

**Faculdade de Engenharia da Universidade do Porto**



## **Trabalho 2**

**‘Configuração de uma rede e  
desenvolvimento de uma aplicação de  
download ‘**

**Projeto RCOM 22/23**

(L.EC025: Redes de Computadores)

**Licenciatura em Engenharia Informática e Computação**

Professor Manuel Ricardo

Professor Rui Prior

Professor Helder Fontes

**Estudantes & Autores (Turma 12 Grupo 3):**

Pedro Balazeiro [up202005097@fe.up.pt](mailto:up202005097@fe.up.pt)

Rúben Viana [up202005108@fe.up.pt](mailto:up202005108@fe.up.pt)

## Resumo

Este relatório foi realizado no âmbito da unidade curricular de Redes de Computadores, e trata-se de uma complementação ao segundo trabalho laboratorial, cuja essência é exatamente o nome da cadeira. O trabalho consiste na configuração e estudo da mesma e no desenvolvimento de uma aplicação de download de um ficheiro de acordo com o protocolo FTP (File Transfer Protocol). Em suma, o projeto foi finalizado com sucesso, pois todos os objetivos estabelecidos foram alcançados e foi concluída uma aplicação capaz de exercer a sua função, assim como foi feita a correta configuração da rede.

## Introdução

No âmbito da cadeira de Redes de Computadores desenvolveu-se o segundo projeto, durante as aulas laboratoriais, consistente no estudo de uma rede de computadores, da sua configuração e posterior ligação a uma aplicação de *download* desenvolvida pelo grupo. Na primeira aula focou-se na primeira parte do guião, ou seja, desenvolvimento de uma aplicação e numa maior interiorização sobre os protocolos de aplicação IETF (Internet Engineering Task Force). O protocolo focado neste trabalho foi o FTP (File Transfer Protocol) com auxílio de um servidor da faculdade, como por exemplo *ftp.fe.up.pt*. Para alcançar os objetivos pretendidos, além de seguir as recomendações e instruções fornecidas pelo guião, foram necessárias pesquisas extras acerca dos assuntos em questão. Quanto à segunda parte (configuração de rede), o principal objetivo é permitir a execução da aplicação referida a partir de duas bridges dentro de um switch. No que toca aos objetivos da aplicação de download era importante a compreensão dos tópicos referidos no guião. Alcançando-os, seria então realizado o desenvolvimento da aplicação implementando um cliente FTP e uma ligação TCP a partir de sockets. Após isso seria então possível concluir a importância do DNS na conversão de um URL para um IP, permitindo a sua localização num host com domínio determinado.

Quanto ao relatório, o seu objetivo é expor e explicar toda a componente teórica presente neste segundo trabalho, tendo a seguinte estrutura:

- **Resumo**, onde é descrito brevemente o relatório;
- **Introdução**, onde são descritos os objetivos do trabalho;
- **Parte 1: Aplicação de Download**, onde é descrita a sua arquitetura, respetivos resultados e análise além de documentos utilizados no auxílio da sua implementação;
- **Parte 2: Configuração de Rede e Análise**, onde é descrita a sua arquitetura, objetivos de cada experiência, comandos de configuração e análise dos logs gravados durante a sua realização;
- **Conclusões**, síntese da informação apresentada nas secções anteriores e reflexão sobre os objetivos de aprendizagem alcançados;
- **Anexos**, imagens e informação complementar.

Antes de prosseguir é de referir que o grupo desenvolveu este projeto em ambiente LINUX, com a linguagem de programação C.

## Parte 1: Aplicação de Download

Como já foi referido, a primeira parte deste trabalho foi desenvolver uma aplicação download. Esta aceita um link como argumento:  
*ftp://[<user>:<password>@]<host>/<url-path>*. A aplicação consegue fazer o download de qualquer tipo de ficheiros de um servidor FTP. Para tal efeito foi estudado o RFC959 que fala sobre o FTP e o RFC1738 que fala sobre o tratamento de informação provenientes de URLs.

Abaixo está descrito resumidamente a arquitetura da aplicação, a apresentação de resultados e a sua análise.

### Arquitetura:

Inicialmente, começa-se por fazer o processamento do *URL*. Para o fazer, é reservado espaço para as variáveis *user*, *password*, *host*, *path* e *filename* e depois é chamada a função ***parseInput*** para obter estas mesmas variáveis a partir do *URL*. O nome do ficheiro é obtido a partir do *path* e o endereço de *IP* a partir da função ***getIp***. A porta usada é sempre a 21 como referido no guião. Depois é aberta a conexão utilizando a função ***openConnection*** que vai ser responsável por abrir o *socket TCP* e conectar-se ao servidor. De seguida são enviados os comandos para ser feito o login definindo o *user* e a *pass* (para tal, são utilizadas as funções ***sendCMD*** e ***recvMSG***, que enviam e recebem comandos, respetivamente). Após este passo é feita a entrada em modo passivo, que vai retornar a porta necessária para a abertura de um outro socket (através da função ***recvMSGpasv***) que vai servir para troca de dados. Seguidamente é usada ***sendCMD*** no intuito de pedir o ficheiro e consequentemente é feito o seu download com a ajuda da função ***saveToFile***. No final, são fechadas as duas conexões tanto a de transferência de comandos como a de dados com a ajuda da função ***closeConnection***.

### Resultados:

A nossa aplicação foi testada nas diversas condições propostas: modo anónimo, modo não anónimo, vários tipos e tamanhos de ficheiros, entre outros passando em todos na presença do professor durante a apresentação, pelo que não vimos necessidade de os colocar aqui. Além disso, em caso de erro ou caso o ficheiro não exista o programa é terminado (presença de controlo de erros). Também é de notar que ao longo do seu funcionamento são impressas mensagens para um maior controlo por parte do utilizador (serve para se guiar melhor e entender certos pormenores que estão a acontecer).

## Experiência 1 - Configurar um IP de rede

Esta experiência tem como objetivo a compreensão da configuração de *IPs* em duas máquinas diferentes, de modo a permitir a comunicação entre ambas.

### 1 - O que são os pacotes ARP e para o que são usados?

*ARP* (*Address Resolution Protocol*) é um protocolo de comunicação, usado para a conversão de endereços entre as camadas de rede e enlace (por exemplo endereço *IPv4* em endereço *MAC*).

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

Nós sabemos que quando fazemos *ping* do *tux53* para o *tux54*, este último envia um pacote a perguntar quem é o *tux* com aquele *IP*, ou seja, a perguntar que endereço *MAC* tem o *tux* que lhe está a tentar mandar algo. Esta pergunta vem na forma de um pacote *ARP* com o endereço *IP* e endereço *MAC* do *tux54* (*172.16.50.254* e *00:21:5a:c3:78:70*, respetivamente) e com o endereço *IP* do *tux target*, ou seja, o *tux* do qual se quer saber o *MAC* (*172.16.50.1*), que é o *tux53* neste caso. Como não se sabe o *MAC* do *tux target* este está registado como *00:00:00:00:00:00* (deverão ser consultados os *logs* para mais informação). Depois, o *tux53* responde a dizer que é ele que tem aquele *IP* (ou seja, a dizer que é o *tux target*) enviando o seu endereço *MAC*. No pacote de resposta presente, o endereço *IP* e *MAC* de origem são o do *tux53* (*172.16.50.1* e *00:21:5a:61:2d:72*, respetivamente) e o endereço *IP* e *MAC* de destino são o do *tux54* (*172.16.50.254* e *00:21:5a:c3:78:70*, respetivamente).

### 3 - Quais os pacotes gerados pelo comando ping?

O comando *ping* começa por gerar pacotes *ARP* para obter os endereços *MAC* e depois gera pacotes *ICMP* (*Internet Control Message Protocol*).

### 4 - Quais são os endereços MAC e IP dos pacotes ping?

Nós sabemos que quando fazemos *ping* do *tux53* para o *tux54* os endereços de origem e de destino *IP* e *MAC* dos pacotes vão ser os destes *tuxs*. Exemplo do que nos apareceu:

Pacote de pedido:

Endereço *MAC* de origem do pacote: *00:21:5a:61:2d:72* (*tux53*);  
Endereço *MAC* de destino do pacote: *00:21:5a:c3:78:70* (*tux54*);  
Endereço *IP* de origem do pacote: *172.16.50.1* (*tux53*);  
Endereço *IP* de destino do pacote: *172.16.50.254* (*tux54*).

Pacote de resposta:

Endereço *MAC* de origem do pacote: *00:21:5a:c3:78:70* (*tux54*);  
Endereço *MAC* de destino do pacote: *00:21:5a:61:2d:72* (*tux53*);  
Endereço *IP* da origem do pacote: *172.16.50.254* (*tux54*);  
Endereço *IP* do destino do pacote: *172.16.50.1* (*tux53*).

### 5 - Como determinar se o frame Ethernet recetor é ARP, IP, ICMP?

Conseguimos determinar o tipo de frame Ethernet recetor ao inspecionar o *Ethernet header* de um pacote. Caso o *type* tenha o valor *0x0800*, significa que o tipo de *frame* é *IP*, e se *type* tiver o valor *0x0806*, significa que o tipo de *frame* é *ARP*. Depois, ao analisar o *IP header* se ele tiver o valor 1 isso significa que o tipo de protocolo é *ICMP*. Em *frames* do tipo *Ethernet*, os bits 21 e 22 do frame identificam o protocolo para o qual deve ser enviado o *payload*.

### 6 - Como determinar o comprimento de um frame recetor?

O comprimento de um *frame* recetor pode ser determinado usando o *wireshark* para o inspecionar.

### 7 - O que é a interface loopback e porque é que é importante?

Uma interface de loopback é uma interface de rede virtual que permite que um cliente e um servidor no mesmo host se comuniquem entre si usando a pilha de protocolos TCP/IP, com o objetivo de realizar testes de diagnóstico ou aceder a servidores na própria máquina. Logo, uma interface loopback permite a existência de um endereço IP no router, que está sempre ativo, em vez de ser dependente de uma interface física.

## Experiência 2 – Implementar duas bridges no switch

Nesta experiência foram criadas duas *bridges* no *switch*: a primeira constituída pelas máquinas *tux53* e *tux54*, e a segunda pela máquina *tux52*. Com esta configuração, a máquina *tux52* deixaria de ter acesso às máquinas *tux53* e *tux54*, uma vez que se encontrariam em sub-redes diferentes.

### 1 - Como configurar bridgeY0?

Primeiramente, cria-se a *bridge50* (no nosso caso) no *switch*. Depois removemos as portas onde os *tux53* e *tux54* estão conectados com a *bridge default*. E, por fim, adicionamos as portas correspondentes à *bridge50*.

Assim, para criar a *bridge50* invocam-se os seguintes comandos no GTKTerm do *tux* escolhido:

```
/interface bridge add name=bridge50;  
/interface bridge port remove [find interface=ether16];  
/interface bridge port remove [find interface=ether8];
```

Depois deverá adicionar-se as portas dos *tux53* e *tux54*:

```
/interface bridge port add bridge=bridge50 interface=ether8;  
/interface bridge port add bridge=bridge50 interface=ether16.
```

Cabos:

```
tux53 E0 ---- switch eth8;  
tux54 E0 ---- switch eth16;  
tux54 S0 ---- switch/router serial port.
```

## 2 - Quantos domínios de transmissão existem? O que se pode concluir a partir dos registos?

Existem dois domínios de transmissão, visto que o *tux53* recebe resposta do *tux54* quando faz *ping broadcast*, mas não do *tux52*. O *tux54* recebe resposta do *tux53* quando faz *ping broadcast*, mas não do *tux52*. E, por fim, o *tux52* não recebe resposta de ninguém quando faz *ping broadcast*. Assim, existem dois domínios de *broadcast*: o que contém o *tux53* e *tux54* e o que contém o *tux52*. Ou seja, duas sub-redes diferentes basicamente.

## Experiência 3 – Configurar um router em Linux

O objetivo desta experiência era configurar a máquina *tux54* como *router* entre as duas sub-redes criadas na experiência anterior.

### 1 - Que rotas há nos tuxs? Qual o seu significado?

Os tuxs têm as rotas para as bridges associadas por natureza e as que foram criadas durante a experiência devido à necessidade.

As rotas para as bridges associadas:

O *Tux52* tem uma rota para a *bridge51* (172.16.51.0) pela *gateway* 172.16.51.1.

O *Tux53* tem uma rota para a *bridge50* (172.16.50.0) pela *gateway* 172.16.50.1.

O *Tux54* tem uma rota para a *bridge50* (172.16.50.0) pela *gateway* 172.16.50.254 e uma rota para a *bridge51* (172.16.51.0) pela *gateway* 172.16.51.253.

As rotas que foram criadas durante a experiência:

O *Tux52* tem uma rota para a *bridge50* (172.16.50.0) pela *gateway* 172.16.51.253.

```
route add -net 172.16.50.0/24 gw 172.16.51.253
```

O *Tux53* tem uma rota para a *bridge51* (172.16.51.0) pela *gateway* 172.16.50.254.

```
route add -net 172.16.51.0/24 gw 172.16.50.254
```

O destino das rotas é até onde o *tux* que está na origem da rota consegue chegar.

