

Faculdade de Engenharia da Universidade do Porto



Lab2 - Computer Networks

Relatório

Redes de Computadores 2025/26

L.EIC025 - Turma 3

Autores:

João Junior, up202306719@up.pt

Miguel Neri, up202006475@up.pt

21 de Dezembro de 2025

Sumário

Este relatório descreve o trabalho realizado no âmbito do Laboratório 2 da unidade curricular de Redes de Computadores, cujo objetivo foi o desenvolvimento de uma aplicação de transferência de dados com protocolos TCP/IP e a configuração e estudo de uma rede de computadores.

Introdução

A resolução deste trabalho foi dividida em 2 partes fundamentais.

A primeira parte focou-se em desenvolver uma aplicação de terminal cujo objetivo era fazer a transferência de dados usando protocolos FTP, como descrito no RFC 959, adotando a sintaxe de um URL, descrita no RFC 1738.

A segunda parte focou-se na configuração de uma rede de computadores através do uso de switches e routers MikroTik dividindo-se em 6 tarefas sendo as mais relevantes a configuração de uma rede IP, a criação de pontes no switch para a conexão das várias redes, a configuração de um router, a criação de um router comercial e a implementação da NAT, a configuração de DNS para cada máquina, e por fim, relacionar ambas as partes através do teste da aplicação criada com a rede configurada.

Parte 1 - Downloader

Arquitetura da Aplicação

A aplicação desenvolvida implementa um cliente FTP com o objetivo de efetuar o *download* de um ficheiro a partir de um URL no formato [ftp://\[user:password@\]host/path](ftp://[user:password@]host/path), para tal foram tidos como referência os princípios do protocolo FTP definidos na RFC 959 e o formato de URLs definido na RFC 1738.

1. Processamento do URL e Resolução DNS:

Inicialmente o URL fornecido como argumento é processado pela função *parseURL*, recorrendo a expressões regulares e a *sscanf* para extrair e organizar a informação necessária ao estabelecimento da ligação. A estrutura *URL* guarda os campos: *host*, *resource*, *file*, *user* e *password* (caso estas credenciais não estejam incluídas a aplicação assume autenticação anónima (*user* = “*anonymous*”/*password* = “*anonymous@*”)).

Após a extração do host, é realizada a resolução DNS através do *gethostbyname*, obtendo-se o endereço IP a utilizar na ligação e são ainda efetuadas validações básicas para assegurar que os campos essenciais do URL ficam corretamente preenchidos.

2. Ligação de Controlo (TCP/21) e Gestão de Respostas:

A comunicação FTP é feita através de duas ligações distintas: uma ligação de controlo e uma ligação de dados. A ligação de controlo é criada com a função *createSocket*, estabelecendo uma sessão TCP para o servidor na porta 21. Todos os comandos são enviados com *sendCommand*, garantindo que cada comando termina com *\r\n*, conforme

definido pelo protocolo. A receção e interpretação das respostas do servidor é assegurada pelo *readResponse*, que implementa uma máquina de estados capaz de lidar com respostas de linha única e respostas multi-linha, devolvendo sempre o código numérico de três dígitos.

3. Autenticação e modo Passivo:

Depois de estabelecida a ligação, a aplicação lê a mensagem de boas-vindas (código 220) e procede à autenticação através de *ftpLogin*, enviando os comandos *USER* e, quando necessário, *PASS*. De seguida, é solicitado o modo passivo com o comando *PASV* (*ftpPassiveMode*). A resposta 227 é analisada em *parsePasv*, que extrai o IP e a porta disponibilizados pelo servidor para a ligação de dados, calculando a porta no formato $p1 * 256 + p2$.

4. Ligação de dados e Transferência do Ficheiro:

Com estes parâmetros, é criada uma segunda ligação TCP (socket de dados) para receber o conteúdo do ficheiro. A transferência é iniciada através do comando *RETR* (*ftpRequestFile*), sendo o ficheiro recebido no socket de dados e escrito localmente em modo binário na função *ftpDownloadFile*. Durante a receção, a aplicação lê blocos de dados e escreve-os no ficheiro, contabilizando os bytes transferidos. No final, o socket de dados é fechado e é lida a resposta final no canal de controlo.

5. Término e Encerramento da Sessão:

Por fim, a sessão é terminada de forma controlada com o comando *QUIT* (*ftpQuit*), encerrando a ligação de controlo. Ao longo do processo, a aplicação valida os códigos de resposta esperados em cada etapa e, caso ocorra um valor fora do previsto, interrompe a execução com mensagens de erro informativas.

Testes e Analise de Resultados

A aplicação foi testada para todos os testes providenciados pelos professores da UC, tendo feito download com sucesso de todos os testes FTP públicos (anônimos ou com autenticação) e aos testes anônimos da NetLab, tendo, nos testes não conseguidos, feito de forma bem sucedida a autenticação com o servidor.

No teste apresentado foi utilizada uma aplicação para a transferência do ficheiro *teste.txt* a partir do servidor FTP *ftp://ftp.netlab.fe.up.pt/pub/teste.txt*. Durante este processo, o Wireshark manteve-se ativo, permitindo a captura e análise dos pacotes FTP trocados entre o cliente e o servidor.

A partir da análise dos dados recolhidos foi possível identificar o ponto de origem e destino de cada ligação envolvida na transferência, bem como observar em detalhe os diferentes pacotes transmitidos ao longo da sessão FTP, incluindo as fases de autenticação, estabelecimento da ligação de dados e transferência efetiva do ficheiro. [anexo 2.6.3.1](#)

Todo o código da aplicação encontra-se no [anexo 1.1](#) e [anexo 1.2](#)

Parte 2 - Configuração e Análise de uma Network

As 6 experiências nas quais é dividida a parte 2 deste projeto segue uma linha ordenada de eventos na qual a sucessão de uma experiência tem como configuração base a configuração final da sua anterior, isto é, uma experiência só pode ser começada caso a sua configuração esteja no mesmo estado do fim da experiência anterior.

Experiência 1 - Configurar uma Rede IP

Esta experiência teve como objetivo a configuração de dois endereços IP em duas máquinas diferentes, Tux63 e Tux64, ligados a um switch. Posteriormente pretendeu-se analisar as rotas, e o funcionamento das tabelas ARP, verificando o comportamento do seu protocolo quando as entradas na tabela são apagadas e uma conexão é gerada entre as duas.

Para esse fim, começamos por ligar a interface if_e1 das máquinas ao switch e configuramos o IP de cada uma através do comando ifconfig :

- ifconfig if_e1 172.16.60.1/24 (Tux 63)
- ifconfig if_e1 172.16.60.254/24 (Tux 64)

Podemos verificar os endereços IP e MAC com o uso do comando ifconfig if_e1 em cada uma das máquinas (para o Tux63 os respetivos endereços IP e MAC eram 172.16.60.1 e 8c:90:2d:fd:a6:09, e para o Tux64 eram 172.16.60.254 e ec:75:0c:c2:17:49).

Por fim, apagamos as entradas na tabela ARP referentes ao IP configurado no Tux63, e usando o Wireshark para analisar a troca de packets entre os pontos de origem e destino da ligação, estabelecemos a conexão do Tux63 para o Tux64. Verificamos que ao estabelecer a conexão, os primeiros packets trocados não foram os ICMP mas sim ARP, evidenciando a criação de novas entradas na tabela ARP.

A obrigatoriedade da criação destas entradas é notória para o estabelecimento de qualquer ligação e acontece devido ao Protocolo ARP ser o que relaciona o endereço IP e o respetivo MAC, ou seja, faz a ligação entre a Link layer e a Network layer.

Os logs e imagens pertinentes relativas a esta experiência estão no [anexo 2.1](#)

Experiência 2 - Implementar duas bridges num switch

O objetivo desta experiência foi a criação de dois isolated broadcast domains usando o switch MikroTik.

Para tal, começamos por configurar o endereço IP do Tux62 - ifconfig if_e1 172.16.61.1/24. De seguida, usamos o terminal GTK no switch para criar duas bridges:

- bridge60, designada para as portas 16 e 19 que correspondem ao Tux63 e Tux64, respetivamente (subnet 172.16.60.0/24)
- bridge61, designada para a porta 13 que corresponde ao Tux62 (subnet 172.16.61.0/24)

Após a criação das duas bridges, removemos as interfaces (portas) que correspondiam aos Tux's usados (portas 13, 16 e 19) da bridge global existente por padrão do sistema e ligamos cada Tux a respetiva bridge.

Para testar a conectividade e o isolamento de cada bridge, usamos o comando ping -d 172.16.60.255 em cada Tux e verificamos que quando este comando era corrido, o ping só chegava as máquinas correspondentes à mesma bridge, isto é, o Tux63 conectava-se com sucesso ao Tux64 e vice versa mas a conexão não chegava ao Tux62, e o Tux62 não se conectava com nenhum dos dois outros Tux's.

Chegamos a esta conclusão ao analisar os pacotes ICMP na conexão entre as 3 máquinas, sendo evidenciado por não haverem packets de retorno do Tux62 quando a conexão era gerada pelos Tux's da outra bridge.

Os logs e imagens pertinentes relativas a esta experiência estão no [anexo 2.2](#)

Experiência 3 - Configurar um Router em Linux

Esta experiência teve como objetivo configurar o Tux64 de forma a torna-lo num router que serve como ligação entre as duas subnets 172.16.60.0/24 e 172.16.61.0/24, de forma a que os Tux's 62 e 63 que se encontram em bridges diferentes, possam comunicar entre si.

Para isto, começamos por configurar a interface if_e2 do Tux64 com o IP 172.16.61.253 e liga-la à porta 20 do switch. Depois removemos a respetiva interface da bridge que estava ligada por padrão e conectamos à bridge61 onde o Tux62 estava isolado. Posteriormente, ativamos o IP forwarding e desativamos o ICMP echo-ignore-broadcast no Tux64. Por fim, usamos os comandos de route:

- route add -net 172.16.60.0/24 gw 172.16.61.253 (no Tux62)
- route add -net 172.16.61.0/24 gw 172.16.60.254 (no Tux63)

para que cada máquina usasse o Tux64 como gateway para a outra bridge/subnet.

Para testar a ligação entre os Tux's 62 e 63, assegurando que a configuração do router foi feita de forma bem sucedida, usou-se o comando ping do Tux63 para todos os outros IPs. Os packets ICMP e ARP chegam todos ao destino, o que confirma que a ligação foi bem sucedida para todos os pings feitos.

O último teste desta experiência foi apagar as entradas na tabela ARP dos 3 Tux's e mais uma vez usar o comando ping, desta vez apenas do Tux63 para o Tux62. Usamos duas instâncias do Wireshark no Tux64 para analisar a conexão, uma para cada interface (e1 e e2). Concluímos então que cada um dos Tux's 63 e 62 faz ARP para o endereço IP do router, e não diretamente entre si. Por sua vez, o router mantém entradas ARP separadas para cada interface o que demonstra corretamente a separação de domínios de broadcast e o papel do router como intermediário.

