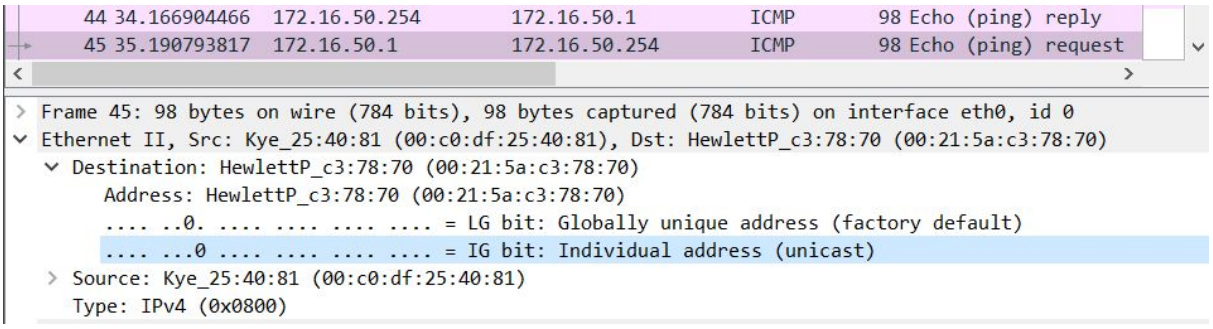

Figura 13: log de pacote de resposta do ping do tux1 para tux4
Pode-se verificar também o tipo de trama (0x0800 -> IPv4)

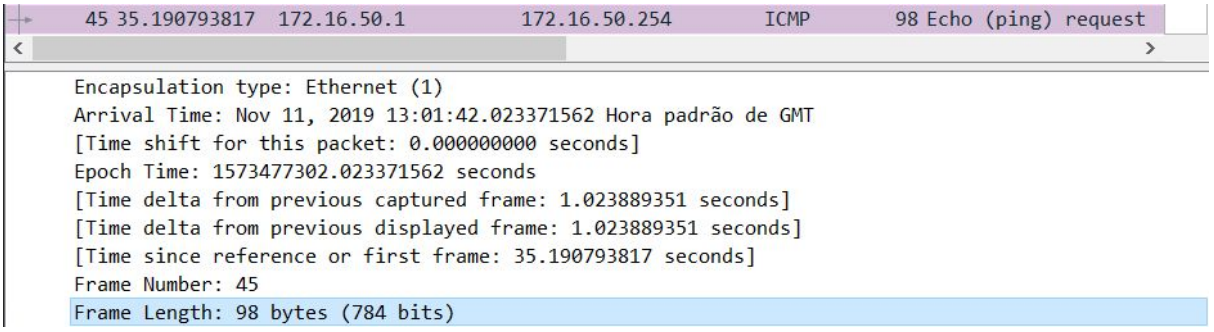

Figura 14: tamanho de um pacote de resposta do ping do tux1 para tux4.