### 2 - Que informação é que uma entrada da tabela de forwarding contém?

A tabela de *forwarding* tem as seguintes informações:

*IP Address*: endereço de *IP* a ser mapeado;

*MAC Address*: endereço *MAC* a ser mapeado para;

*Interface*: Nome da interface à qual o endereço *IP* está atribuído.

A tabela de roteamento tem as seguintes informações:

*Destination*: indica o destino da rota;

*Gateway*: indica o IP do próximo ponto por onde passará a rota;

*Netmask*: usado para determinar o ID da rede a partir do endereço IP do destino;

*Flags*: fornece-nos informações sobre a rota;  
*Metric*: indica o custo de cada rota;  
*Ref*: indica o número de referências para esta rota;  
*Use*: contador de pesquisas pela rota, dependendo do uso de -F ou -C isto vai ser o número de falhas da cache (-F) ou o número de sucessos (-C);  
*Interface*: indica qual a placa de rede responsável pela gateway (eth0/eth1).

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

Nós sabemos que quando um *tux* dá ping a outro e o *tux* que recebeu o *ping* não conhece o endereço *MAC* do que enviou o *ping*, pergunta qual o endereço *MAC* do *tux* com aquele *IP*. Como se sabe, faz isso enviando uma mensagem *ARP*. Essa mensagem vai ter o *MAC* do *tux* de origem associado e 00:00:00:00:00:00 (mensagem enviada em modo de *broadcast*) pois ainda não sabe qual o *tux* de destino. Depois, o *tux* de destino responde uma mensagem *ARP* a dizer o seu endereço *MAC*. Esta mensagem vai ter associado tanto o endereço *MAC* do *tux* de destino como o de origem.

### 4 - Que pacotes ICMP são observados e porquê?

São observados pacotes *ICMP* de *request* e *reply*, pois depois de serem adicionadas as rotas todos os *tuxs* têm acesso e visibilidade uns dos outros. Caso contrário seriam enviados os pacotes *ICMP* de *Host Unreachable*.

### 5 - Quais são os endereços IP e MAC associados a um pacote ICMP e porquê?

Os endereços *IP* e *MAC* associados a um pacote *ICMP* são os endereços *IP* e *MAC* dos *tuxs* de origem e de destino. Por exemplo, quando se faz *ping* do *tux53* para o *tux54* (.253) os endereços de origem vão ser 172.16.50.1 (*IP*) e 00:21:5a:61:2d:72 (*MAC*) e os de destino 172.16.51.253 (*IP*) e 00:21:5a:c3:78:70 (*MAC*).

## Experiência 4 – Configurar um router comercial e implementar o NAT

Nesta experiência foi configurado um router comercial com *NAT* ativo por *default*. *NAT* (Network Address Translation) tem como objetivo possibilitar a comunicação entre os computadores da rede criada com redes externas.

### 1 - Como se configura uma rota estática num router comercial?

Invocam-se os seguintes comandos no GTKTerm do *tux* escolhido:

*Rotas estáticas* (exemplo usado):

```
/ip route add dst-address=172.16.50.0/24 gateway=172.16.51.253
```

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

As rotas seguidas pelos pacotes durante a experiência podem variar. Caso a rota exista, os pacotes usam essa mesma rota. Caso contrário, os pacotes vão ao *router* (rota *default*), o *router* informa que o *tux54* existe, e deverá ser enviado pelo mesmo. Se o *router* não tivesse uma rota *default* os pacotes seriam simplesmente descartados

### 3 - Como se configura o NAT num router comercial?

Num router comercial o NAT está ativo por *default*. Comandos para fazer certas configurações seriam:

Disable default nat: – */ip firewall nat disable 0*

Add NAT rules: – */ip firewall nat add chain=srcnat action=masquerade out-interface=ether1*

### 4 - O que faz o NAT?

O NAT (*Network Address Translation*) tem como objetivo a conservação de endereços *IP*. Essencialmente, é uma técnica que permite reescrever os *IPs* de origem de uma rede interna, para que possam aceder a uma rede externa. Este procedimento gera um número de 16 bits, utilizando esse valor numa hash table, e escrevendo-o no campo da porta de origem. Na resposta, o processo é revertido, e o router sabe para qual computador da rede interna deve enviar a resposta. Resumidamente, permite que os computadores de uma rede interna, como a que foi criada por nós, tenham acesso ao exterior, sendo que, um único endereço *IP* é exigido para representar um grupo de computadores fora da sua própria rede.

## Experiência 5 - DNS

O objetivo desta experiência era conseguir aceder a redes externas, conseguindo desta forma aceder à Internet, através da rede interna criada. Para isto, foi necessário configurar o DNS (*Domain Name System*). Usou-se o servidor de *DNS* *services.netlab.fe.up.pt*.

### 1 - Como configurar o serviço DNS num host?

Para configurar o serviço *DNS* num host, é necessário mudar o ficheiro *resolv.conf* que se localiza em *vi/etc* no *host tux* para conter a seguinte informação:

*search netlab.fe.up.pt*

*nameserver 172.16.1.1*

Sabe-se que *netlab.fe.up.pt* é o nome do servidor *DNS* e *172.16.1.1* o seu endereço de *IP* e que no fim desta experiência é possível aceder à internet nos *tuxs*.

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

Inicialmente, temos um pacote enviado do host para o servidor que contém o *hostname* desejado, pedindo o seu endereço de *IP*. Depois, o servidor responde com um pacote que contém o endereço *IP* do *hostname*.

## Experiência 6 – Conexões TCP

Nesta experiência foi observado o comportamento do protocolo TCP utilizando para isso a aplicação desenvolvida na primeira parte do trabalho.



### **1 - Quantas conexões TCP foram abertas pela aplicação FTP?**

A aplicação FTP abriu 2 conexões TCP. Uma delas para mandar os comandos FTP ao servidor e receber as respostas dele, e outra para receber os dados enviados pelo servidor e enviar as respostas do cliente (como referido na arquitetura).

### **2 - Em que conexão é transportado o controlo de informação?**

O controlo de informação é transportado na conexão TCP responsável pela troca de comandos, ou seja, a primeira conexão referida acima.

### **3 - Quais as fases da conexão TCP?**

Uma conexão TCP pode ser repartida em três fases distintas. Uma primeira fase correspondente ao estabelecimento da conexão, uma segunda fase relativa à troca de dados, e uma última fase que é o encerramento da conexão.

### **4 - Como é que o mecanismo ARQ TCP funciona? Quais os campos TCP relevantes? Qual a informação relevante observada nos logs?**

O TCP (*Transmission Control Protocol*) utiliza o mecanismo ARQ (*Automatic Repeat Request*). É um mecanismo que controla os erros, ou seja, tem por missão corrigir erros na transmissão de dados. Além disso, este método utiliza o protocolo de janela deslizante. Campos TCP relevantes: “*acknowledgment numbers*”, “*tamanho da janela*” e o “*número de sequência*”.

### **5 - Como é que o mecanismo de controlo de congestão TCP funciona? Como é que o fluxo de dados da conexão evoluiu ao longo do tempo? Está de acordo com o mecanismo de controlo de congestão TCP?**

Basicamente no mecanismo de controlo de congestão TCP, ele (TCP) mantém uma janela de congestão que consiste numa estimativa do número de octetos que a rede consegue encaminhar. Não envia mais octetos do que o mínimo da janela definida pelo recetor e pela janela de congestão. E quando se perde um pacote a janela de congestão é dividida por dois.

Aquando do início do primeiro download no tux53, registamos que a taxa de transferência de dados aumentou, chegando esta taxa a um pico após poucos segundos. Quando iniciamos o segundo download verificamos uma descida drástica seguida de uma subida que estabilizou relativamente (apesar de ainda ter algumas perturbações) num nível mais baixo do que estava inicialmente.

Assim, conclui-se que o fluxo de dados da conexão está de acordo com o mecanismo de controlo de congestão TCP pois quando a rede estava mais congestionada tinha um *bitrate* menor.

### **6 - De que forma é afetada a conexão de dados TCP pelo aparecimento de uma segunda conexão TCP? Como?**

O aparecimento de uma segunda conexão TCP, fez com que existisse uma queda na taxa de transmissão, pois quando existe uma transferência de dados em simultâneo a taxa de transferência é dividida de igual forma para cada ligação.

## Conclusões

O segundo trabalho da unidade curricular de Redes de Computadores teve como objetivo a configuração de uma rede e a implementação de uma aplicação de download.

Após a sua conclusão, o grupo interiorizou todos os conceitos necessários para uma estável e coerente implementação do que era pedido no guião de trabalho. Além disso, percebeu como estes estão constantemente presentes no nosso quotidiano.

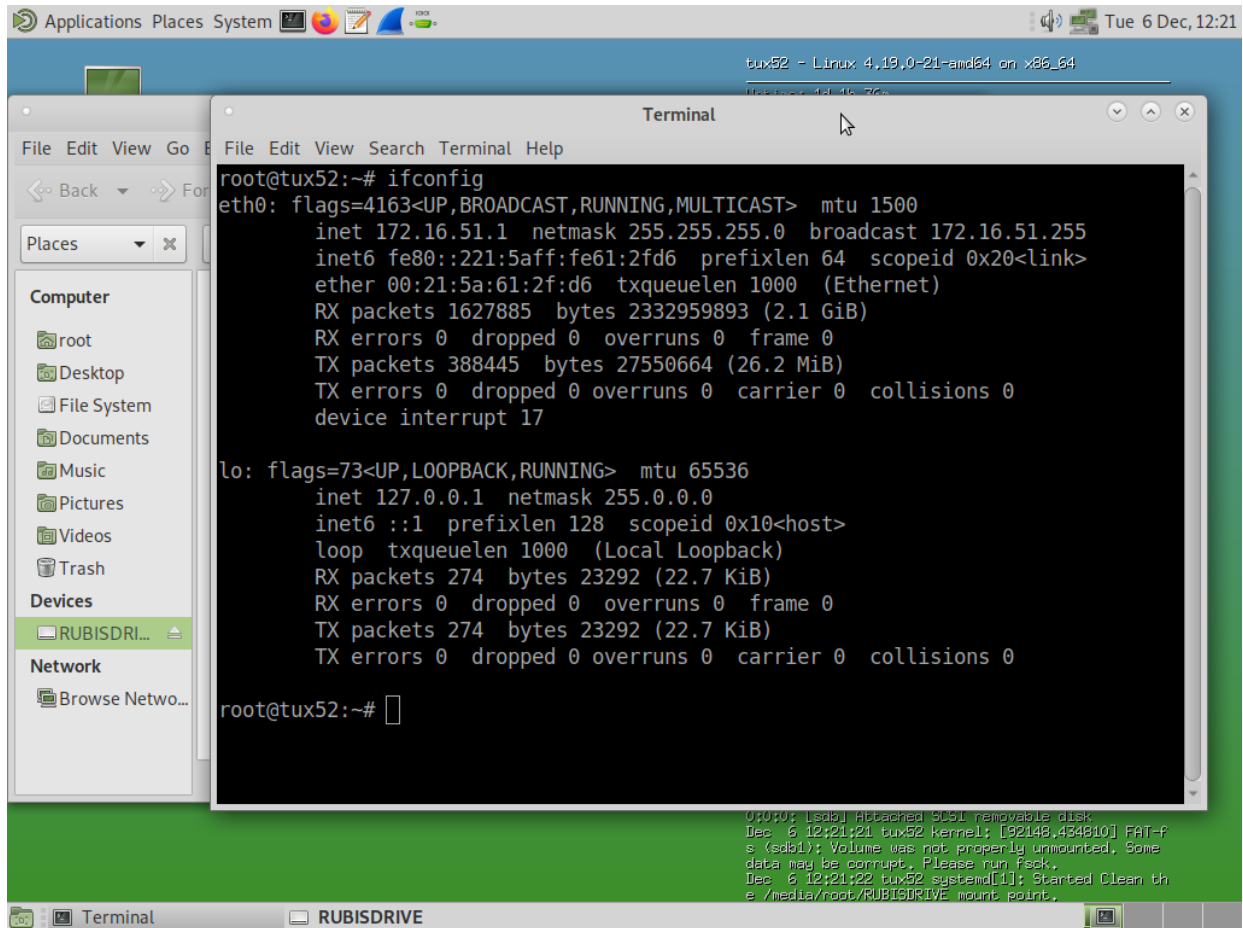
Em suma, o trabalho foi concluído com sucesso, tendo-se cumprido todos os objetivos estabelecidos.