Os logs e imagens pertinentes relativas a esta experiência estão no [anexo 2.3](#)

Experiência 4 - Configurar um Router Comercial e Implementar NAT

Com esta experiência pretendeu-se configurar e integrar um router comercial com NAT, de forma a disponibilizar acesso à Internet; para isso, partiu-se da rede resultante da experiência anterior, composta por duas LANs em sub-redes distintas (172.16.60.0/24 na *bridge60* e 172.16.61.0/24 na *bridge61*) interligadas pelo Tux64, que atua como router entre ambas. Para estabelecer a ligação entre a rede interna e a rede externa do laboratório, foi adicionado um router comercial MikroTik (Rc), ligando a interface ether1 à rede do laboratório (WAN) e a interface ether2 à nossa rede interna, através da *bridge61*. De seguida, configurou-se o endereço IP da interface interna do RC como 172.16.61.254/24, e a externa como 172.16.1.61/24, mantendo a interface externa na rede 172.16.1.0/24 (subnet que faz conexão com o servidor FTP da feup de endereço 172.16.1.10). Após a integração física e lógica do equipamento, foram atualizadas as tabelas de encaminhamento dos hosts Linux (tux62, tux63 e tux64) de modo a utilizarem o RC (172.16.61.254) como *default gateway* para tráfego destinado ao exterior, tirando partido da configuração padrão do MikroTik, que já inclui NAT (Masquerade) ativo na interface externa, permitindo a tradução automática dos endereços privados quando o tráfego sai para a rede do laboratório.

Antes de proceder aos testes de conectividade externa, validou-se que o encaminhamento interno permanecia funcional e que a integração do RC não comprometia a comunicação entre as duas sub-redes. Para isso, realizaram-se testes de conectividade por ICMP a partir do tux63 (172.16.60.1) para as interfaces relevantes da topologia, nomeadamente para 172.16.60.254 (interface do Tux64 na sub-rede 60), 172.16.61.1 (tux62) e 172.16.61.254 (interface interna do RC), confirmando que o tráfego entre sub-redes continuava a ser encaminhado pelo Tux64 e que o RC era alcançável a partir da sub-rede 172.16.60.0/24. Em paralelo, efetuou-se a captura de tráfego com o Wireshark para observar a troca de *Echo Requests* e *Echo Replies* e confirmar que os pacotes eram efetivamente encaminhados pelas interfaces esperadas.

De seguida, analisou-se a eficiência do encaminhamento quando um host da sub-rede 61 utiliza o RC como *default gateway* para alcançar um destino na sub-rede 60. Ao executar traceroute no tux62 com destino ao tux63 (172.16.60.1), observou-se inicialmente um percurso não otimizado (“hairpin”), em que o primeiro salto correspondia ao RC (172.16.61.254) e só depois ao Tux64 (172.16.61.253), apesar de o Tux64 se encontrar no mesmo segmento e ser o gateway direto para a sub-rede 60. Este comportamento está de acordo com o cenário descrito no guião, onde o encaminhamento via *default gateway* pode introduzir um desvio desnecessário. Ao repetir os testes de conectividade, o RC gerou mensagens ICMP Redirect, indicando ao tux62 que o próximo salto mais eficiente para o destino seria o 172.16.61.253 (Tux64). Esta situação foi confirmada tanto pela saída do ping (mensagens “Redirect Host”) como pelas capturas em Wireshark, onde se observam pacotes ICMP do tipo *Redirect* enviados pelo RC para o host. Para permitir que o sistema utilizasse estas indicações, ativou-se a aceitação de redirects (*accept_redirects*) no tux62 e repetiu-se o traceroute, verificando-se então que o percurso passou a ser mais curto, deixando de incluir o RC e encaminhando diretamente via Tux64.

Por fim, validou-se o objetivo principal da experiência realizando testes de conectividade para a rede do lab a partir da rede interna, obtendo resposta com sucesso. A observação do tráfego capturado confirmou a troca de mensagens ICMP com o destino externo e é consistente com o funcionamento do NAT no MikroTik, permitindo que hosts com endereços

privados nas sub-redes 172.16.60.0/24 e 172.16.61.0/24 comuniquem com a rede do laboratório através da tradução de endereços efetuada no RC.

Os logs e imagens pertinentes relativas a esta experiência estão no [anexo 2.4](#)

Experiência 5 - DNS

Após garantir conectividade externa e NAT na experiência anterior, procedeu-se à configuração do DNS (Domain Name System), permitindo que os hosts resolvessem nomes de domínio para endereços IP. Para isso, configurou-se o resolvedor DNS nos hosts Linux da topologia (tux62, tux63 e tux64), editando o ficheiro /etc/resolv.conf e adicionando o servidor DNS do laboratório nameserver 10.227.20.3.

A configuração foi validada através da execução de ping google.com num dos hosts (tux63). Como se observa na captura, o nome google.com foi corretamente resolvido para um endereço público (142.250.200.142) e o host passou a trocar mensagens ICMP com sucesso, sem perdas. Em paralelo, a captura no Wireshark evidencia a sequência esperada: primeiro surgem mensagens DNS (consulta e resposta) para o servidor 10.227.20.3 sobre UDP/53, e só depois o tráfego ICMP relativo ao ping, confirmando que as queries DNS foram encaminhadas corretamente através do gateway/NAT e que a resposta regressou ao host permitindo a comunicação com o IP resolvido.

Os logs e imagens pertinentes relativas a esta experiência estão no [anexo 2.5](#)

Experiência 6 - Ligações TCP

O objetivo desta última experiência foi relacionar ambas as partes do projeto, usando toda a rede configurada ao longo das experiências anteriores e a aplicação criada na parte 1, de forma a analisar todo o processo de envio e receção dos packets, assim como a sua constituição interna e todos os mecanismos de controlo de congestionamento das ligações TCP.

Num primeiro momento começamos por descarregar e compilar a nossa aplicação para o Tux53 (diferente das experiências anteriores devido à troca de bancada de trabalho). Após isso abrimos o wireshark para detetar e registar todas as transferências de dados e pacotes e corremos a aplicação para fazer a transferência de um pequeno ficheiro: ftp://ftp.netlab.fe.up.pt/pub/teste.txt de cerca de 6 bytes. Através da análise dos registos do Wireshark, verificou-se que, como previsto, a transferência do ficheiro era dividida em várias etapas:

1. Os primeiros dois registos foram de pacotes ARP, isto aconteceu uma vez que esta transferência foi a primeira feita na máquina e ainda não havia entradas na tabela ARP para o destinatário do servidor FTP.
2. De seguida, observa-se uma query DNS para o domínio do servidor FTP (ftp.netlab.fe.up.pt), bem como a respetiva resposta, que associa o nome de domínio ao endereço IP 172.16.1.10. Esta etapa permite que a aplicação utilize o endereço IP correto do servidor.
3. Após a resolução DNS, é iniciado a ligação TCP (SYN, SYN-ACK, ACK) entre o cliente e o servidor FTP, utilizando a porta 21, que corresponde ao canal de controlo do protocolo FTP.

4. No canal de controlo FTP é visível a resposta à ligação TCP (220), a autenticação USER e PASS e a devida resposta do servidor (331 e 230 respectivamente).
5. O cliente solicita o modo passivo através do comando PASV. O servidor responde indicando um novo port TCP a utilizar para a ligação de dados. Este comportamento é típico do FTP em modo passivo, onde o cliente inicia ambas as ligações (controlo e dados).
6. É criada uma nova ligação TCP para o canal de dados. De seguida, observa-se o comando RETR, que solicita o ficheiro ao servidor. O servidor responde com 150 Opening BINARY mode data connection, indicando o início da transferência. Os pacotes seguintes correspondem à transmissão efetiva dos dados do ficheiro.
7. Após a transferência completa, o servidor envia a mensagem 226 Transfer complete, confirmado o sucesso da operação. Finalmente, o cliente envia o comando QUIT, e a ligação TCP é encerrada de forma ordenada (FIN/ACK).

Posteriormente procedemos à mesma experiência com um ficheiro de alguns megabytes e analisamos recorrendo à Statistic Tool I/O Graph do Wireshark para analisar as variações do fluxo/velocidade de transferência dos dados e conseguimos verificar uma variação mínima mesmo quando a transferência é de maior tamanho. [anexo 2.6.2](#)

Num segundo momento, testamos a transferência de um ficheiro com alguns gigabytes de tamanho e, a meio da sua transferência, iniciamos no Tux52 outra transferência de forma a verificar se haveria congestionamento ou fricção na transferência de algum dos dois ficheiros. Com o Wireshark e as ferramentas de estatística usadas anteriormente, pudemos verificar que quando uma segunda transferência é iniciada em coincidência com outra, a velocidade de transferência é reduzida para cerca de metade e estabiliza nesse valor. Isto deve-se aos mecanismos de controlo de congestionamento das ligações TCP por parte dos dois computadores, o que resulta rapidamente numa convergência para valores estáveis ao longo da transferência simultânea de dados de ambas as máquinas. [anexo 6.1.1](#)

Apesar de tudo, os gráficos demonstram que a rede configurada ao longo das experiências cria uma infraestrutura funcional, capaz de suportar protocolos complexos como o FTP.

Os logs e imagens pertinentes relativas a esta experiência estão no [anexo 2.6](#)

Conclusão

Com a realização deste trabalho foi possível consolidar os conhecimentos teóricos e práticos sobre protocolos de transferência de dados em redes de computadores. A implementação de uma aplicação FTP em C, baseada no RFC 959, permitiu compreender o funcionamento do protocolo e a utilização de sockets. Adicionalmente, a configuração completa da infraestrutura de rede reforçou a compreensão dos conceitos de addressing, routing e interligação de redes.