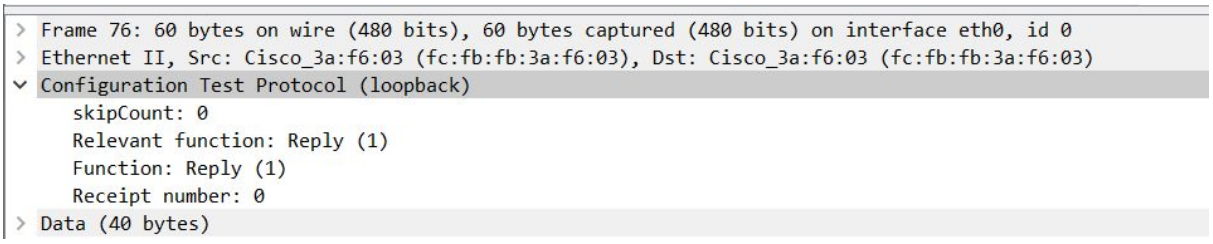

Figura 15: loopback

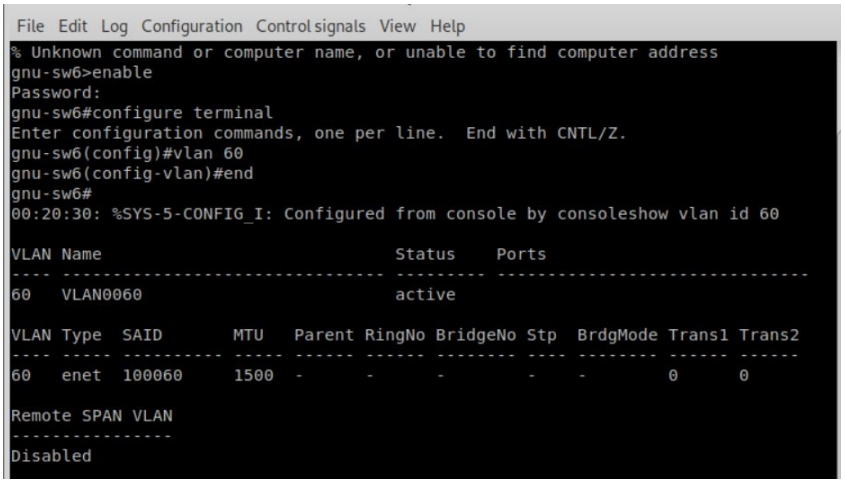

Figura 16: #show vlan id 60

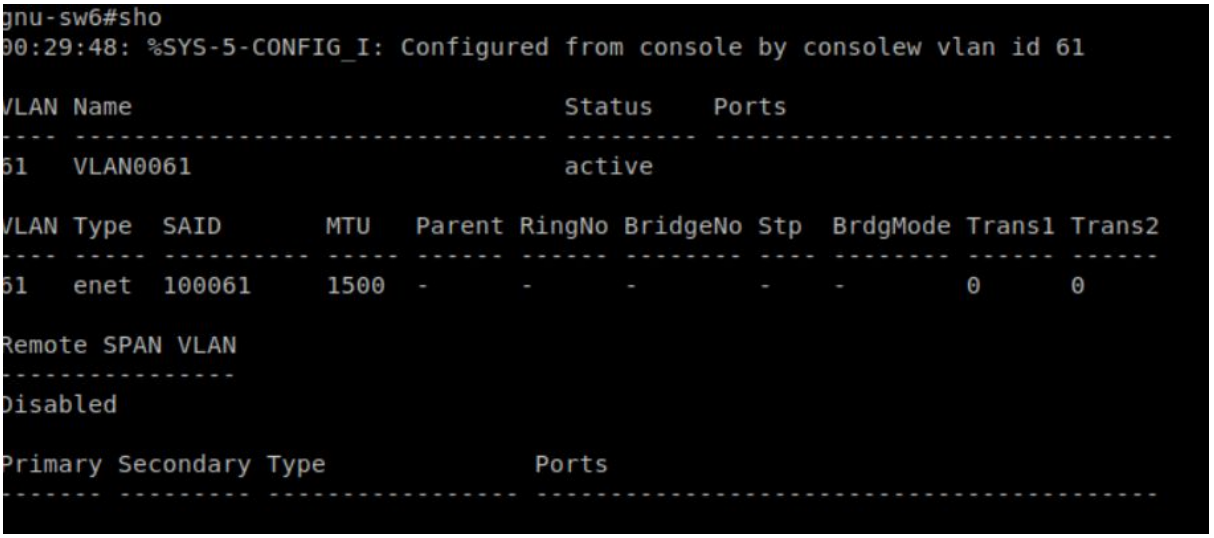

Figura 17: #show vlan id 61

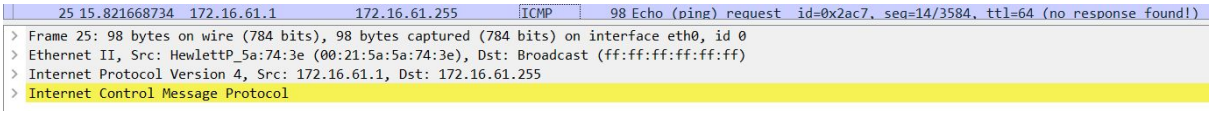

Figura 18: Exp 3, broadcast from tux2 (no response found)