## Referências

- Manuel Ricardo Lab 2 - Computer Networks. 2014
- J. Postel, J. Reinolds File Transfer Protocol 1985
- T. Berners-Lee Uniform Resource Locators 1994

## Anexos

### Ifconfig tux52

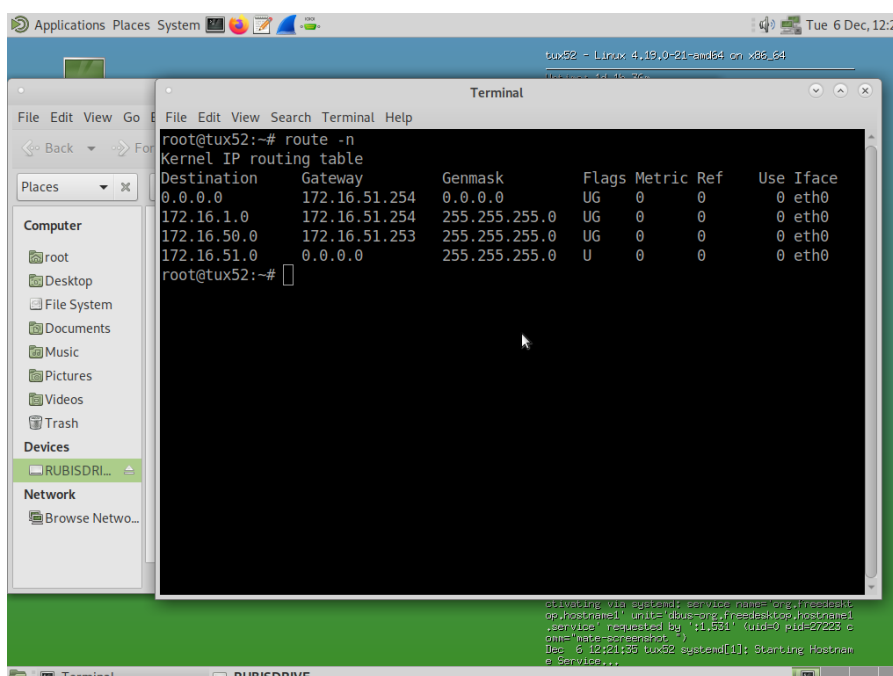


```
root@tux52:~# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 172.16.51.1 netmask 255.255.255.0 broadcast 172.16.51.255
    inet6 fe80::221:5aff:fe61:2fd6 prefixlen 64 scopeid 0x20<link>
    ether 00:21:5a:61:2f:d6 txqueuelen 1000 (Ethernet)
    RX packets 1627885 bytes 2332959893 (2.1 GiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 388445 bytes 27550664 (26.2 MiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
    device interrupt 17

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 274 bytes 23292 (22.7 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 274 bytes 23292 (22.7 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@tux52:~#
```

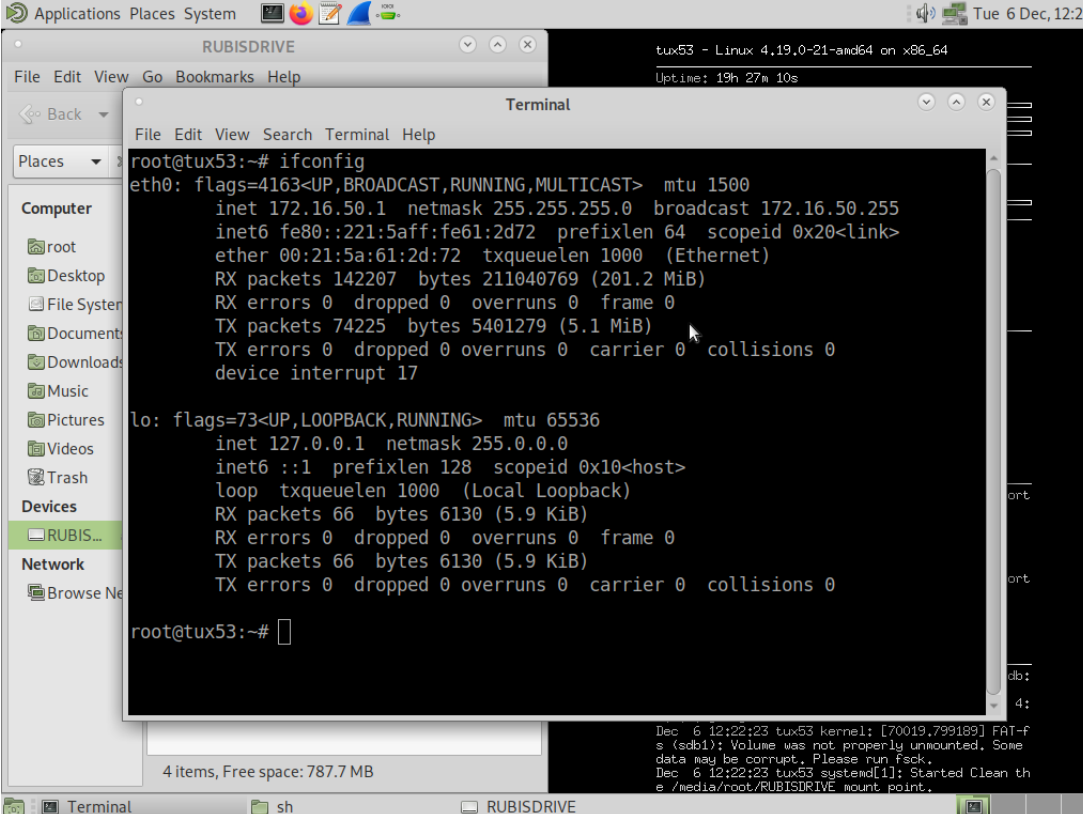
### Route -n tux52



```
root@tux52:~# route -n
Kernel IP routing table
Destination Gateway Genmask Flags Metric Ref Use Iface
0.0.0.0 172.16.51.254 0.0.0.0 UG 0 0 0 eth0
172.16.1.0 172.16.51.254 255.255.255.0 UG 0 0 0 eth0
172.16.50.0 172.16.51.253 255.255.255.0 UG 0 0 0 eth0
172.16.51.0 0.0.0.0 255.255.255.0 U 0 0 0 eth0

root@tux52:~#
```

## Ifconfig tux53

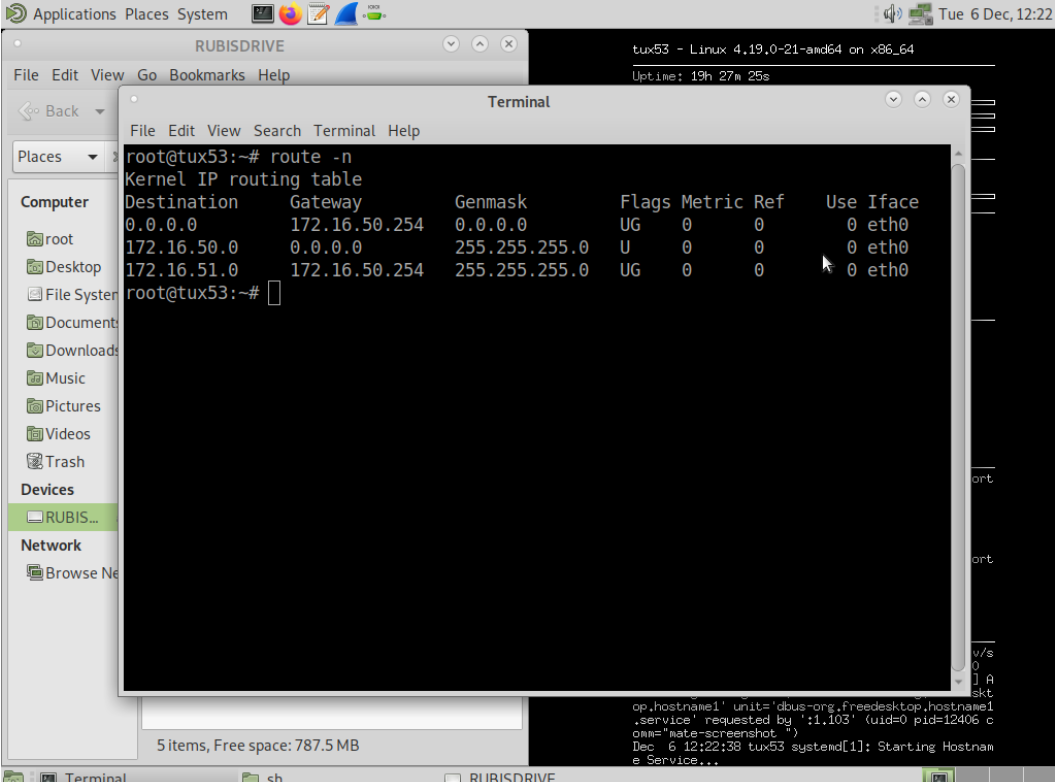


```
root@tux53:~# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 172.16.50.1 netmask 255.255.255.0 broadcast 172.16.50.255
    inet6 fe80::221:5aff:fe61:2d72 prefixlen 64 scopeid 0x20<link>
    ether 00:21:5a:61:2d:72 txqueuelen 1000 (Ethernet)
    RX packets 142207 bytes 211040769 (201.2 MiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 74225 bytes 5401279 (5.1 MiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
    device interrupt 17

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 66 bytes 6130 (5.9 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 66 bytes 6130 (5.9 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@tux53:~#
```

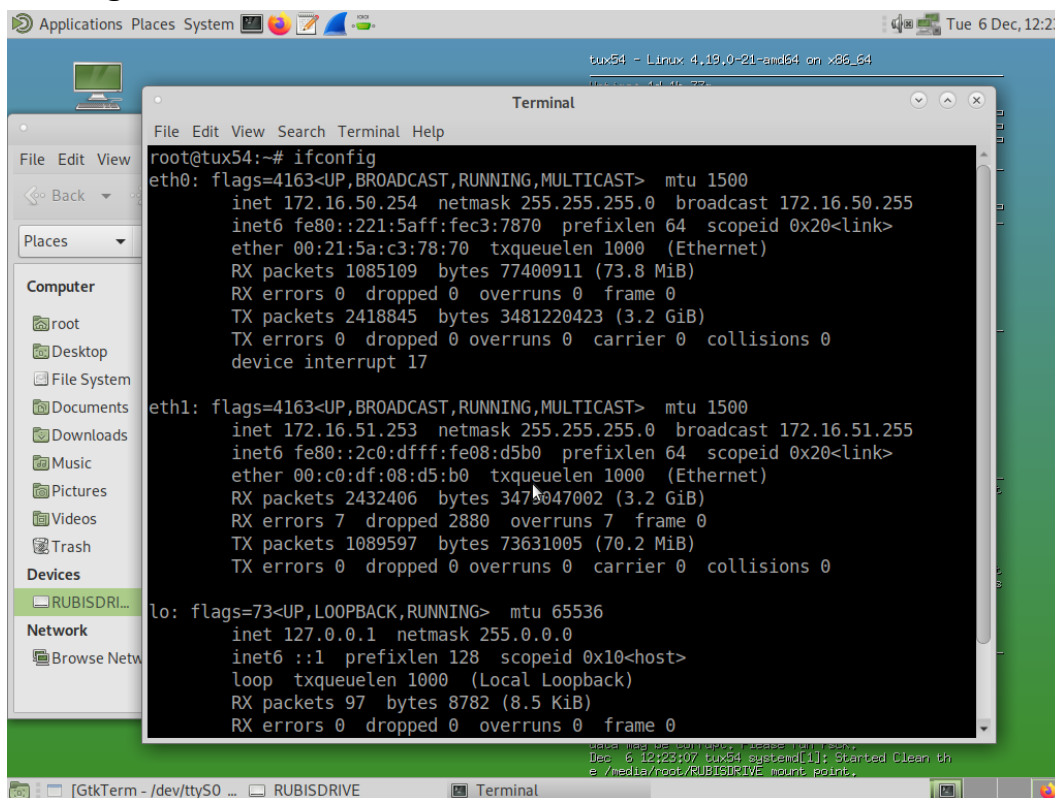
## Route -n tux53



```
root@tux53:~# route -n
Kernel IP routing table
Destination Gateway Genmask Flags Metric Ref Use Iface
0.0.0.0 172.16.50.254 0.0.0.0 UG 0 0 0 eth0
172.16.50.0 0.0.0.0 255.255.255.0 U 0 0 0 eth0
172.16.51.0 172.16.50.254 255.255.255.0 UG 0 0 0 eth0

root@tux53:~#
```