Referências

[RFC 1738: Uniform Resource Locators \(URL\)](#)

[RFC 959: File Transfer Protocol](#)

Beej's Guide to Network Programming _ Brian "Beej Jorgensen" Hall

Lab 2 - Computer Networks _ Manuel Ricardo, Filipe B. Teixeira, Eduardo Nuno Almeida

Anexos

Anexo1 - Código da aplicação:

1.1 Download.h

```
#ifndef DOWNLOAD_H
#define DOWNLOAD_H

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

// ===== Constantes =====

#define MAX_LENGTH 500
#define FTP_PORT 21

// Server response codes (RFC 959)
#define SV_READY_FOR_AUTH      220
#define SV_READY_FOR_PASS      331
#define SV_LOGIN_SUCCESS       230
#define SV_PASSIVE_MODE        227
#define SV_CONN_ALREADY_OPEN   125
#define SV_READY_FOR_TRANSFER  150
#define SV_TRANSFER_COMPLETE   226
#define SV_GOODBYE             221

// Parser regular expressions
#define AT                      "@"
#define BAR                     "/"
#define HOST_REGEX              "%*[^/];//%[^/]"
#define HOST_AT_REGEX           "%*[^/];//%*[^@]@%[^/]"
#define RESOURCE_REGEX          "%*[^/];//%*[^/]/%s"
#define USER_REGEX               "%*[^/];//%[^:/]"
#define PASS_REGEX               "%*[^/];//%*[^:]://%[^@\n$]"
#define RESPCODE_REGEX          "%d"
#define PASSIVE_REGEX           "%*[^()(%d,%d,%d,%d,%d)%*[^\n$)]"
```

```

// Default credentials for anonymous login
#define DEFAULT_USER      "anonymous"
#define DEFAULT_PASSWORD "anonymous@"

// ===== Estruturas =====

/**
 * Estrutura para guardar informação do URL
 * Formato: ftp://[user:password@]host/path
 */
typedef struct {
    char host[MAX_LENGTH];           // 'ftp.up.pt'
    char resource[MAX_LENGTH];       // 'pub/files/test.txt'
    char file[MAX_LENGTH];          // 'test.txt'
    char user[MAX_LENGTH];          // 'username'
    char password[MAX_LENGTH];      // 'password'
    char ip[MAX_LENGTH];            // '193.137.29.15'
} URL;

/**
 * Estados da máquina que processa respostas do servidor
 */
typedef enum {
    START,             // Início da leitura
    SINGLE,            // Resposta de linha única (código + espaço)
    MULTIPLE,          // Resposta multi-linha (código + hifen)
    END               // Fim da resposta
} ResponseState;

// ===== Funções de Parsing =====

/**
 * Parse do URL fornecido pelo utilizador
 * @param input String com o URL (ftp://...)
 * @param url Estrutura que será preenchida
 * @return 0 se sucesso, -1 se erro
 */
int parseURL(const char *input, URL *url);

/**
 * Parse da resposta PASV para obter IP e porta
 * Formato: 227 Entering Passive Mode (h1,h2,h3,h4,p1,p2)
 * @param response Resposta do servidor
 * @param ip String que será preenchida com o IP
 * @param port Ponteiro para inteiro que será preenchido com a porta
 * @return 0 se sucesso, -1 se erro
 */
int parsePasv(const char *response, char *ip, int *port);

```

```

// ===== Funções de Rede =====

/**
* Cria socket TCP e conecta ao servidor
* @param ip Endereço IP do servidor
* @param port Porta do servidor
* @return Socket descriptor se sucesso, -1 se erro
*/
int createSocket(const char *ip, int port);

/**
* Lê resposta do servidor FTP usando máquina de estados
* @param socket Descriptor do socket
* @param buffer Buffer que será preenchido com a resposta
* @return Código de resposta (220, 331, etc)
*/
int readResponse(const int socket, char *buffer);

/**
* Envia comando para o servidor FTP
* @param socket Descriptor do socket
* @param cmd Comando (ex: "USER", "PASS", "RETR")
* @param arg Argumento do comando (pode ser NULL)
* @return 0 se sucesso, -1 se erro
*/
int sendCommand(const int socket, const char *cmd, const char *arg);

// ===== Funções FTP =====

/**
* Autentica no servidor FTP
* @param socket Descriptor do socket de controlo
* @param user Nome de utilizador
* @param password Password
* @return Código de resposta do servidor
*/
int ftpLogin(const int socket, const char *user, const char *password);

/**
* Entra em modo passivo
* @param socket Descriptor do socket de controlo
* @param ip String que será preenchida com IP para conexão de dados
* @param port Ponteiro que será preenchido com porta para conexão de dados
* @return Código de resposta do servidor
*/
int ftpPassiveMode(const int socket, char *ip, int *port);

```

```

/**
 * Requisita um ficheiro ao servidor
 * @param socket Descriptor do socket de controlo
 * @param resource Caminho do ficheiro no servidor
 * @return Código de resposta do servidor
 */
int ftpRequestFile(const int socket, const char *resource);

/**
 * Descarrega o ficheiro do servidor
 * @param control_socket Socket de controlo
 * @param data_socket Socket de dados
 * @param filename Nome do ficheiro a criar localmente
 * @return Código de resposta do servidor
 */
int ftpDownloadFile(const int control_socket, const int data_socket, const char
*filename);

/**
 * Fecha a conexão FTP
 * @param control_socket Socket de controlo
 * @return 0 se sucesso, -1 se erro
 */
int ftpQuit(const int control_socket);

#endif// DOWNLOAD_H

```

1.2 Download.c

```

#include "../include/download.h"

/**
 * Parse do URL usando regex
 */

```

```

int parseURL(const char *input, URL *url) {
    regex_t regex;

    // Verificar se tem barra
    regcomp(&regex, BAR, 0);
    if (regexec(&regex, input, 0, NULL, 0)) {
        fprintf(stderr, "Erro: URL inválido\n");
        return -1;
    }

    // Verificar se tem autenticação (@)
    regcomp(&regex, AT, 0);
    if (regexec(&regex, input, 0, NULL, 0) != 0) {
        // Formato: ftp://<host>/<url-path>
        sscanf(input, HOST_REGEX, url->host);
        strcpy(url->user, DEFAULT_USER);
        strcpy(url->password, DEFAULT_PASSWORD);
    } else {
        // Formato: ftp://[<user>:<password>@]<host>/<url-path>
        sscanf(input, HOST_AT_REGEX, url->host);
        sscanf(input, USER_REGEX, url->user);
        sscanf(input, PASS_REGEX, url->password);
    }

    // Extrair resource e filename
    // Encontrar o caminho após o host
    const char *pathStart = strstr(input, "://");
    if (pathStart != NULL) {
        pathStart += 3; // Pular "://"
        pathStart = strchr(pathStart, '/'); // Encontrar primeira barra após host
        if (pathStart != NULL) {
            strcpy(url->resource, pathStart); // Copiar o caminho completo (com a
barra inicial)
        } else {
            strcpy(url->resource, "/");
        }
    } else {
        strcpy(url->resource, "/");
    }

    // Extrair nome do ficheiro
    char *lastSlash = strrchr(url->resource, '/');
    if (lastSlash != NULL && strlen(lastSlash) > 1) {
        strcpy(url->file, lastSlash + 1);
    } else {
        strcpy(url->file, "downloaded_file");
    }
}

```

```

// Resolver hostname para IP
struct hostent *h;
if (strlen(url->host) == 0) {
    fprintf(stderr, "Erro: Host vazio\n");
    return -1;
}
if ((h = gethostbyname(url->host)) == NULL) {
    perror("gethostbyname()");
    return -1;
}

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

// Validar se todos os campos foram preenchidos
if (strlen(url->host) == 0 || strlen(url->user) == 0 ||
    strlen(url->password) == 0 || strlen(url->resource) == 0 ||
    strlen(url->file) == 0) {
    fprintf(stderr, "Erro: Parse incompleto\n");
    return -1;
}

return 0;
}

/**
* Cria socket e conecta ao servidor
*/
int createSocket(const char *ip, int port) {
    int sockfd;
    struct sockaddr_in server_addr;

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

    // Configurar endereço do servidor
    bzero((char *)&server_addr, sizeof(server_addr));
    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = inet_addr(ip);
    server_addr.sin_port = htons(port);

    // Conectar
    if (connect(sockfd, (struct sockaddr *)&server_addr, sizeof(server_addr)) < 0) {
        perror("connect()");
    }
}

```

```

        close(sockfd);
        return -1;
    }

    return sockfd;
}

/***
* Lê resposta do servidor usando máquina de estados
*/
int readResponse(const int socket, char *buffer) {
    char byte;
    int index = 0, responseCode = 0;
    ResponseState state = START;
    char code[4] = {0}; // Para armazenar os 3 dígitos do código
    int codeIndex = 0;

    memset(buffer, 0, MAX_LENGTH);

    while (state != END) {
        if (read(socket, &byte, 1) <= 0) {
            break;
        }

        switch (state) {
            case START:
                if (codeIndex < 3 && byte >= '0' && byte <= '9') {
                    // Ler os primeiros 3 dígitos do código
                    code[codeIndex++] = byte;
                } else if (byte == '+') {
                    state = SINGLE; // Resposta de linha única
                } else if (byte == '-') {
                    state = MULTIPLE; // Resposta multi-linha
                } else if (byte == '\n') {
                    state = END;
                }
                break;

            case SINGLE:
                if (byte == '\n') {
                    state = END;
                } else {
                    buffer[index++] = byte;
                }
                break;

            case MULTIPLE:

```

```

        if (byte == '\n') {
            // Nova linha em resposta multi-linha
            // Verificar se a próxima linha começa com o código seguido de
espaço
            char nextCode[4] = {0};
            int tempIndex = 0;
            char tempByte;

            // Ler próximos 3 caracteres
            for (int i = 0; i < 3; i++) {
                if (read(socket, &tempByte, 1) > 0) {
                    nextCode[tempIndex++] = tempByte;
                }
            }

            // Ler o 4º caractere (espaço ou hífen)
            if (read(socket, &tempByte, 1) > 0) {
                if (strcmp(code, nextCode) == 0 && tempByte == ' ') {
                    // Fim da resposta multi-linha
                    state = SINGLE;
                } else {
                    // Continuar lendo a linha
                    // (descartar o que foi lido)
                }
            }
        } else {
            // Ignorar o conteúdo da resposta multi-linha
        }
        break;
    }

    case END:
        break;

    default:
        break;
}
}

// Converter código para inteiro
if (strlen(code) == 3) {
    responseCode = atoi(code);
}

printf("< %d %s\n", responseCode, buffer);

return responseCode;
}

```

```

/**
 * Envia comando FTP (COM \r\n que é o correto!)
 */
int sendCommand(const int socket, const char *cmd, const char *arg) {
    char command[MAX_LENGTH];

    if (arg != NULL && strlen(arg) > 0) {
        sprintf(command, sizeof(command), "%s %s\r\n", cmd, arg);
    } else {
        sprintf(command, sizeof(command), "%s\r\n", cmd);
    }

    printf("> %s", command);

    if (write(socket, command, strlen(command)) < 0) {
        perror("write()");
        return -1;
    }

    return 0;
}

/**
 * Parse da resposta PASV
 */
int parsePasv(const char *response, char *ip, int *port) {
    int ip1, ip2, ip3, ip4, port1, port2;

    if (sscanf(response, PASSIVE_REGEX, &ip1, &ip2, &ip3, &ip4, &port1, &port2) != 6) {
        return -1;
    }

    sprintf(ip, "%d.%d.%d.%d", ip1, ip2, ip3, ip4);
    *port = port1 * 256 + port2;

    return 0;
}

/**
 * Autentica no servidor
 */
int ftpLogin(const int socket, const char *user, const char *password) {
    char buffer[MAX_LENGTH];
    int code;