Aplicação FTP

- **FTP_download.c**

```c
#include "auxiliar_func.h"
#include "macros.h"


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



    if(argc != 2){
        printf("Arguments are not correct\n");
        return -2;
    }

    //parse arguments


    int url_size = strlen(argv[1]);
    int new_size = url_size - 7;


    if(new_size <= 8){
        printf("Error: Check url... Something went wrong\n");
        return -1;
    }

    char argv1[new_size];

    int j = 7;
    for(size_t i = 0 ; i <= new_size ; i++){
        argv1[i] = (argv[1])[j];
        j++;
    }

    //state machine

    char name[SIZE];
    memset(name, 0, SIZE);

    char password[SIZE];
    memset(password, 0, SIZE);

    char host[SIZE];
    memset(host, 0, SIZE);

    char path_file[SIZE];
    memset(path_file, 0, SIZE);

    char file_name[SIZE];
    memset(file_name, 0, SIZE);
```

```
50
51      int estado = 0;
52      int i = 0;
53      int indice = 0;
54
55      bool erro = false;
56      bool found_1 = false;
57      bool found_2 = false;
58      bool found_3 = false;
59
60
61
62      while( i < strlen(argv1) && !erro){
63          switch (estado)
64          {
65          case 0:
66              if(argv1[i] == ':'){
67                  estado = 1;
68                  indice = 0;
69                  found_1 = true;
70              }
71              else{
72                  name[indice] = argv1[i];
73                  indice++;
74              }
75              break;
76
77          case 1:
78              if(argv1[i] == '@'){
79                  estado = 2;
80                  indice = 0;
81                  found_2 = true;
82              }
83              else{
84                  password[indice] = argv1[i];
85                  indice++;
86              }
87              break;
```

```
88          case 2:
89              if(argv1[i] == ']'){
90                  estado = 3;
91                  indice = 0;
92              }
93              else{
94                  erro = true;
95              }
96              break;
97          case 3:
98              if(argv1[i] == '/'){
99                  estado = 4;
100                 indice = 0;
101                 found_3 = true;
102             }
103             else{
104                 host[indice] = argv1[i];
105                 indice++;
106             }
107             break;
108
109         case 4:
110             path_file[indice] = argv1[i];
111             indice++;
112             break;
113         }
114
115         i++;
116     }
117
118     if(erro || !found_1 || !found_2 || !found_3){
119         printf("Error parsing url\n");
120         return -1;
121     }
122
123     //com o urlpath fazer parser ate chegar ao filename
124     int n_barras = 0;
125     for(int i = 0; i < strlen(path_file); i++){
126         if(path_file[i] == '/'){
127             n_barras++;
128         }
129     }
```

```
131     int barras_atuais = 0;
132     int indice_file = 0;
133     for(int i = 0; i < strlen(path_file); i++){
134         if(barras_atuais == n_barras){
135             file_name[indice_file] = path_file[i];
136             indice_file++;
137         }
138 
139         if(path_file[i] == '/'){
140             barras_atuais++;
141         }
142 
143     }
144 
145 
146     printf("Name: %s\n", name);
147     printf("Password: %s\n", password);
148     printf("Host: %s\n", host);
149     printf("URL path: %s\n",path_file);
150     printf("Filename: %s\n",file_name);
151 
152 
153     //get host ip
154 
155 
156     //check given code to understand
157     struct hostent *h;