## Ifconfig tux54

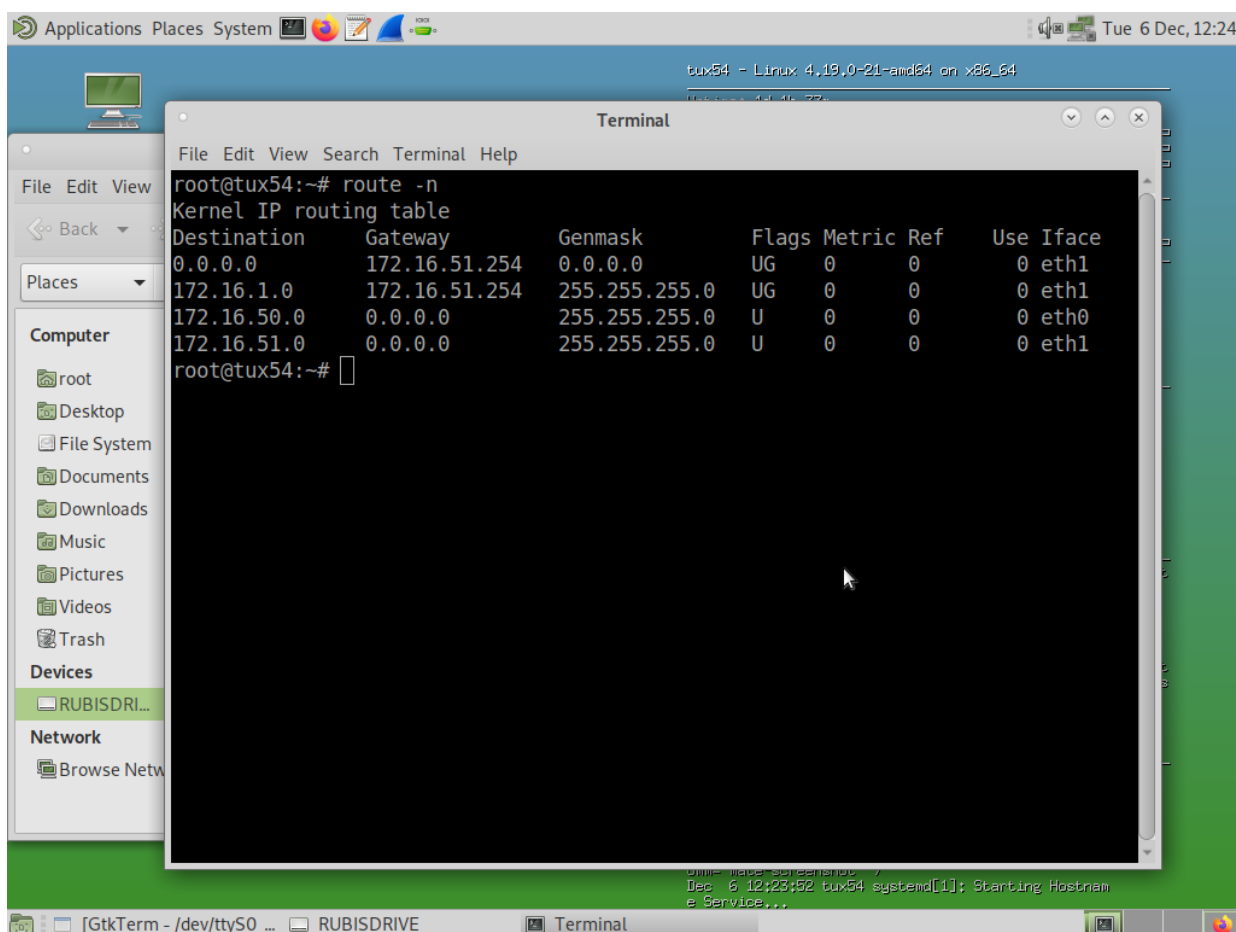


```
root@tux54:~# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 172.16.50.254 netmask 255.255.255.0 broadcast 172.16.50.255
    inet6 fe80::221:5aff:fec3:7870 prefixlen 64 scopeid 0x20<link>
    ether 00:21:5a:c3:78:70 txqueuelen 1000 (Ethernet)
    RX packets 1085109 bytes 77400911 (73.8 MiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 2418845 bytes 3481220423 (3.2 GiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
    device interrupt 17

eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 172.16.51.253 netmask 255.255.255.0 broadcast 172.16.51.255
    inet6 fe80::2c0:dfff:fe08:d5b0 prefixlen 64 scopeid 0x20<link>
    ether 00:c0:df:08:d5:b0 txqueuelen 1000 (Ethernet)
    RX packets 2432406 bytes 3475047002 (3.2 GiB)
    RX errors 7 dropped 2880 overruns 7 frame 0
    TX packets 1089597 bytes 73631005 (70.2 MiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

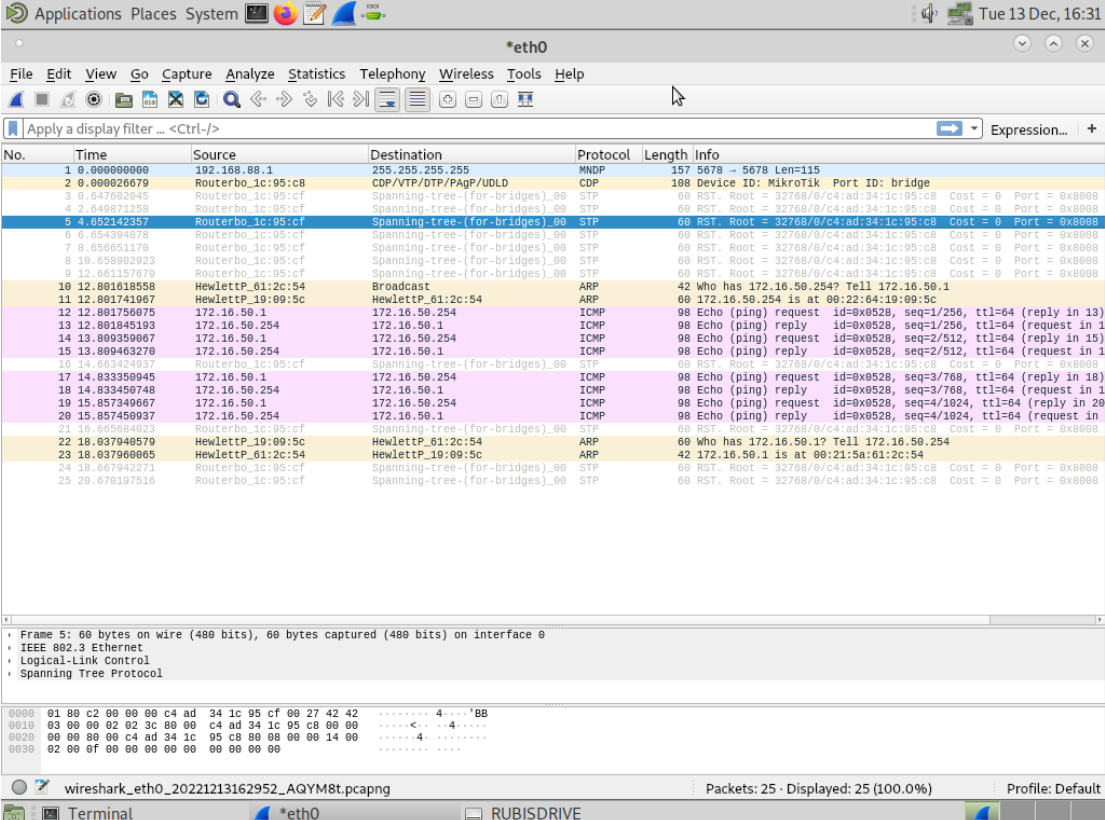
lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 97 bytes 8782 (8.5 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
```

## Route -n tux54



```
root@tux54:~# route -n
Kernel IP routing table
Destination Gateway Genmask Flags Metric Ref Use Iface
0.0.0.0 172.16.51.254 0.0.0.0 UG 0 0 0 eth1
172.16.1.0 172.16.51.254 255.255.255.0 UG 0 0 0 eth1
172.16.50.0 0.0.0.0 255.255.255.0 U 0 0 0 eth0
172.16.51.0 0.0.0.0 255.255.255.0 U 0 0 0 eth1
```

## Exp 1 log



Applications Places System Tue 13 Dec, 16:31

\*eth0

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter ... <Ctrl-/> Expression... +

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	192.168.00.1	255.255.255.255	NDP	157	5678 - 5678 Len=115
2	0.000026679	Routerbo.1c:95:c8	CDP/VTP/DTP/PagP/UDLD	CDP	108	Device ID: Mikrotik Port ID: bridge
3	0.047602045	Routerbo.1c:95:cf	Spanning-tree-(for-bridges).00	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:c8 Cost = 0 Port = 0x0000
4	2.649871258	Routerbo.1c:95:cf	Spanning-tree-(for-bridges).00	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:c8 Cost = 0 Port = 0x0000
5	4.652142357	Routerbo.1c:95:cf	Spanning-tree-(for-bridges).00	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:c8 Cost = 0 Port = 0x0000
6	6.059394878	Routerbo.1c:95:cf	Spanning-tree-(for-bridges).00	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:c8 Cost = 0 Port = 0x0000
7	8.656651178	Routerbo.1c:95:cf	Spanning-tree-(for-bridges).00	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:c8 Cost = 0 Port = 0x0000
8	10.658902023	Routerbo.1c:95:cf	Spanning-tree-(for-bridges).00	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:c8 Cost = 0 Port = 0x0000
9	12.661157679	Routerbo.1c:95:cf	Spanning-tree-(for-bridges).00	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:c8 Cost = 0 Port = 0x0000
10	12.801618558	HewlettP.61:2c:54	Broadcast	ARP	42	Who has 172.16.50.254? Tell 172.16.50.1
11	12.801741967	HewlettP.10:09:5c	HewlettP.61:2c:54	ARP	60	172.16.50.254 is at 00:22:64:19:09:5c
12	12.801756075	172.16.50.1	172.16.50.254	ICMP	98	Echo (ping) request id=0x0528, seq=1/256, ttl=64 (reply in 13)
13	12.801845193	172.16.50.254	172.16.50.1	ICMP	98	Echo (ping) reply id=0x0528, seq=1/256, ttl=64 (request in 1)
14	13.809359067	172.16.50.1	172.16.50.254	ICMP	98	Echo (ping) request id=0x0528, seq=2/512, ttl=64 (reply in 15)
15	13.809463270	172.16.50.254	172.16.50.1	ICMP	98	Echo (ping) reply id=0x0528, seq=2/512, ttl=64 (request in 1)
16	14.065024057	Routerbo.1c:95:cf	Spanning-tree-(for-bridges).00	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:c8 Cost = 0 Port = 0x0000
17	14.833350945	172.16.50.1	172.16.50.254	ICMP	98	Echo (ping) request id=0x0528, seq=3/768, ttl=64 (reply in 18)
18	14.833450748	172.16.50.254	172.16.50.1	ICMP	98	Echo (ping) reply id=0x0528, seq=3/768, ttl=64 (request in 1)
19	15.857340667	172.16.50.1	172.16.50.254	ICMP	98	Echo (ping) request id=0x0528, seq=4/1024, ttl=64 (reply in 20)
20	15.857450937	172.16.50.254	172.16.50.1	ICMP	98	Echo (ping) reply id=0x0528, seq=4/1024, ttl=64 (request in 1)
21	16.065040923	Routerbo.1c:95:cf	Spanning-tree-(for-bridges).00	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:c8 Cost = 0 Port = 0x0000
22	16.037940850	HewlettP.61:2c:54	HewlettP.10:09:5c	ARP	60	Who has 172.16.50.1? Tell 172.16.50.254
23	16.837960665	HewlettP.61:2c:54	HewlettP.10:09:5c	ARP	42	172.16.50.1 is at 00:21:5a:61:2c:54
24	18.667942271	Routerbo.1c:95:cf	Spanning-tree-(for-bridges).00	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:c8 Cost = 0 Port = 0x0000
25	20.678197516	Routerbo.1c:95:cf	Spanning-tree-(for-bridges).00	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:c8 Cost = 0 Port = 0x0000

Frame 5: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0

- IEEE 802.3 Ethernet
- Logical-Link Control
- Spanning Tree Protocol

0000 01 00 c2 00 00 c4 ad 34 1c 95 cf 00 27 42 42 .....4....\*B

0010 03 00 00 02 02 3c 80 00 c4 ad 34 1c 95 c8 00 00 .....<...4.....