```

```

// Enviar USER
if (sendCommand(socket, "USER", user) < 0) {
    return -1;
}

code = readResponse(socket, buffer);
if (code != SV_READY_FOR_PASS && code != SV_LOGIN_SUCCESS) {
    fprintf(stderr, "Erro: USER falhou (código %d)\n", code);
    return code;
}

// Se precisar de password
if (code == SV_READY_FOR_PASS) {
    if (sendCommand(socket, "PASS", password) < 0) {
        return -1;
    }

    code = readResponse(socket, buffer);
    if (code != SV_LOGIN_SUCCESS) {
        fprintf(stderr, "Erro: PASS falhou (código %d)\n", code);
        return code;
    }
}

return code;
}

/***
* Entra em modo passivo
*/
int ftpPassiveMode(const int socket, char *ip, int *port) {
    char buffer[MAX_LENGTH];

    if (sendCommand(socket, "PASV", NULL) < 0) {
        return -1;
    }

    int code = readResponse(socket, buffer);
    if (code != SV_PASSIVE_MODE) {
        fprintf(stderr, "Erro: PASV falhou (código %d)\n", code);
        return code;
    }

    if (parsePasv(buffer, ip, port) < 0) {
        fprintf(stderr, "Erro: Parse da resposta PASV falhou\n");
        return -1;
    }
}

```

```

        return code;
    }

/***
* Requisita ficheiro
*/
int ftpRequestFile(const int socket, const char *resource) {
    char buffer[MAX_LENGTH];

    if (sendCommand(socket, "RETR", resource) < 0) {
        return -1;
    }

    return readResponse(socket, buffer);
}

/***
* Descarrega ficheiro
*/
int ftpDownloadFile(const int control_socket, const int data_socket, const char
*filename) {
    char buffer[MAX_LENGTH];

    // Abrir ficheiro para escrita
    FILE *file = fopen(filename, "wb");
    if (file == NULL) {
        perror("fopen()");
        return -1;
    }

    // Receber dados
    printf("\nA transferir ficheiro '%s'...\n", filename);
    int total_bytes = 0;
    int bytes;

    while ((bytes = read(data_socket, buffer, MAX_LENGTH)) > 0) {
        if (fwrite(buffer, bytes, 1, file) < 0) {
            perror("fwrite()");
            fclose(file);
            return -1;
        }
        total_bytes += bytes;
        printf("\rRecebidos: %d bytes", total_bytes);
        fflush(stdout);
    }
}

```

```

printf("\n");
fclose(file);
close(data_socket); // Fechar socket de dados antes de ler resposta final

// Ler resposta final do servidor
char response[MAX_LENGTH];
int code = readResponse(control_socket, response);

printf("✓ Ficheiro '%s' transferido com sucesso! (%d bytes)\n", filename,
total_bytes);

return code;
}

/***
* Fecha conexão FTP
*/
int ftpQuit(const int control_socket) {
    char buffer[MAX_LENGTH];

    if (sendCommand(control_socket, "QUIT", NULL) < 0) {
        return -1;
    }

    int code = readResponse(control_socket, buffer);
    if (code != SV_GOODBYE) {
        fprintf(stderr, "Aviso: QUIT retornou código %d\n", code);
    }

    close(control_socket);

    return (code == SV_GOODBYE) ? 0 : -1;
}

/***
* Main
*/
int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "Uso: %s ftp://[<user>:<password>@]<host>/<url-path>\n",
argv[0]);
        fprintf(stderr, "Exemplo: %s
ftp://demo:password@test.rebex.net/readme.txt\n", argv[0]);
        return 1;
    }

    URL url;

```

```

memset(&url, 0, sizeof(url));

printf("==== Cliente FTP ====\n\n");

// 1. Parse do URL
printf("1. A fazer parse do URL...\n");
if (parseURL(argv[1], &url) != 0) {
    fprintf(stderr, "Erro no parse do URL\n");
    return 1;
}

printf("    Host: %s\n", url.host);
printf("    IP: %s\n", url.ip);
printf("    User: %s\n", url.user);
printf("    Resource: %s\n", url.resource);
printf("    File: %s\n\n", url.file);

// 2. Conectar ao servidor FTP (porta 21)

printf("2. A conectar ao servidor %s:%d...\n", url.ip, FTP_PORT);
int control_socket = createSocket(url.ip, FTP_PORT);
if (control_socket < 0) {
    fprintf(stderr, "Erro ao conectar ao servidor\n");
    return 1;
}
printf("✓ Conectado!\n\n");

// 3. Ler mensagem de boas-vindas
printf("3. A ler mensagem de boas-vindas...\n");
char buffer[MAX_LENGTH];
int code = readResponse(control_socket, buffer);
if (code != SV_READY_FOR_AUTH) {
    fprintf(stderr, "Erro: Mensagem de boas-vindas inválida (código %d)\n",
code);
    close(control_socket);
    return 1;
}
printf("\n");

// 4. Autenticar
printf("4. A autenticar...\n");
code = ftpLogin(control_socket, url.user, url.password);
if (code != SV_LOGIN_SUCCESS) {
    fprintf(stderr, "Erro na autenticação\n");
    close(control_socket);
    return 1;
}

```

```

printf("✓ Login com sucesso!\n\n");

// 5. Entrar em modo passivo
printf("5. A entrar em modo passivo...\n");
char data_ip[MAX_LENGTH];
int data_port;
code = ftpPassiveMode(control_socket, data_ip, &data_port);
if (code != SV_PASSIVE_MODE) {
    fprintf(stderr, "Erro no modo passivo\n");
    close(control_socket);
    return 1;
}
printf("✓ Modo passivo: %s:%d\n\n", data_ip, data_port);

// 6. Conectar ao socket de dados
printf("6. A conectar ao socket de dados...\n");
int data_socket = createSocket(data_ip, data_port);
if (data_socket < 0) {
    fprintf(stderr, "Erro ao conectar ao socket de dados\n");
    close(control_socket);
    return 1;
}
printf("✓ Conectado ao socket de dados!\n\n");

// 7. Requisitar ficheiro
printf("7. A requisitar ficheiro '%s'...\n", url.resource);
code = ftpRequestFile(control_socket, url.resource);
if (code != SV_READY_FOR_TRANSFER && code != SV_CONN_ALREADY_OPEN) {
    fprintf(stderr, "Erro ao requisitar ficheiro (código %d)\n", code);
    close(control_socket);
    close(data_socket);
    return 1;
}

// 8. Descarregar ficheiro
char output_path[MAX_LENGTH * 2];
snprintf(output_path, sizeof(output_path), "test_folder/%s", url.file);

code = ftpDownloadFile(control_socket, data_socket, output_path);
if (code != SV_TRANSFER_COMPLETE) {
    fprintf(stderr, "Aviso: Transferência pode estar incompleta (código %d)\n",
code);
}

// 9. Fechar conexão
printf("\n9. A fechar conexão...\n");
ftpQuit(control_socket);

```

```

    printf("\n==== Download concluído com sucesso! ====\n");

    return 0;
}

```

Anexo 2 - Experiências

2.1 Experiência 1

2.1.1

```

netedu@tux63:~$ ping statistics ---
143 packets transmitted, 143 received, 0% packet loss, time 145407ms
rtt min/avg/max/mdev = 0.193/0.223/0.244/0.011 ms
netedu@tux63:~$ route -n
bash: route: command not found
netedu@tux63:~$ /sbin/route -n
Kernel IP routing table
Destination      Gateway          Genmask        Flags Metric Ref  Use Iface
0.0.0.0          10.227.20.254  0.0.0.0        UG   100    0        0 if_mng
10.227.20.0     0.0.0.0        255.255.255.0  U     100    0        0 if_mng
172.16.60.0     0.0.0.0        255.255.255.0  U     0      0        0 if_e1
172.16.61.0     172.16.60.254  255.255.255.0  UG    0      0        0 if_e1
netedu@tux63:~$ /sbin/arp -a
_gateway (10.227.20.254) at e4:8d:8c:20:25:c8 [ether] on if_mng
? (172.16.60.254) at ec:75:0c:c2:17:49 [ether] on if_e1
fog.netlab.fe.up.pt (10.227.20.223) at bc:24:11:f8:0d:71 [ether] on if_mng
ns1.netlab.fe.up.pt (10.227.20.3) at bc:24:11:e7:5e:5b [ether] on if_mng
dns.netlab.fe.up.pt (10.227.20.2) at bc:24:11:f5:a2:54 [ether] on if_e1
netedu@tux63:~$ 

```

2.1.2

```

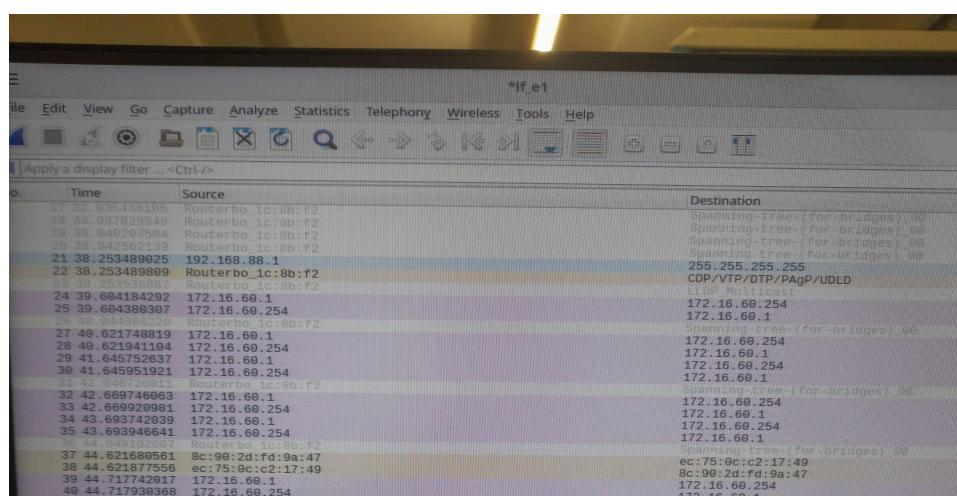
netedu@tux63:~$ ping 172.16.60.254
PING 172.16.60.254(172.16.60.254) 56(84) bytes from 172.16.60.254: icmp_seq=1 ttl=64 time=0.232 ms
64 bytes from 172.16.60.254: icmp_seq=140 ttl=64 time=0.227 ms
64 bytes from 172.16.60.254: icmp_seq=141 ttl=64 time=0.226 ms
64 bytes from 172.16.60.254: icmp_seq=142 ttl=64 time=0.234 ms
64 bytes from 172.16.60.254: icmp_seq=143 ttl=64 time=0.204 ms
^C
--- 172.16.60.254 ping statistics ---
143 packets transmitted, 143 received, 0% packet loss, time 145407ms
rtt min/avg/max/mdev = 0.193/0.223/0.244/0.011 ms
netedu@tux63:~$ route -n
bash: route: command not found
netedu@tux63:~$ /sbin/route -n
Kernel IP routing table
Destination      Gateway          Genmask        Flags Metric Ref  Use Iface
0.0.0.0          10.227.20.254  0.0.0.0        UG   100    0        0 if_mng
10.227.20.0     0.0.0.0        255.255.255.0  U     100    0        0 if_mng
172.16.60.0     0.0.0.0        255.255.255.0  U     0      0        0 if_e1
172.16.61.0     172.16.60.254  255.255.255.0  UG    0      0        0 if_e1