158 
159 
160     if((h=gethostbyname(host)) == NULL){
161         herror("gethostbyname");
162         exit(1);
163     }
164 
165     char* ip = inet_ntoa(*((struct in_addr *) h->h_addr));
166     printf("IP Address : %s\n", ip);
167 
168 
```

```
171
172     //============= FTP client process ===========//
173
174
175     //check clientTCP.c from the given code (lot of code taken from there)
176     int sockfd; //um socket para estabelecer ligação
177     int sockfd_file_transfer; //um socket para download do ficherio
178     struct  sockaddr_in server_addr;
179     struct  sockaddr_in server_addr_file_transfer;
180
181
182     /*server address handling*/
183     bzero((char*)&server_addr,sizeof(server_addr));
184     server_addr.sin_family = AF_INET;
185     server_addr.sin_addr.s_addr = inet_addr(ip);    /*32 bit Internet address network byte ordered*/
186     server_addr.sin_port = htons(SERVER_PORT);      /*server TCP port must be network byte ordered */
187
188     /*open an TCP socket*/
189     if ((sockfd = socket(AF_INET,SOCK_STREAM,0)) < 0) {
190             perror("socket()");
191             exit(0);
192         }
193     /*connect to the server*/
194         if(connect(sockfd, (struct sockaddr *)&server_addr, sizeof(server_addr)) < 0){
195             perror("connect()");
196             exit(0);
197     }
198
199
200     char responseCode[3];
201     readResponse(sockfd, responseCode);
202     if (responseCode[0] == '2')
203     {
204         printf("> Connection Estabilished\n");
205     }
206
207
208
209     printf("> Sending username \n");
210
211     char response[SIZE];
212     dprintf(sockfd, "user %s\r\n", name);
213     read(sockfd, response, SIZE);
```

```c
    if(strncmp(response, "331", 3) == 0){//user certo,  mandar pass
        printf("> Sending password \n");

        char response2[SIZE];
        dprintf(sockfd, "pass %s\r\n", password);
        read(sockfd, response2, SIZE);


        if(strncmp(response2, "230", 3) == 0){// se pass certa dowload ficheiro(entrar em modo passivo e abrir 2º socket)
            //entrar em modo passivo e fazer download do ficheiro
            dprintf(sockfd, "pasv\r\n");//printf para um filedescriptor
            printf("> Entered passive mode\n");

            char response3[SIZE];
            read(sockfd, response3, SIZE);
            printf("%s", response3);

            int port_to_download = parseResponse(response3);
            printf("Port: %d\n", port_to_download);

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

            /*open an TCP socket*/
            if ((sockfd_file_transfer = socket(AF_INET,SOCK_STREAM,0)) < 0) {
                    perror("socket()");
                    exit(0);
                }
            /*connect to the server*/
                if(connect(sockfd_file_transfer, (struct sockaddr *)&server_addr_file_transfer, sizeof(server_addr_file_transfer)) < 0){
                    perror("connect()");
                    exit(0);
            }

            printf("> Sending retrieve\n");
            dprintf(sockfd, "retr %s\r\n", path_file);//printf para um filedescriptor(antigo)
            create_file(sockfd_file_transfer, file_name);

        }
```

```c
        else if(strncmp(response2, "430", 3) == 0){
            printf("> Invalid credentials \n");
            return -1;
        }
        else{
            printf("> Error occured2 \n");
            return -1;
        }
    }
    else{
        printf("> Error occured1 \n");

    }

    close(sockfd_file_transfer);
    close(sockfd);



    return 0;

}
```