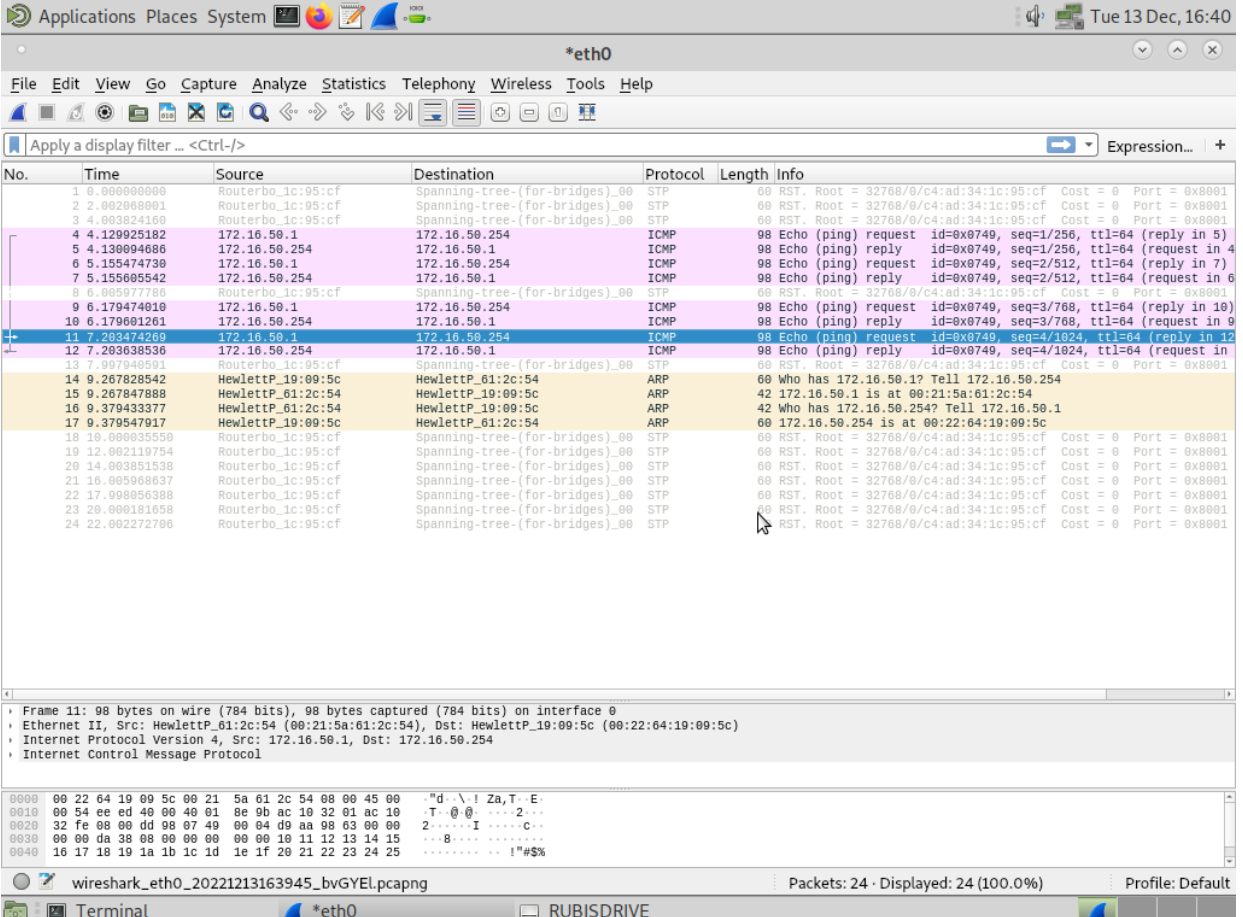
0020 00 00 00 c4 ad 34 1c 95 c8 00 00 00 14 00 .....4.....

0030 02 00 0f 00 00 00 00 00 00 00 00 00 00 ..... .....

wireshark\_eth0\_20221213162952\_AQYM8t.pcapng Packets: 25 · Displayed: 25 (100.0%) Profile: Default

Terminal \*eth0 RUBISDRIVE

## Exp 2 step 6 log



Applications Places System Tue 13 Dec, 16:40

\*eth0

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter ... <Ctrl-/> Expression... +

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	Routerbo.1c:95:cf	Spanning-tree-(for-bridges).00	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:cf Cost = 0 Port = 0x0001
2	2.002068001	Routerbo.1c:95:cf	Spanning-tree-(for-bridges).00	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:cf Cost = 0 Port = 0x0001
3	4.003824100	Routerbo.1c:95:cf	Spanning-tree-(for-bridges).00	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:cf Cost = 0 Port = 0x0001
4	4.129025182	172.16.50.1	172.16.50.254	ICMP	98	Echo (ping) request id=0x0749, seq=1/256, ttl=64 (reply in 5)
5	4.136094080	172.16.50.254	172.16.50.1	ICMP	98	Echo (ping) reply id=0x0749, seq=1/256, ttl=64 (request in 4)
6	5.155474730	172.16.50.1	172.16.50.254	ICMP	98	Echo (ping) request id=0x0749, seq=2/512, ttl=64 (reply in 7)
7	5.155605542	172.16.50.254	172.16.50.1	ICMP	98	Echo (ping) reply id=0x0749, seq=2/512, ttl=64 (request in 6)
8	6.005977786	Routerbo.1c:95:cf	Spanning-tree-(for-bridges).00	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:cf Cost = 0 Port = 0x0001
9	6.179474010	172.16.50.1	172.16.50.254	ICMP	98	Echo (ping) request id=0x0749, seq=3/768, ttl=64 (reply in 10)
10	6.179601261	172.16.50.254	172.16.50.1	ICMP	98	Echo (ping) reply id=0x0749, seq=3/768, ttl=64 (request in 9)
11	7.203474269	172.16.50.1	172.16.50.254	ICMP	98	Echo (ping) request id=0x0749, seq=4/1024, ttl=64 (reply in 12)
12	7.203638536	172.16.50.254	172.16.50.1	ICMP	98	Echo (ping) reply id=0x0749, seq=4/1024, ttl=64 (request in 11)
13	7.097940501	Routerbo.1c:95:cf	Spanning-tree-(for-bridges).00	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:cf Cost = 0 Port = 0x0001
14	9.267828542	HewlettP.10:09:5c	HewlettP.61:2c:54	ARP	60	Who has 172.16.50.1? Tell 172.16.50.254
15	9.267847898	HewlettP.61:2c:54	HewlettP.10:09:5c	ARP	42	172.16.50.1 is at 00:21:5a:61:2c:54
16	9.379433377	HewlettP.61:2c:54	HewlettP.10:09:5c	ARP	60	Who has 172.16.50.254? Tell 172.16.50.1
17	9.379547917	HewlettP.10:09:5c	HewlettP.61:2c:54	ARP	60	172.16.50.254 is at 00:22:64:19:09:5c
18	10.000035550	Routerbo.1c:95:cf	Spanning-tree-(for-bridges).00	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:cf Cost = 0 Port = 0x0001
19	12.002119754	Routerbo.1c:95:cf	Spanning-tree-(for-bridges).00	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:cf Cost = 0 Port = 0x0001
20	14.003851538	Routerbo.1c:95:cf	Spanning-tree-(for-bridges).00	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:cf Cost = 0 Port = 0x0001
21	16.000968637	Routerbo.1c:95:cf	Spanning-tree-(for-bridges).00	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:cf Cost = 0 Port = 0x0001
22	17.990856387	Routerbo.1c:95:cf	Spanning-tree-(for-bridges).00	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:cf Cost = 0 Port = 0x0001
23	20.000181658	Routerbo.1c:95:cf	Spanning-tree-(for-bridges).00	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:cf Cost = 0 Port = 0x0001
24	22.002272706	Routerbo.1c:95:cf	Spanning-tree-(for-bridges).00	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:cf Cost = 0 Port = 0x0001

Frame 11: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface 0

- Ethernet II, Src: HewlettP.61:2c:54 (00:21:5a:61:2c:54), Dst: HewlettP.10:09:5c (00:22:64:19:09:5c)
- Internet Protocol Version 4, Src: 172.16.50.1, Dst: 172.16.50.254
- Internet Control Message Protocol

0000 00 22 64 19 09 5c 00 21 5a 61 2c 54 08 00 45 00 ..d..\.! Za, T. E.

0010 00 54 ee ed 40 00 40 01 8e 9b ac 10 32 01 ac 10 ..T..@.....2...

0020 32 fe 08 00 dd 08 07 40 00 04 09 aa 98 03 00 00 2....I....c...

0030 00 00 0a 38 00 00 00 00 00 18 11 12 13 14 15 .....8.....

0040 16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25 .....!#"\$\$%

wireshark\_eth0\_20221213163945\_bvGYEL.pcapng Packets: 24 · Displayed: 24 (100.0%) Profile: Default

Terminal \*eth0 RUBISDRIVE



## Exp 2 step 9 tux52 log

Applications Places System Tue 13 Dec, 16:44

\*eth0

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter ... <Ctrl-/> Expression...

No.	Time	Source	Destination	Protocol	Length	Info
2	2.002987207	Routerbo_1c:95:c9	Spanning-tree-(for-bridges)_00	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:c9 Cost = 0 Port = 0
3	4.004218204	Routerbo_1c:95:c9	Spanning-tree-(for-bridges)_00	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:c9 Cost = 0 Port = 0
4	6.006331532	Routerbo_1c:95:c9	Spanning-tree-(for-bridges)_00	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:c9 Cost = 0 Port = 0
5	8.008116745	Routerbo_1c:95:c9	Spanning-tree-(for-bridges)_00	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:c9 Cost = 0 Port = 0
6	10.010236009	Routerbo_1c:95:c9	Spanning-tree-(for-bridges)_00	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:c9 Cost = 0 Port = 0
7	12.012307432	Routerbo_1c:95:c9	Spanning-tree-(for-bridges)_00	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:c9 Cost = 0 Port = 0
8	14.014414054	Routerbo_1c:95:c9	Spanning-tree-(for-bridges)_00	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:c9 Cost = 0 Port = 0
9	16.016509912	Routerbo_1c:95:c9	Spanning-tree-(for-bridges)_00	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:c9 Cost = 0 Port = 0
10	18.018618779	Routerbo_1c:95:c9	Spanning-tree-(for-bridges)_00	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:c9 Cost = 0 Port = 0
11	20.020739221	Routerbo_1c:95:c9	Spanning-tree-(for-bridges)_00	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:c9 Cost = 0 Port = 0
12	22.022815193	Routerbo_1c:95:c9	Spanning-tree-(for-bridges)_00	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:c9 Cost = 0 Port = 0
13	24.024931523	Routerbo_1c:95:c9	Spanning-tree-(for-bridges)_00	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:c9 Cost = 0 Port = 0
14	26.026994355	Routerbo_1c:95:c9	Spanning-tree-(for-bridges)_00	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:c9 Cost = 0 Port = 0
15	28.029099162	Routerbo_1c:95:c9	Spanning-tree-(for-bridges)_00	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:c9 Cost = 0 Port = 0
16	30.031296693	Routerbo_1c:95:c9	Spanning-tree-(for-bridges)_00	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:c9 Cost = 0 Port = 0
17	31.909394922	0.0.0.0	255.255.255.255	MNDP	159	5678 -> 5678 Len=117
18	31.909408332	Routerbo_1c:95:c9	CDP/VTP/DTP/PagP/UDLD	CDP	93	Device ID: Mikrotik Port ID: bridge51
19	31.909445707	Routerbo_1c:95:c9	LLDP Multicast	LLDP	110	TTL = 120 System Name = Mikrotik System Description = Mikrotik
20	32.023281942	Routerbo_1c:95:c9	Spanning-tree-(for-bridges)_00	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:c9 Cost = 0 Port = 0
21	34.025385903	Routerbo_1c:95:c9	Spanning-tree-(for-bridges)_00	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:c9 Cost = 0 Port = 0
22	36.027265346	Routerbo_1c:95:c9	Spanning-tree-(for-bridges)_00	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:c9 Cost = 0 Port = 0
23	38.029309944	Routerbo_1c:95:c9	Spanning-tree-(for-bridges)_00	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:c9 Cost = 0 Port = 0
24	40.031424458	Routerbo_1c:95:c9	Spanning-tree-(for-bridges)_00	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:c9 Cost = 0 Port = 0
25	42.033512922	Routerbo_1c:95:c9	Spanning-tree-(for-bridges)_00	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:c9 Cost = 0 Port = 0
26	44.035617161	Routerbo_1c:95:c9	Spanning-tree-(for-bridges)_00	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:c9 Cost = 0 Port = 0

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

- IEEE 802.3 Ethernet
- Logical-Link Control
- Spanning Tree Protocol

wireshark\_eth0\_20221213164202\_eYCAkl.pcapng Packets: 28 · Displayed: 28 (100.0%) Profile: Default

Terminal \*eth0

## Exp 2 step 9 tux53 log

Applications Places System Tue 13 Dec, 16:45

\*eth0

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter ... <Ctrl-/> Expression...