```

2.1.3

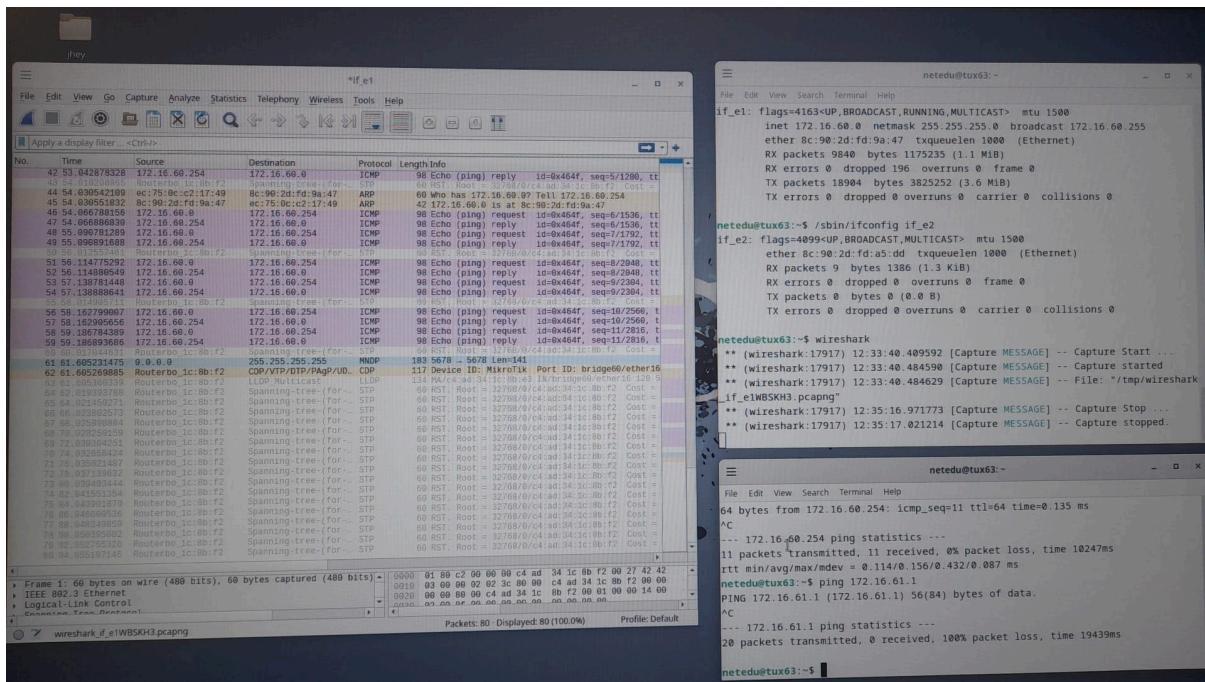
```
* ping from 172.16.60.1: icmp_seq=380 ttl=64 time=0.219 ms
64 bytes from 172.16.60.1: icmp_seq=381 ttl=64 time=0.206 ms
^C
--- 172.16.60.1 ping statistics ---
381 packets transmitted, 381 received, 0% packet loss, time 389088ms
rtt min/avg/max/mdev = 0.128/0.210/0.345/0.021 ms
netedu@tux64:~$ /sbin/route -n
Kernel IP routing table
Destination     Gateway         Genmask        Flags Metric Ref    Use Iface
0.0.0.0          10.227.20.254  0.0.0.0        UG    100    0        0 if_mng
10.227.20.0      0.0.0.0        255.255.255.0  U      100    0        0 if_mng
172.16.60.0      0.0.0.0        255.255.255.0  U      0      0        0 if_e1
172.16.61.0      0.0.0.0        255.255.255.0  U      0      0        0 if_e2
netedu@tux64:~$ /sbin/arp -a
_gateway (10.227.20.254) at e4:8d:8c:20:25:c8 [ether] on if_mng
dns.netlab.fe.up.pt (10.227.20.2) at bc:24:11:f5:a2:54 [ether] on if_mng
fog.netlab.fe.up.pt (10.227.20.223) at bc:24:11:f8:0d:71 [ether] on if_mng
? (172.16.60.1) at 8c:90:2d:fd:9a:47 [ether] on if_e1
ns1.netlab.fe.up.pt (10.227.20.3) at bc:24:11:e7:5e:5b [ether] on if_mng
? (172.16.61.1) at 8c:90:2d:fd:a6:09 [ether] on if_e2
netedu@tux64:~$
```

2.1.4



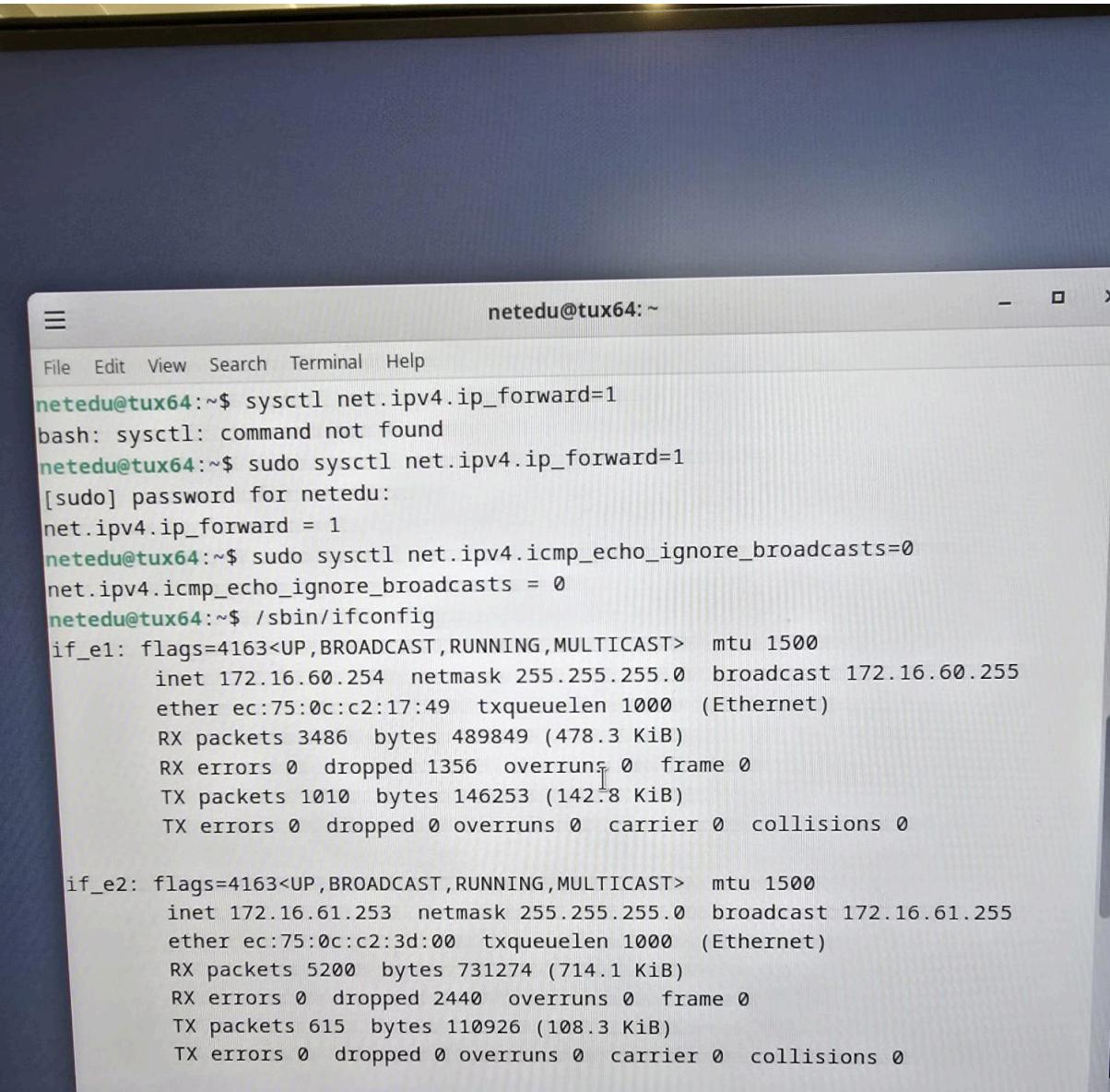
2.2 Experiência 2

2.2.1



2.3 Experiência 3

2.3.1



The screenshot shows a terminal window titled "netedu@tux64:~". The window has a dark blue header bar with the title and a light gray body. The terminal displays the following command-line session:

```
netedu@tux64:~$ sysctl net.ipv4.ip_forward=1
bash: sysctl: command not found
netedu@tux64:~$ sudo sysctl net.ipv4.ip_forward=1
[sudo] password for netedu:
net.ipv4.ip_forward = 1
netedu@tux64:~$ sudo sysctl net.ipv4.icmp_echo_ignore_broadcasts=0
net.ipv4.icmp_echo_ignore_broadcasts = 0
netedu@tux64:~$ /sbin/ifconfig
if_e1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        inet 172.16.60.254 netmask 255.255.255.0 broadcast 172.16.60.255
              ether ec:75:0c:c2:17:49 txqueuelen 1000  (Ethernet)
                    RX packets 3486 bytes 489849 (478.3 KiB)
                    RX errors 0 dropped 1356 overruns 0 frame 0
                    TX packets 1010 bytes 146253 (142.8 KiB)
                    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

if_e2: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        inet 172.16.61.253 netmask 255.255.255.0 broadcast 172.16.61.255
              ether ec:75:0c:c2:3d:00 txqueuelen 1000  (Ethernet)
                    RX packets 5200 bytes 731274 (714.1 KiB)
                    RX errors 0 dropped 2440 overruns 0 frame 0
                    TX packets 615 bytes 110926 (108.3 KiB)
                    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

2.3.2