- **auxiliar_func.c**

```c
#include "auxiliar_func.h"

void read_answer(int socket, char *host_answer){

    bool stop = false;
    char response;
    int estado = 0;
    int i = 0;

    while(!stop){
        //ler do socket um char de cada vez
        read(socket, &response, 1);
        //imprime na consola a resposta do user
        printf("%c", response);

        switch(estado){
            case 0:
                if(i == 3){
                    estado = 1;
                }
                else{
                    host_answer[i] = response;
                    i++;
                }

                break;
            case 1:
                if(response == '\n'){
                    stop = true;
                }
                break;
        }
    }
}
```

```
35
36  int parseResponse(char* response){
37      char mostSig[5];
38      memset(mostSig, 0, 5);
39      char lessSig[4];
40      memset(lessSig, 0, 5);
41
42      int i = 0;
43      int ms = 0;
44      int ls = 0;
45      int state = 0;
46      int number_comma = 0;
47
48      while(i < strlen(response)){
49          if(state == 0){
50              if(response[i] == '('){
51                  state = 1;
52                  i++;
53              }
54              else{
55                  i++;
56              }
57          }
58          else if(state == 1){
59              if(number_comma < 4){
60                  if(response[i] == ','){
61                      number_comma++;
62                  }
63                  i++;
64              }
65              else if(number_comma == 4){
66                  if(response[i] == ','){
67                      number_comma++;
68                  }
69                  else{
70                      mostSig[ms] = response[i];
71                      ms++;
72                  }
73                  i++;
74              }
75              else if(number_comma == 5){
76                  if(response[i] == ')'){
77                      state = 2;;
78                  }
79                  else{
80                      lessSig[ls] = response[i];
81                      ls++;
82                  }
```

```
83                  i++;
84              }
85
86          }
87          else{
88              i++;
89          }
90
91      }
92
93
94      int mostSignificant = atoi(mostSig);
95      int lessSignificant = atoi(lessSig);
96      return (mostSignificant * 256 + lessSignificant);
97  }
```

```
99  void create_file(int sockfd_file_transfer, char* path_file){
100     FILE *file = fopen(path_file, "wb+");
101
102
103     char buffer[1000];
104     int bytes;
105     int counter = 0;
106
107     printf("> Starting download!\n");
108
109     while ((bytes = read(sockfd_file_transfer, buffer, 1000))>0) {
110             printf("...");
111             bytes = fwrite(buffer, bytes, 1, file);
112     }
113
114     fclose(file);
115     printf("\n");
116     printf("> Done!\n");
117
118 }
119
```

```c
void readResponse(int sockfd, char *response)
{
    int indece = 0;
    char c;
    int estado = 0;


    while (estado != 3)
    {
        read(sockfd, &c, 1);
        printf("%c", c);
        switch (estado){
        case 0:
            if (c == ' '){
                if (indece != 3){
                    printf(" > Error\n");
                    return;
                }
                indece = 0;
                estado = 1;
            }
            else{
                if (c == '-'){
                    estado = 2;
                    indece=0;
                }
                else{
                    if (isdigit(c)){
                        response[indece] = c;
                        indece++;
                    }
                }
            }
            break;
        case 1:
            if (c == '\n'){
                estado = 3;
            }
            break;
```

```
160             case 2:
161                 if (c == response[indece]){
162                     indece++;
163                 }
164                 else{
165                     if (indece== 3 && c == ' '){
166                         estado = 1;
167                     }
168                     else {
169                       if(indece==3 && c=='-'){
170                         indece=0;
171
172                     }
173                     }
174
175                 }
176                 break;
177             }
178         }
179     }
```

- **auxiliar_func.h**

```
1   #include <stdio.h>
2   #include <sys/types.h>
3   #include <sys/socket.h>
4   #include <netinet/in.h>
5   #include <arpa/inet.h>
6   #include <stdlib.h>
7   #include <unistd.h>
8   #include <signal.h>
9   #include <netdb.h>
10  #include <string.h>
11  #include <stdbool.h>
12  #include <ctype.h>
13
14
15
16  #include "macros.h"
17
18  void read_answer(int socket, char *host_answer);
19  int parseResponse(char* response);
20  void create_file(int sockfd_file_transfer, char* path_file);
21  void readResponse(int sockfd, char *response);
```

- **macros.h**

```
1   #include <stdlib.h>
2
3   #define SIZE 256
4   #define SERVER_PORT  21
```