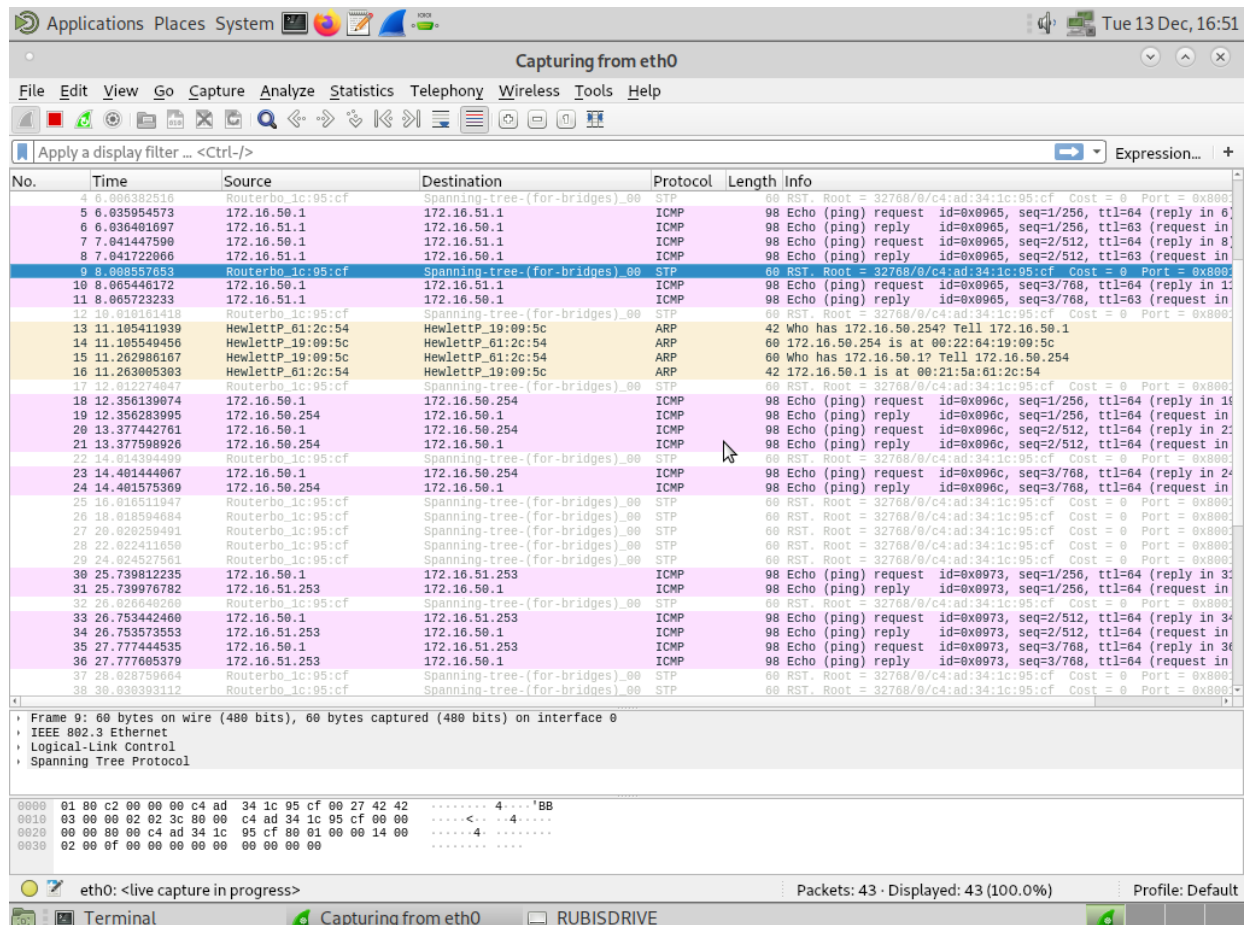
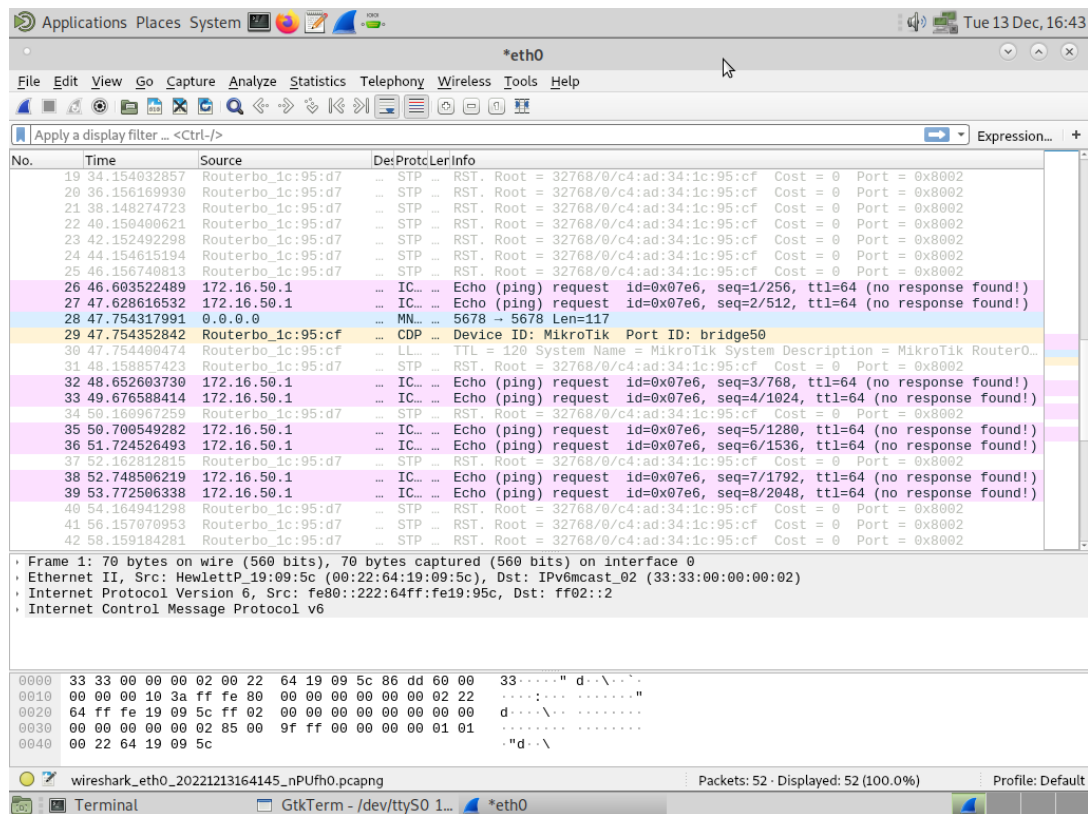
No.	Time	Source	Destination	Protocol	Length	Info
23	36.006157418	Routerbo_1c:95:cf	Spanning-tree-(for-bridges)_00	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:cf Cost = 0 Port = 0
24	38.008353298	Routerbo_1c:95:cf	Spanning-tree-(for-bridges)_00	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:cf Cost = 0 Port = 0
25	40.009483993	Routerbo_1c:95:cf	Spanning-tree-(for-bridges)_00	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:cf Cost = 0 Port = 0
26	42.002683722	Routerbo_1c:95:cf	Spanning-tree-(for-bridges)_00	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:cf Cost = 0 Port = 0
27	44.004856964	Routerbo_1c:95:cf	Spanning-tree-(for-bridges)_00	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:cf Cost = 0 Port = 0
28	46.006703437	Routerbo_1c:95:cf	Spanning-tree-(for-bridges)_00	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:cf Cost = 0 Port = 0
29	48.008878993	Routerbo_1c:95:cf	Spanning-tree-(for-bridges)_00	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:cf Cost = 0 Port = 0
30	50.001065506	Routerbo_1c:95:cf	Spanning-tree-(for-bridges)_00	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:cf Cost = 0 Port = 0
31	52.003179805	Routerbo_1c:95:cf	Spanning-tree-(for-bridges)_00	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:cf Cost = 0 Port = 0
32	54.005324561	Routerbo_1c:95:cf	Spanning-tree-(for-bridges)_00	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:cf Cost = 0 Port = 0
33	56.007524292	Routerbo_1c:95:cf	Spanning-tree-(for-bridges)_00	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:cf Cost = 0 Port = 0
34	58.009681540	Routerbo_1c:95:cf	Spanning-tree-(for-bridges)_00	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:cf Cost = 0 Port = 0
35	60.001120965	Routerbo_1c:95:cf	Spanning-tree-(for-bridges)_00	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:cf Cost = 0 Port = 0
36	62.003994504	Routerbo_1c:95:cf	Spanning-tree-(for-bridges)_00	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:cf Cost = 0 Port = 0
37	64.006128295	Routerbo_1c:95:cf	Spanning-tree-(for-bridges)_00	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:cf Cost = 0 Port = 0
38	66.008289603	Routerbo_1c:95:cf	Spanning-tree-(for-bridges)_00	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:cf Cost = 0 Port = 0
39	68.010456630	Routerbo_1c:95:cf	Spanning-tree-(for-bridges)_00	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:cf Cost = 0 Port = 0
40	68.457181254	172.16.50.1	172.16.50.255	ICMP	98	Echo (ping) request id=0x07e6, seq=1/256, ttl=64 (no re
41	69.482297576	172.16.50.1	172.16.50.255	ICMP	98	Echo (ping) request id=0x07e6, seq=2/512, ttl=64 (no re
42	69.608108064	0.0.0.0	255.255.255.255	MNDP	159	5678 -> 5678 Len=117
43	69.608130413	Routerbo_1c:95:cf	CDP/VTP/DTP/PagP/UDLD	CDP	93	Device ID: Mikrotik Port ID: bridge50
44	69.608135993	Routerbo_1c:95:cf	LLDP Multicast	LLDP	110	TTL = 120 System Name = Mikrotik System Description = Mikrotik
45	70.812618125	Routerbo_1c:95:cf	Spanning-tree-(for-bridges)_00	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:cf Cost = 0 Port = 0
46	70.506293016	172.16.50.1	172.16.50.255	ICMP	98	Echo (ping) request id=0x07e6, seq=3/768, ttl=64 (no re
47	71.530297535	172.16.50.1	172.16.50.255	ICMP	98	Echo (ping) request id=0x07e6, seq=4/1024, ttl=64 (no re
48	72.814759770	Routerbo_1c:95:cf	Spanning-tree-(for-bridges)_00	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:cf Cost = 0 Port = 0
49	72.554290617	172.16.50.1	172.16.50.255	ICMP	98	Echo (ping) request id=0x07e6, seq=5/1280, ttl=64 (no re
50	73.578294009	172.16.50.1	172.16.50.255	ICMP	98	Echo (ping) request id=0x07e6, seq=6/1536, ttl=64 (no re
51	74.816645205	Routerbo_1c:95:cf	Spanning-tree-(for-bridges)_00	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:cf Cost = 0 Port = 0
52	74.602296704	172.16.50.1	172.16.50.255	ICMP	98	Echo (ping) request id=0x07e6, seq=7/1792, ttl=64 (no re
53	75.626394087	172.16.50.1	172.16.50.255	ICMP	98	Echo (ping) request id=0x07e6, seq=8/2048, ttl=64 (no re
54	76.810815993	Routerbo_1c:95:cf	Spanning-tree-(for-bridges)_00	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:cf Cost = 0 Port = 0
55	78.810905617	Routerbo_1c:95:cf	Spanning-tree-(for-bridges)_00	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:cf Cost = 0 Port = 0
56	80.813134144	Routerbo_1c:95:cf	Spanning-tree-(for-bridges)_00	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:cf Cost = 0 Port = 0

Frame 27: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0

- IEEE 802.3 Ethernet
- Logical-Link Control
- Spanning Tree Protocol

wireshark\_eth0\_20221213164124\_AcaQ3U.pcapng Packets: 56 · Displayed: 56 (100.0%) Profile: Default

Terminal \*eth0 RUBISDRIVE





## Exp 3 step 8 log

The image displays two side-by-side Wireshark network traffic capture windows. The left window, titled '\*eth0', shows a list of captured packets with details for frame 29, which is an IEEE 802.3 Ethernet frame containing Logical-Link Control (LLC) and Spanning Tree Protocol (STP) data. The right window, titled '\*eth1', shows a list of captured packets with details for frame 1, which is also an IEEE 802.3 Ethernet frame containing LLC and STP data. Both windows show packet details, including source and destination MAC addresses, and the bottom status bar indicates the number of packets displayed and the profile used.