```
netedu@tux63:~$ sudo /sbin/route add -net 172.16.61.0/24 gw 172.16.60.254
[sudo] password for netedu:
SIOCADDRT: File exists
netedu@tux63:~$ /sbin/route -n
Kernel IP routing table
Destination     Gateway         Genmask        Flags Metric Ref    Use Iface
0.0.0.0         10.227.20.254   0.0.0.0       UG    100    0        0 if_mng
10.227.20.0     0.0.0.0        255.255.255.0  U      100    0        0 if_mng
172.16.1.0       172.16.60.254   255.255.255.0  UG      0    0        0 if_e1
172.16.60.0     0.0.0.0        255.255.255.0  U      0    0        0 if_e1
172.16.61.0     172.16.60.254   255.255.255.0  UG      0    0        0 if_e1
netedu@tux63:~$
```

2.3.3

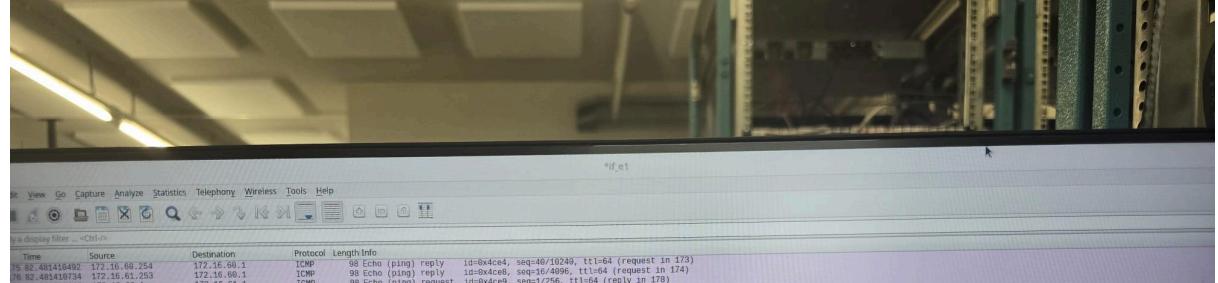
```
netedu@tux62:~  
File Edit View Search Terminal Help  
netedu@tux62:~$ /sbin/route add -net 172.16.60.0/24  
SIOCADDRT: Operation not permitted  
netedu@tux62:~$ sudo/sbin/route add -net 172.16.60.0/24  
bash: sudo/sbin/route: No such file or directory  
netedu@tux62:~$ sudo /sbin/route add -net 172.16.60.0/24  
[sudo] password for netedu:  
SIOCADDRT: No such device  
netedu@tux62:~$ sudo /sbin/route add -net 172.16.60.0/24 gw 172.16.60.254  
SIOCADDRT: Network is unreachable  
netedu@tux62:~$ sudo /sbin/route add -net 172.16.60.0/24 gw 172.16.60.253  
SIOCADDRT: Network is unreachable  
netedu@tux62:~$ sudo /sbin/route add -net 172.16.60.0/24 gw 172.16.61.253  
netedu@tux62:~$ /sbin/route -n  
Kernel IP routing table  


| Destination | Gateway       | Genmask       | Flags | Metric | Ref | Use | Iface  |
|-------------|---------------|---------------|-------|--------|-----|-----|--------|
| 0.0.0.0     | 10.227.20.254 | 0.0.0.0       | UG    | 100    | 0   | 0   | if_mng |
| 10.227.20.0 | 0.0.0.0       | 255.255.255.0 | U     | 100    | 0   | 0   | if_mng |
| 172.16.60.0 | 172.16.61.253 | 255.255.255.0 | UG    | 0      | 0   | 0   | if_e1  |
| 172.16.61.0 | 0.0.0.0       | 255.255.255.0 | U     | 0      | 0   | 0   | if_e1  |

  
netedu@tux62:~$
```

2.3.4

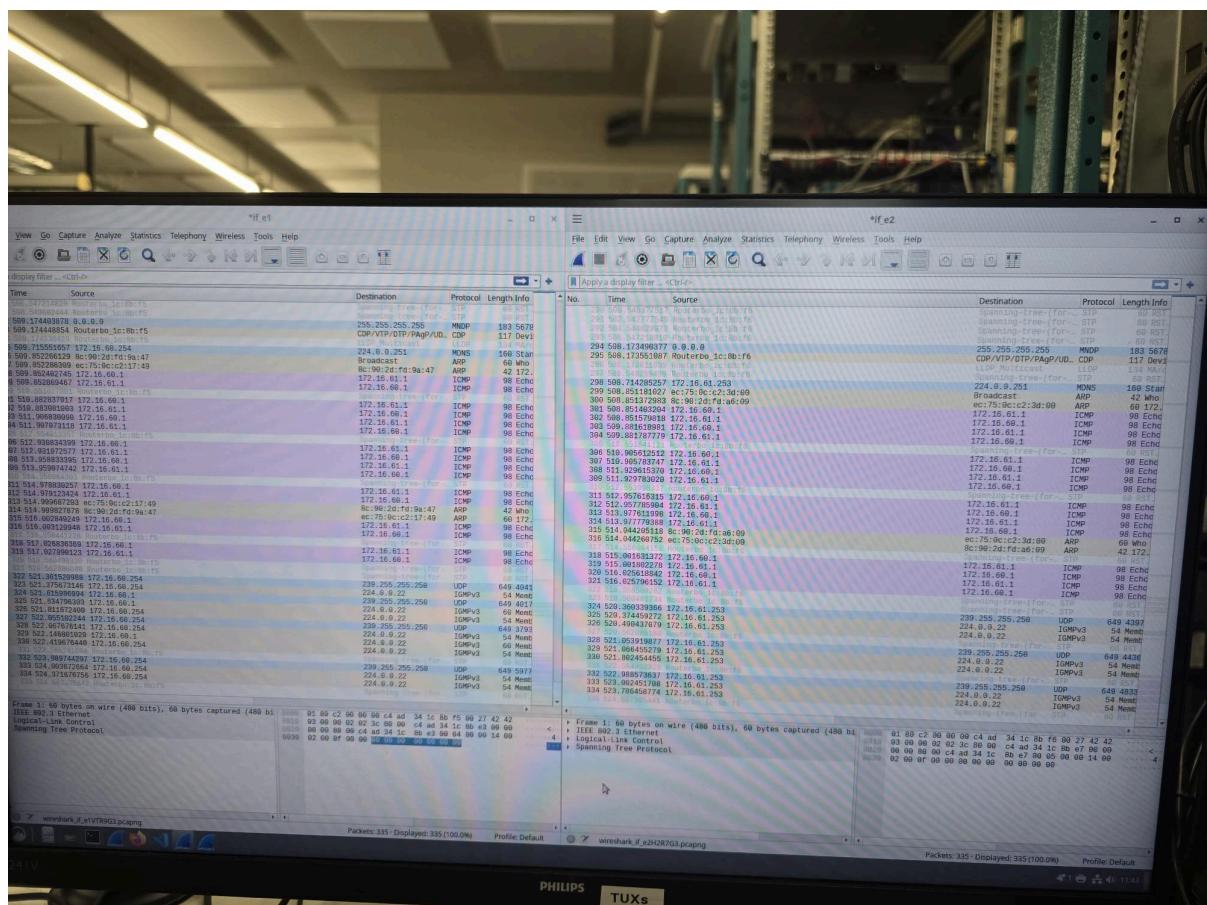
2.3.5



2.3.6

2.4 Experiência 4

2.4.1

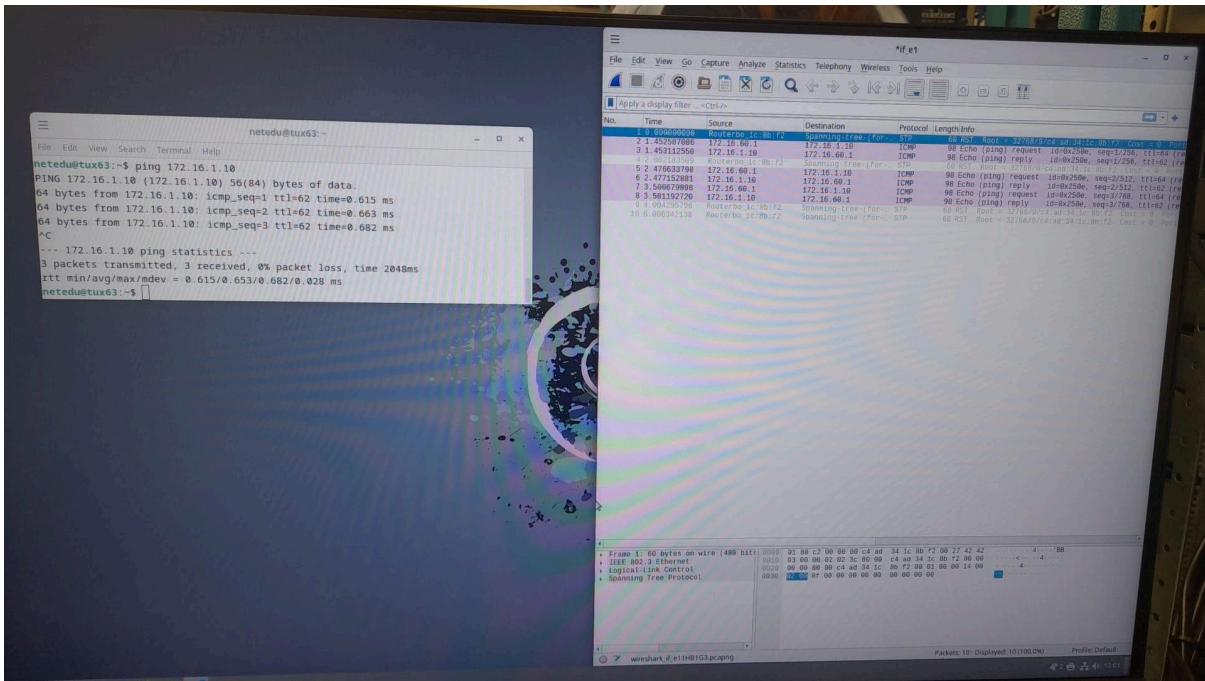


```
netedu@tux63:~
```

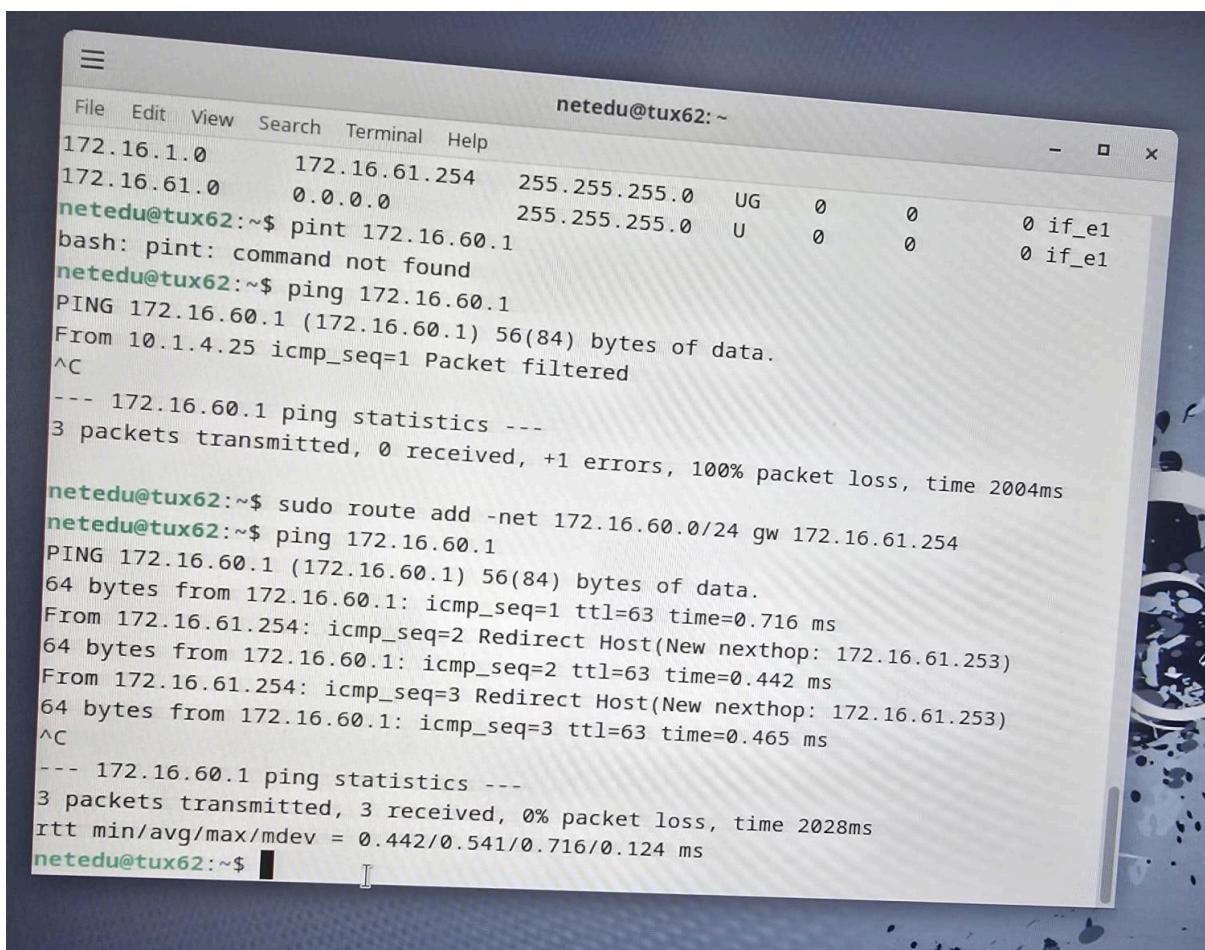
```
File Edit View Search Terminal Help
```