## Exp 4 step 4 traceroute 1

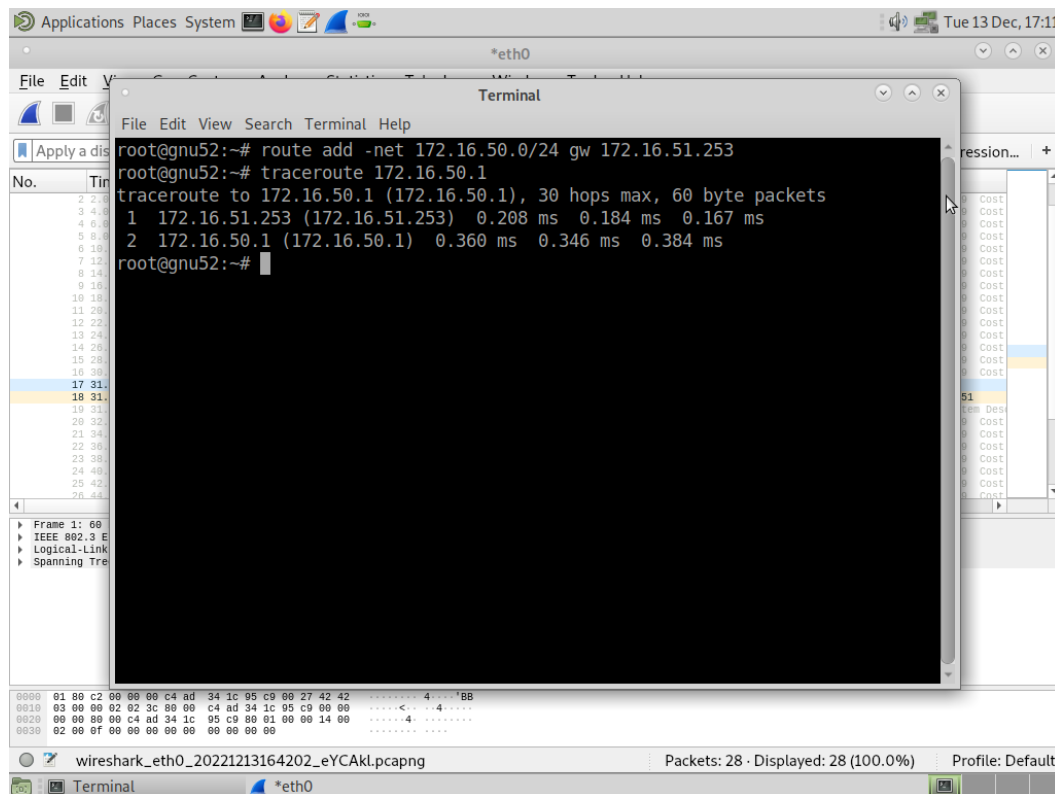
The image shows a terminal window with the output of a traceroute command. The command executed is `tracert 172.16.50.1`. The output shows the path taken by the packets from the source to the destination, including the number of hops, the IP addresses of the intermediate routers, and the round-trip times for each hop. The terminal window is titled 'Terminal' and shows the command prompt `root@gnu52:~#`.

```

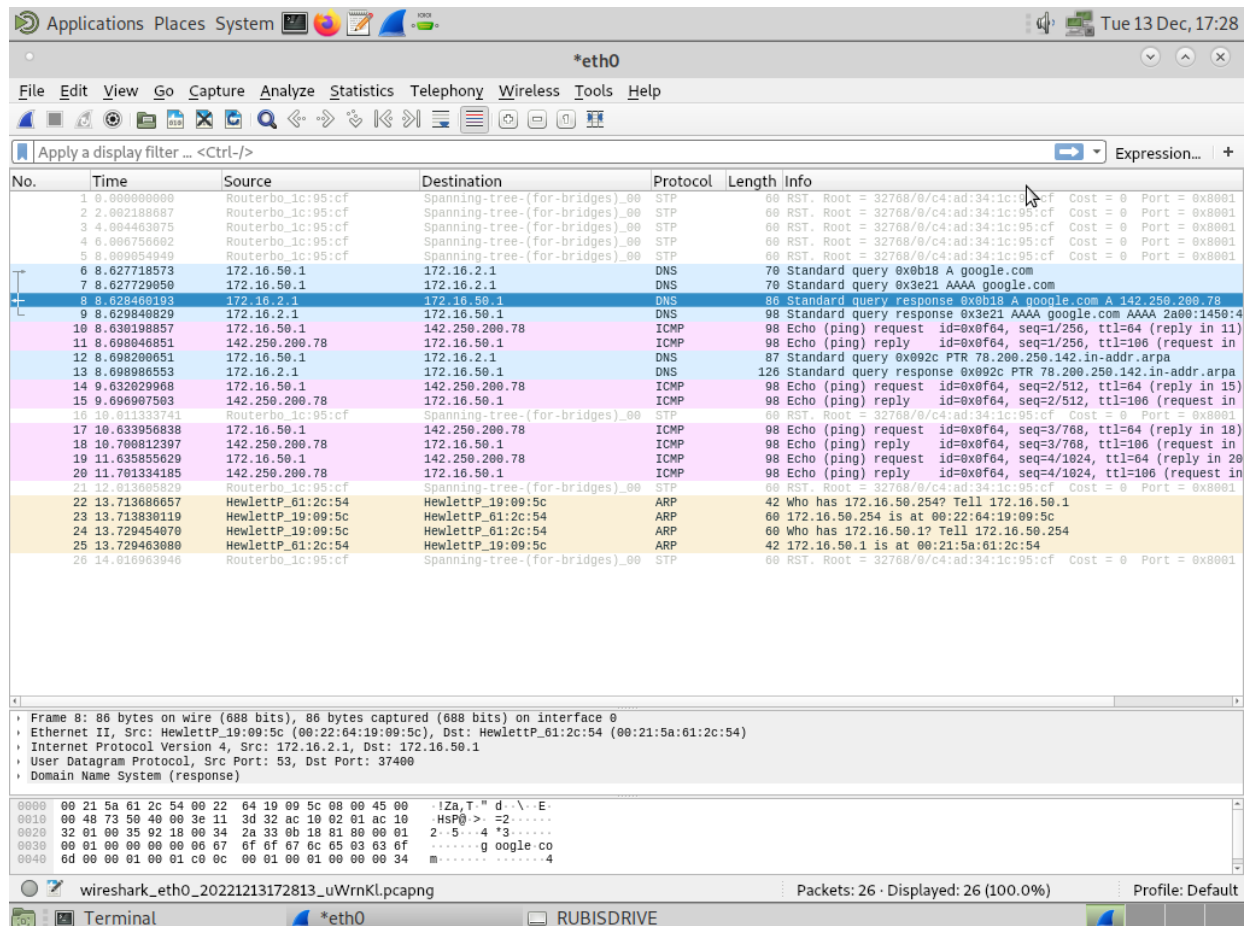
root@gnu52:~# tracert 172.16.50.1
tracert to 172.16.50.1 (172.16.50.1), 30 hops max, 60 byte packets
 1 172.16.51.254 (172.16.51.254) 0.181 ms 0.171 ms 0.183 ms
 2 172.16.51.253 (172.16.51.253) 0.268 ms 0.284 ms 0.267 ms
 3 172.16.50.1 (172.16.50.1) 0.522 ms 0.506 ms 0.488 ms
root@gnu52:~#

```

## Exp 4 step 4 traceroute 2



## Exp 5 log



## Exp 6 log

Applications Places System

Tue 13 Dec, 17:44

Capturing from eth0

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter ... <Ctrl-F>

Expression...

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	Routerbo-1c:95:cf	Spanning-tree-(for-bridges)_00	STP	60	RST: Root = 32768/0/c4:ad:34:1c:95:cf Cost = 0 Port = 0x8000
2	2.402000037	Routerbo-1c:95:cf	Spanning-tree-(for-bridges)_00	STP	60	RST: Root = 32768/0/c4:ad:34:1c:95:cf Cost = 0 Port = 0x8000
3	2.422500096	172.16.50.1	172.16.2.1	DNS	76	Standard query 0xd7f1 A netlab1.fe.up.pt
4	2.743777720	172.16.2.1	172.16.50.1	DNS	92	Standard query response 0xd7f1 A netlab1.fe.up.pt A 192.168.1.1
5	2.744881530	172.16.50.1	192.168.109.136	TCP	74	44952 -> 21 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1
6	2.744938242	192.168.109.136	172.16.50.1	TCP	74	21 -> 44952 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1
7	2.744945471	172.16.50.1	192.168.109.136	TCP	60	44952 -> 21 [ACK] Seq=1 Ack=1 Win=64250 Len=0 TSval=4118517072
8	2.746997197	192.168.109.136	172.16.50.1	FTP	100	Response: 220 Welcome to netlab-FTP server
9	2.747007673	172.16.50.1	192.168.109.136	TCP	60	44952 -> 21 [ACK] Seq=1 Ack=35 Win=64250 Len=0 TSval=4118517074
10	2.747031978	172.16.50.1	192.168.109.136	FTP	76	Request: user rcom
11	2.747593128	192.168.109.136	172.16.50.1	TCP	60	21 -> 44952 [ACK] Seq=35 Ack=11 Win=65280 Len=0 TSval=774403888
12	2.747574855	192.168.109.136	172.16.50.1	FTP	100	Response: 331 Please specify the password.
13	2.747591687	172.16.50.1	192.168.109.136	FTP	76	Request: pass rcom
14	2.748238316	192.168.109.136	172.16.50.1	TCP	60	21 -> 44952 [ACK] Seq=69 Ack=21 Win=65280 Len=0 TSval=774403888
15	2.771391875	192.168.109.136	172.16.50.1	FTP	89	Response: 230 Login successful.
16	2.771431206	172.16.50.1	192.168.109.136	FTP	71	Request: pasv
17	2.772142740	192.168.109.136	172.16.50.1	TCP	60	21 -> 44952 [ACK] Seq=92 Ack=26 Win=65280 Len=0 TSval=774403904
18	2.772388823	192.168.109.136	172.16.50.1	FTP	119	Response: 227 Entering Passive Mode (102,168,109,136,177,76).
19	2.772388816	172.16.50.1	192.168.109.136	TCP	74	57080 -> 45388 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1
20	2.773895166	192.168.109.136	172.16.50.1	TCP	74	45388 -> 57080 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460
21	2.773111719	172.16.50.1	192.168.109.136	TCP	60	57080 -> 45388 [ACK] Seq=1 Ack=1 Win=64250 Len=0 TSval=411851717
22	2.773146011	172.16.50.1	192.168.109.136	FTP	87	Request: retr /files/crab.mp4
23	2.773736869	192.168.109.136	172.16.50.1	TCP	60	21 -> 44952 [ACK] Seq=145 Ack=47 Win=65280 Len=0 TSval=774403906
24	2.773877189	192.168.109.136	172.16.50.1	FTP	145	Response: 150 Opening BINARY mode data connection for /files/crab.mp4
25	2.774240774	192.168.109.136	172.16.50.1	FTP-DATA	1514	FTP Data: 1448 bytes (PASV) (retr /files/crab.mp4)
26	2.774254114	172.16.50.1	192.168.109.136	TCP	60	57080 -> 45388 [ACK] Seq=1 Ack=1449 Win=64128 Len=0 TSval=411851718
27	2.774324393	192.168.109.136	172.16.50.1	FTP-DATA	1514	FTP Data: 1448 bytes (PASV) (retr /files/crab.mp4)
28	2.774358355	172.16.50.1	192.168.109.136	TCP	60	57080 -> 45388 [ACK] Seq=1 Ack=2897 Win=64128 Len=0 TSval=411851719
29	2.774480670	192.168.109.136	172.16.50.1	FTP-DATA	1514	FTP Data: 1448 bytes (PASV) (retr /files/crab.mp4)
30	2.774488882	172.16.50.1	192.168.109.136	TCP	60	57080 -> 45388 [ACK] Seq=1 Ack=4345 Win=64128 Len=0 TSval=411851720
31	2.774604857	192.168.109.136	172.16.50.1	FTP-DATA	1514	FTP Data: 1448 bytes (PASV) (retr /files/crab.mp4)
32	2.774612190	172.16.50.1	192.168.109.136	TCP	60	57080 -> 45388 [ACK] Seq=1 Ack=5793 Win=64128 Len=0 TSval=411851721
33	2.774726889	192.168.109.136	172.16.50.1	FTP-DATA	1514	FTP Data: 1448 bytes (PASV) (retr /files/crab.mp4)
34	2.774733595	172.16.50.1	192.168.109.136	TCP	60	57080 -> 45388 [ACK] Seq=1 Ack=7241 Win=64128 Len=0 TSval=411851722
35	2.774850970	192.168.109.136	172.16.50.1	FTP-DATA	1514	FTP Data: 1448 bytes (PASV) (retr /files/crab.mp4)

\* Frame 7: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0

\* Ethernet II, Src: HewlettP\_01:2c:54 (00:21:5a:01:2c:54), Dst: HewlettP\_10:09:5c (00:22:64:10:09:5c)

\* Internet Protocol Version 4, Src: 172.16.50.1, Dst: 192.168.109.136

\* Transmission Control Protocol, Src Port: 44952, Dst Port: 21, Seq: 1, Ack: 1, Len: 0

eth0: <live capture in progress>

Packets: 42031 · Displayed: 42031 (100.0%)

Profile: Default

Terminal

Capturing from eth0

Computer

RUBISDRIVE

## Download.c

```

1. #include "Download.h"
2. #include <stdio.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 <string.h>
9. #include <netdb.h>
10. #include <regex.h>
11. #include <fcntl.h>
12.
13.
14. int main(int argc, char **argv) {
15.
16.     FILE* sockFile;
17.     int sockfd;
18.     int sockfdrecv;
19.     info h_info;
20.
21.     if (argc != 2) {
22.         fprintf(stderr, "Usage: Download ftp://[<user>:<password>@]<host>/<url-
path>\n");
23.         exit(-1);
24.     }
25.
26.     if (parseInput(argv[1], &h_info) == -1) {
27.         fprintf(stderr, "Usage: Download ftp://[<user>:<password>@]<host>/<url-
path>\n");
28.         exit(-1);
29.     }

```

```
30.
31.     if (getIp(&h_info) == -1) {
32.         fprintf(stderr, "Error on getIp\n");
33.         exit(-1);
34.     }
35.
36.     printf("\n[!] DOWNLOAD PARAMETERS :\n");
37.     printf("    - Host: %s\n", h_info.host);
38.     printf("    - User: %s\n", h_info.user);
39.     printf("    - Password: %s\n", h_info.pass);
40.     printf("    - Path: %s\n", h_info.path);
41.     printf("    - Filename: %s\n\n", h_info.file_name);
42.
43.     //Open connection 1
44.     if (openConnection(&sockfd, h_info.ip, SERVER_PORT) == -1) {
45.         fprintf(stderr, "Error on openConnection\n");
46.         exit(-1);
47.     }
48.     printf("[+] CONNECTION ESTABLISHED [%s:%d]\n", h_info.ip, SERVER_PORT);
49.
50.     sockFile = fdopen(sockfd, "r");
51.     recvMSG(sockFile);
52.
53.     char cmd[516];
54.
55.     //Login set user
56.     sprintf(cmd, "user %s\n", h_info.user);
57.     if (sendCMD(sockfd, cmd) == -1) {
58.         perror("CMD error user");
59.         exit(-1);
60.     }
61.     if (recvMSG(sockFile) == -1) {
62.         perror("MSG error user");
63.         exit(-1);
64.     }
65.
66.     //Login set pass
67.     sprintf(cmd, "pass %s\n", h_info.pass);
68.     if (sendCMD(sockfd, cmd) == -1) {
69.         perror("CMD error pass");
70.         exit(-1);
71.     }
72.     if (recvMSG(sockFile) == -1) {
73.         perror("MSG error pass");
74.         exit(-1);
75.     }
76.
77.     //Set to passive
78.     sprintf(cmd, "pasv\n");
79.     if (sendCMD(sockfd, cmd) == -1) {
80.         perror("CMD error pasv");
81.         exit(-1);
82.     }
83.
84.     char iprecv[100];
85.     char portrecv[100];
86.
87.     if (recvMSGpasv(sockFile, iprecv, portrecv) == -1) {
88.         perror("MSG error pasv");
89.         exit(-1);
90.     }
91.
92.     //open connection 2
93.     if (openConnection(&sockfdrecv, iprecv, atoi(portrecv)) == -1) {
94.         fprintf(stderr, "Error on openConnection\n");
95.         exit(-1);
```

```
96.     }
97.     printf("[+] CONNECTION ESTABLISHED [%s:%s]\n", iprecv, portrecv);
98.
99.     //Set to retr
100.    sprintf(cmd, "retr %s\n", h_info.path);
101.    if (sendCMD(sockfd, cmd) == -1) {
102.        perror("CMD error pasv");
103.        exit(-1);
104.    }
105.    if (recvMSG(sockFile) == -1) {
106.        perror("MSG error pass");
107.        exit(-1);
108.    }
109.
110.    if (saveToFile(sockfdrecv, h_info.file_name) == -1) {
111.        perror("Save to file error");
112.        exit(-1);
113.    }
114.    printf("\n[+] FILE SAVED IN CWD [%s]\n\n", h_info.file_name);
115.
116.    if (closeConnection(sockfdrecv) == -1) exit(-1);
117.    printf("[+] CONNECTION CLOSE [%s:%s]\n", iprecv, portrecv);
118.
119.    if (closeConnection(sockfd) == -1) exit(-1);
120.    printf("[+] CONNECTION CLOSE [%s:%d]\n", h_info.ip, SERVER_PORT);
121.
122.    return 0;
123. }
124.
125. int openConnection(int* fd, char* ip, int port) {
126.     struct sockaddr_in server_addr;
127.
128.     /*server address handling*/
129.     bzero((char *) &server_addr, sizeof(server_addr));
130.     server_addr.sin_family = AF_INET;
131.     server_addr.sin_addr.s_addr = inet_addr(ip);    /*32 bit Internet address
network byte ordered*/
132.     server_addr.sin_port = htons(port);            /*server TCP port must be
network byte ordered */
133.
134.     /*open a TCP socket*/
135.     if ((*fd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
136.         perror("socket()");
137.         return -1;
138.     }
139.     /*connect to the server*/
140.     if (connect(*fd, (struct sockaddr *) &server_addr, sizeof(server_addr)) <
0) {
141.         perror("connect()");
142.         return -1;
143.     }
144.
145.     return 0;
146. }
147.
148. int closeConnection(int fd) {
149.     if (close(fd)<0) {
150.         perror("close()");
151.         return -1;
152.     }
153.     return 0;
154. }
155.
156. int getIp(info* h_info) {
157.     struct hostent *h;
158.
```

```
159.     if ((h = gethostbyname(h_info->host)) == NULL) {
160.         perror("gethostbyname()");
161.         return -1;
162.     }
163.
164.     strcpy(h_info->host_name, h->h_name);
165.     strcpy(h_info->ip, inet_ntoa(*(struct in_addr *) h->h_addr));
166.
167.     return 0;
168. }
169.
170. int sendCMD(int sockfd, char* cmd) {
171.     int bytes = send(sockfd, cmd, strlen(cmd), 0);
172.     if (bytes < 0) {
173.         perror("sendCMD");
174.         return(-1);
175.     }
176.     return 0;
177. }
178.
179. int recvMSG(FILE* fd) {
180.     char *buf;
181.     size_t rbytes = 0;
182.     while (1){
183.         getline(&buf, &rbytes, fd);
184.         //printf("< %s", buf);
185.         if (buf[3] == ' '){
186.             long code = strtol(buf, &buf, 10);
187.             //printf("code: %d\n", code);
188.             if (code == 550 || code == 530)
189.             {
190.                 printf("Command error\n");
191.                 return -1;
192.             }
193.             break;
194.         }
195.     }
196.     return 0;
197. }
198.
199. int recvMSGpasv(FILE* fd, char* iprecv, char* portrecv) {
200.     char* msg;
201.     char ip1[10], ip2[10], ip3[10], ip4[10], ip5[10], ip6[10];
202.     size_t rbytes = 0;
203.
204.     getline(&msg, &rbytes, fd);
205.     //printf("< %s", msg);
206.
207.     sscanf(msg, IP1_REGEX, ip1);
208.     sscanf(msg, IP2_REGEX, ip2);
209.     sscanf(msg, IP3_REGEX, ip3);
210.     sscanf(msg, IP4_REGEX, ip4);
211.     sscanf(msg, IP5_REGEX, ip5);
212.     sscanf(msg, IP6_REGEX, ip6);
213.
214.     sprintf(iprecv, "%s.%s.%s.%s", ip1, ip2, ip3, ip4);
215.     sprintf(portrecv, "%d", (atoi(ip5) * 256) + atoi(ip6));
216.
217.     //printf("ip | port : %s | %s", iprecv, portrecv);
218.
219.     return 0;
220. }
221.
222. int parseInput(char* input, info* h_info) {
223.
224.     strcpy(h_info->pass, "");
```

```
225.
226.         sscanf(input, HOST_REGEX, h_info->host);
227.         sscanf(input, USER_REGEX, h_info->user);
228.         sscanf(input, PASSWORD_REGEX, h_info->pass);
229.         sscanf(input, PATH_REGEX, h_info->path);
230.         strcpy(h_info->file_name, strchr(h_info->path, '/') + 1);
231.
232.         if (strcmp(h_info->pass, "") == 0){
233.             strcpy(h_info->user, "anonymous");
234.             strcpy(h_info->pass, "pass");
235.             sscanf(input, HOST2_REGEX, h_info->host);
236.         }
237.
238.         /*
239.         printf("host: %s\n", h_info->host);
240.         printf("user: %s\n", h_info->user);
241.         printf("pass: %s\n", h_info->pass);
242.         printf("path: %s\n", h_info->path);
243.         printf("filename: %s\n", h_info->file_name);
244.         */
245.
246.         return 0;
247.     }
248.
249.     int saveToFile(int fd, char* filename) {
250.         int file;
251.         int rbytes;
252.         char buf[512];
253.         if ((file = open(filename, O_WRONLY | O_CREAT, 0777)) == -1) {
254.             printf("Cannot open/create file\n");
255.             return -1;
256.         }
257.
258.         while((rbytes = read(fd, buf, sizeof(buf))) != 0){
259.             if (rbytes > 0) {
260.                 write(file, buf, rbytes);
261.                 //printf("%s", buf);
262.             }
263.         }
264.         close(file);
265.
266.         return 0;
267.     }
```

## Download.h

```
1. #ifndef _DOWNLOAD_H_
2. #define _DOWNLOAD_H_
3.
4. #define HOST_REGEX "%^[^/]*%^[^@]@%^[^/]"
5. #define HOST2_REGEX "%^[^/]*%^[^/]"
6. #define PATH_REGEX "%^[^/]*%^[^/]*s"
7. #define USER_REGEX "%^[^/]*%^[^:]"
8. #define PASSWORD_REGEX "%^[^/]*%^[^:]:%^[^@]"
9.
10. #define IP1_REGEX "%^[^()]*%^[^,]"
11. #define IP2_REGEX "%^[^()]*%^[^,]*%^[^,]"
12. #define IP3_REGEX "%^[^()]*%^[^,]*%^[^,]*%^[^,]"
13. #define IP4_REGEX "%^[^()]*%^[^,]*%^[^,]*%^[^,]*%^[^,]"
14. #define IP5_REGEX "%^[^()]*%^[^,]*%^[^,]*%^[^,]*%^[^,]*%^[^,]"
15. #define IP6_REGEX "%^[^()]*%^[^,]*%^[^,]*%^[^,]*%^[^,]*%^[^,]*%^[^,]"
16.
17. #define SERVER_PORT 21
```



```
18.
19. #include <stdio.h>
20.
21. typedef struct info
22. {
23.     char user[128];        ///< User used for Login
24.     char pass[128];        ///< Password used for Login
25.     char host[256];        ///< Host name in URL
26.     char path[256];        ///< Path to the file
27.     char file_name[128];    ///< Name of the File
28.     char host_name[128];    ///< Host Name from gethostbyname()
29.     char ip[128];          ///< IP Address from gethostbyname()
30.
31. } info;
32.
33. int main(int argc, char **argv);        ///
```

## Notes.txt

```
1. +-----+
2. |      RC LabSetUp CMDs      |
3. +-----+
4.
5.
6. > Cables:
7.
8.     tux52 E0 ---- switch eth2
9.     tux53 E0 ---- switch eth8
10.    tux54 E0 ---- switch eth16
11.    tux54 E1 ---- switch eth17
12.    tux54 S0 ---- switch/router serial port
13.    router eth1 ---- netlab (5.1)
14.    router eth2 ---- switch eth24
15.
16.
17. > Config tux52:
18.
19.     ifconfig eth0 172.16.51.1/24
20.     route add -net 172.16.50.0/24 gw 172.16.51.253
21.     route add -net 172.16.1.0/24 gw 172.16.51.254
```



```
22.    route add default gw 172.16.51.254
23.
24.
25. > Config tux53:
26.
27.    ifconfig eth0 172.16.50.1/24
28.    route add -net 172.16.51.0/24 gw 172.16.50.254
29.    route add default gw 172.16.50.254
30.
31.
32. > Config tux54:
33.
34.    ifconfig eth0 172.16.50.254/24
35.    ifconfig eth1 172.16.51.253/24
36.    echo 1 > /proc/sys/net/ipv4/ip_forward
37.    echo 0 > /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts
38.    route add -net 172.16.1.0/24 gw 172.16.51.254
39.    route add default gw 172.16.51.254
40.
41.
42. > Reset Mikrotik switch/router:
43.
44.    /system reset-configuration
45.
46.
47. > Create bridge 50 and 51:
48.
49.    /interface bridge add name=bridge50
50.    /interface bridge add name=bridge51
51.
52.
53. > Remove port from bridge:
54.
55.    /interface bridge port remove [find interface=ether2]
56.    /interface bridge port remove [find interface=ether8]
57.    /interface bridge port remove [find interface=ether16]
58.    /interface bridge port remove [find interface=ether17]
59.    /interface bridge port remove [find interface=ether24]
60.
61.
62. > Add port to bridge:
63.
64.    /interface bridge port add bridge=bridge50 interface=ether8
65.    /interface bridge port add bridge=bridge50 interface=ether16
66.    /interface bridge port add bridge=bridge51 interface=ether17
67.    /interface bridge port add bridge=bridge51 interface=ether2
68.    /interface bridge port add bridge=bridge51 interface=ether24
69.
70.
71. > Show bridges and ports:
72.
73.    /interface bridge port print brief
74.
75.
76. > Config Mikrotik router:
77.
78.    /ip address add address=172.16.1.59/24 interface=ether1
79.    /ip address add address=172.16.51.254/24 interface=ether2
80.    /ip route add dst-address=172.16.50.0/24 gateway=172.16.51.253
81.    /ip route add dst-address=0.0.0.0/0 gateway=172.16.1.254
82.
83.
84. > Config DNS at tux52, tux53, tux54:
85.
86.    set nameserver to 172.16.1.1 at /etc/resolv.conf
```

## Tux52.sh

```
#set eth0 up
ifconfig eth0 172.16.51.1/24

#add a route to tux53
route add -net 172.16.50.0/24 gw 172.16.51.253

#add a route to RC
route add -net 172.16.1.0/24 gw 172.16.51.254

#add default route
route add default gw 172.16.51.254
```

## Tux53.sh

```
#set eth0 up
ifconfig eth0 172.16.50.1/24

#add a route to tux52
route add -net 172.16.51.0/24 gw 172.16.50.254

#add default route
route add default gw 172.16.50.254
```

## Tux54.sh

```
#set eth0 up
ifconfig eth0 172.16.50.254/24

#set eth1 up
ifconfig eth1 172.16.51.253/24

#enable ip forwarding and ignore broadcasts
echo 1 > /proc/sys/net/ipv4/ip_forward
echo 0 > /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts

#add a route to RC
route add -net 172.16.1.0/24 gw 172.16.51.254

#add default route
route add default gw 172.16.51.254
```