```
netedu@tux63:~$ ping 172.16.60.254
PING 172.16.60.254 (172.16.60.254) 56(84) bytes of data.
64 bytes from 172.16.60.254: icmp_seq=1 ttl=64 time=0.227 ms
64 bytes from 172.16.60.254: icmp_seq=2 ttl=64 time=0.230 ms
64 bytes from 172.16.60.254: icmp_seq=3 ttl=64 time=0.208 ms
^C
--- 172.16.60.254 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2043ms
rtt min/avg/max/mdev = 0.208/0.221/0.230/0.009 ms
netedu@tux63:~$ ping 172.16.61.1
PING 172.16.61.1 (172.16.61.1) 56(84) bytes of data.
64 bytes from 172.16.61.1: icmp_seq=1 ttl=63 time=0.678 ms
64 bytes from 172.16.61.1: icmp_seq=2 ttl=63 time=0.445 ms
^C
--- 172.16.61.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1009ms
rtt min/avg/max/mdev = 0.445/0.561/0.678/0.116 ms
netedu@tux63:~$ ping 172.16.61.254
PING 172.16.61.254 (172.16.61.254) 56(84) bytes of data.
64 bytes from 172.16.61.254: icmp_seq=1 ttl=63 time=7.13 ms
64 bytes from 172.16.61.254: icmp_seq=2 ttl=63 time=0.403 ms
64 bytes from 172.16.61.254: icmp_seq=3 ttl=63 time=0.392 ms
64 bytes from 172.16.61.254: icmp_seq=4 ttl=63 time=0.414 ms
64 bytes from 172.16.61.254: icmp_seq=5 ttl=63 time=0.408 ms
64 bytes from 172.16.61.254: icmp_seq=6 ttl=63 time=0.389 ms
64 bytes from 172.16.61.254: icmp_seq=7 ttl=63 time=0.421 ms
64 bytes from 172.16.61.254: icmp_seq=8 ttl=63 time=0.408 ms
64 bytes from 172.16.61.254: icmp_seq=9 ttl=63 time=0.647 ms
^C
--- 172.16.61.254 ping statistics ---
9 packets transmitted, 9 received, 0% packet loss, time 8148ms
rtt min/avg/max/mdev = 0.389/1.178/7.127/2.104 ms
netedu@tux63:~$
```

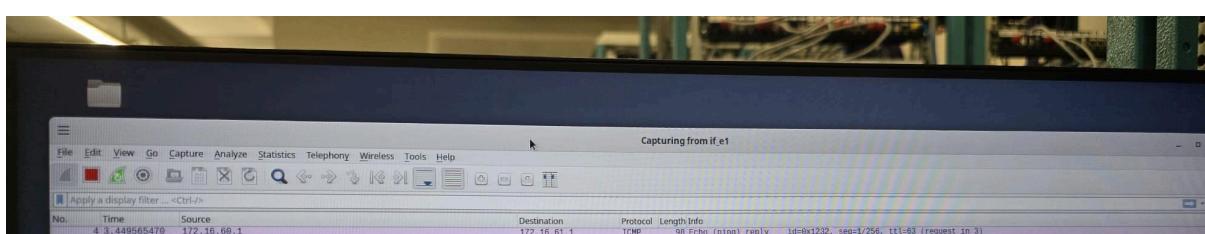
2.4.2



2.4.3

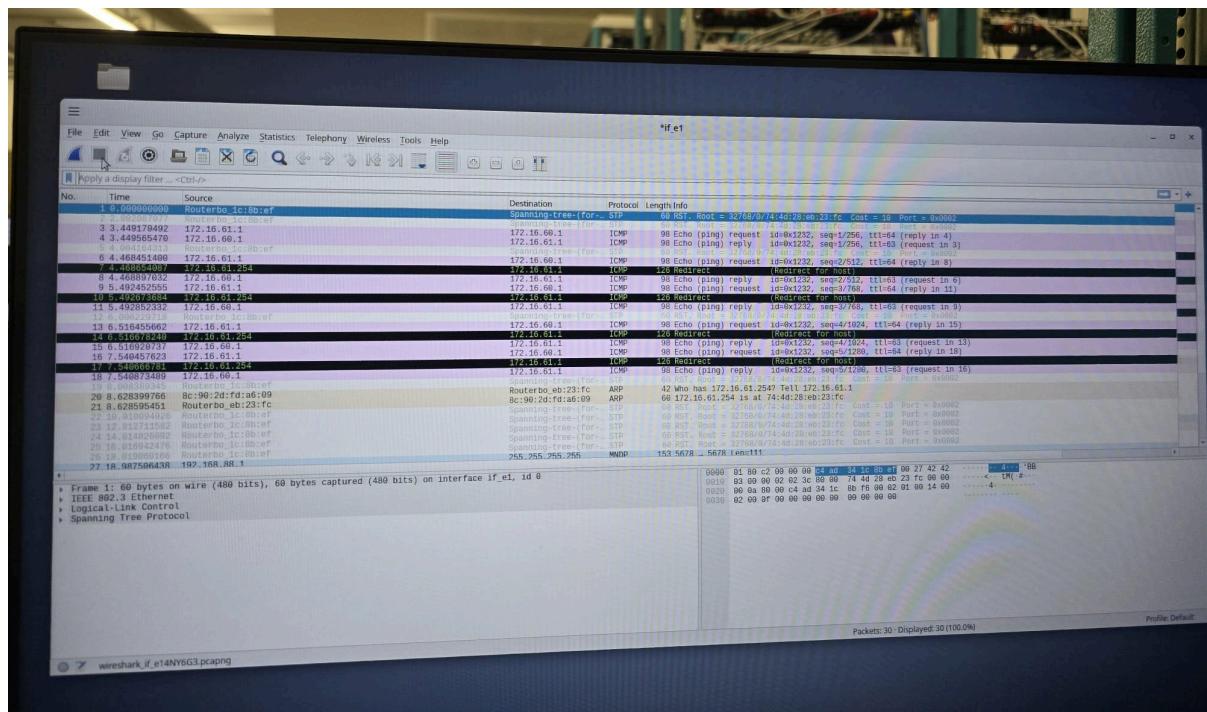


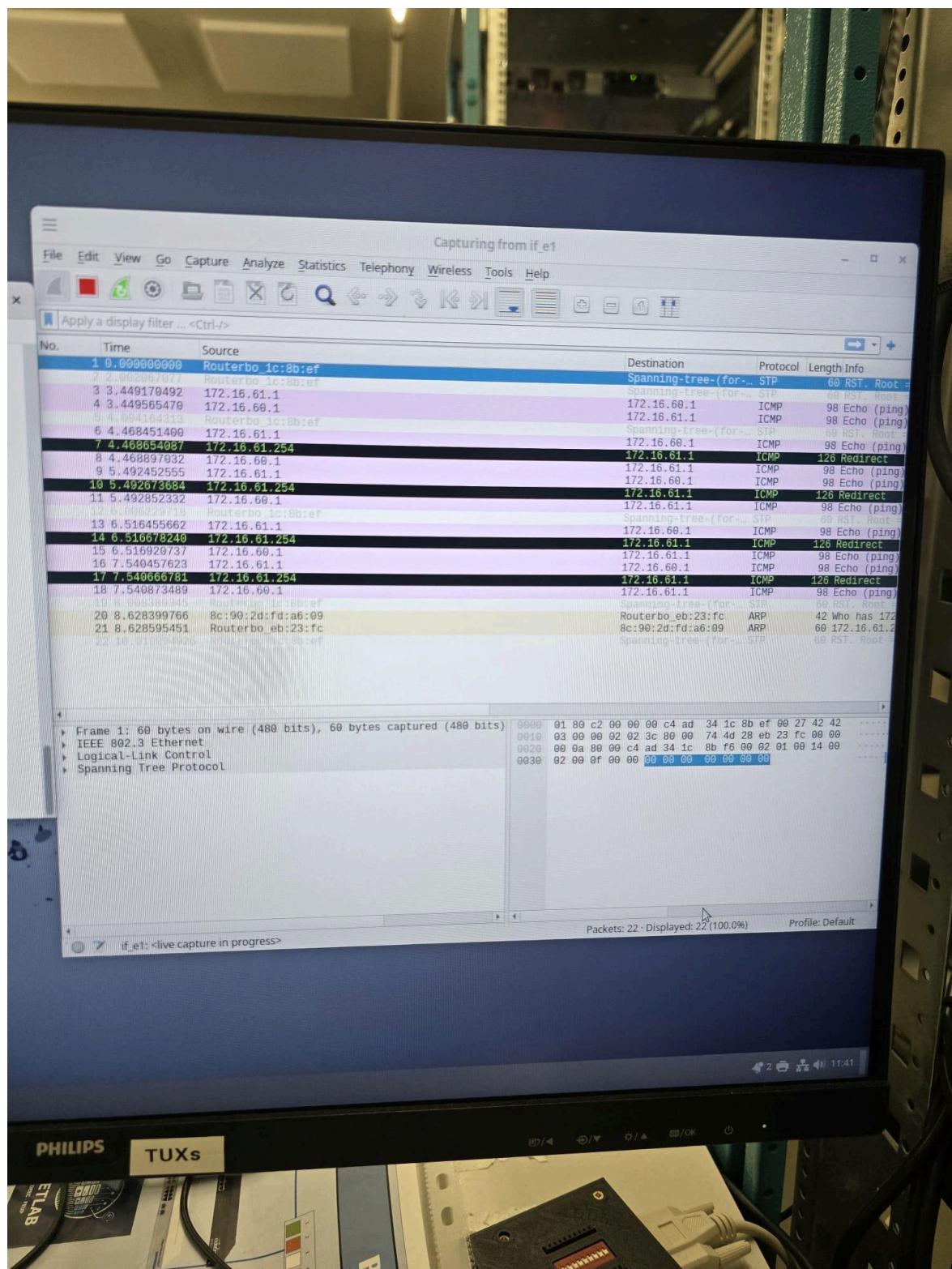
2.4.4



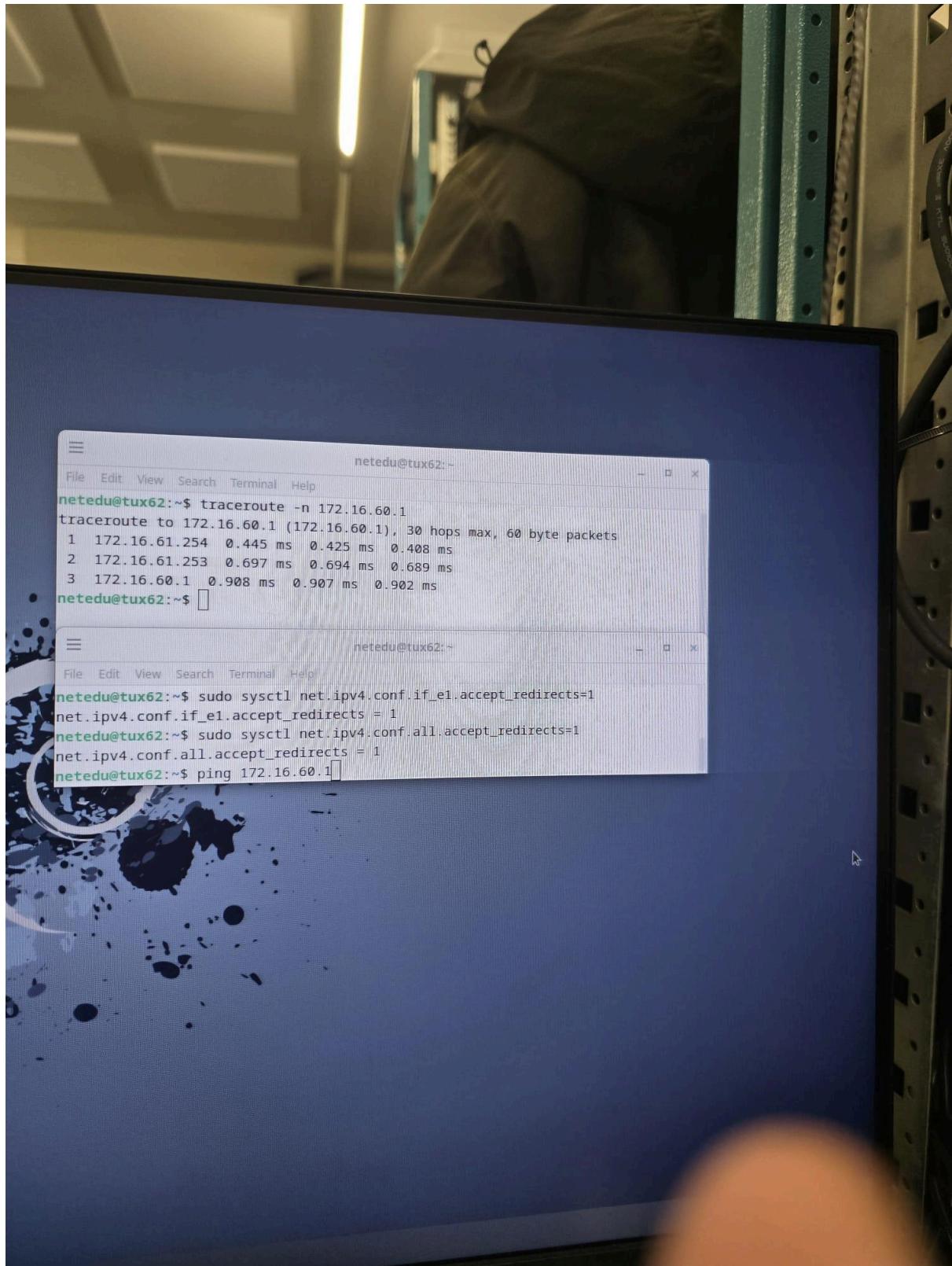
2.4.5

2.4.6





2.4.7

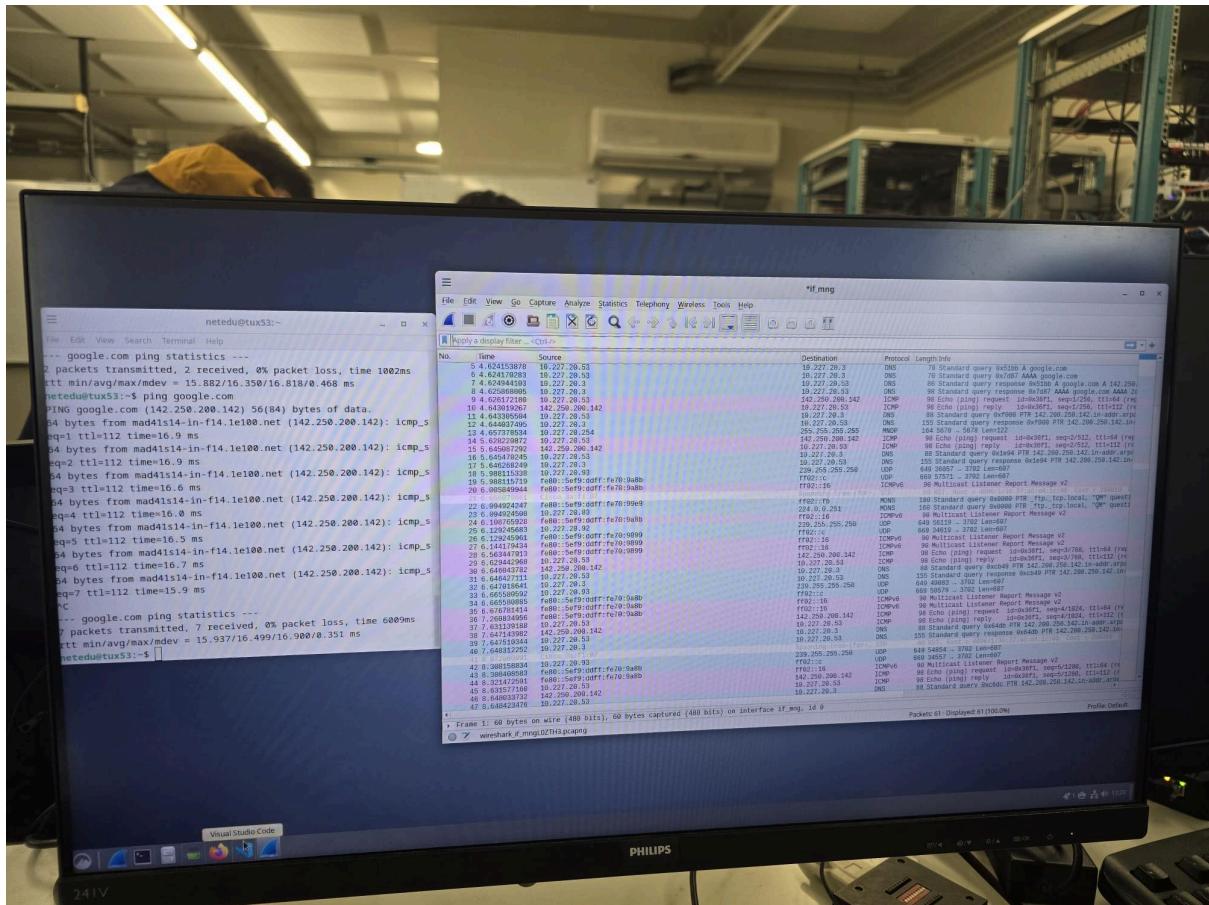


2.4.8

```
netedu@tux62:~$ traceroute -n 172.16.60.1
traceroute to 172.16.60.1 (172.16.60.1), 30 hops max, 60 byte packets
1  172.16.61.254  0.163 ms  0.168 ms  0.176 ms
2  172.16.61.253  0.379 ms  0.364 ms  0.384 ms
3  172.16.60.1  0.600 ms  0.602 ms  0.587 ms
netedu@tux62:~$ traceroute -n 172.16.60.1
traceroute to 172.16.60.1 (172.16.60.1), 30 hops max, 60 byte packets
1  172.16.61.253  0.236 ms  0.209 ms  0.194 ms
2  172.16.60.1  0.450 ms  0.429 ms  0.409 ms
netedu@tux62:~$
```

2.5 Experiência 5

2.5.1

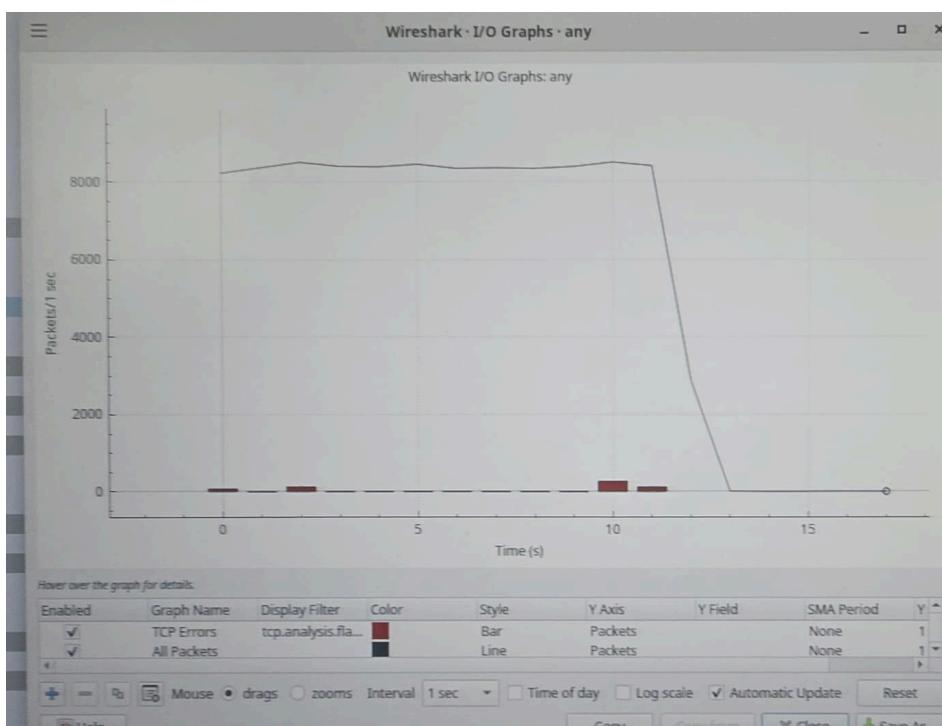


2.6 Experiencia 6

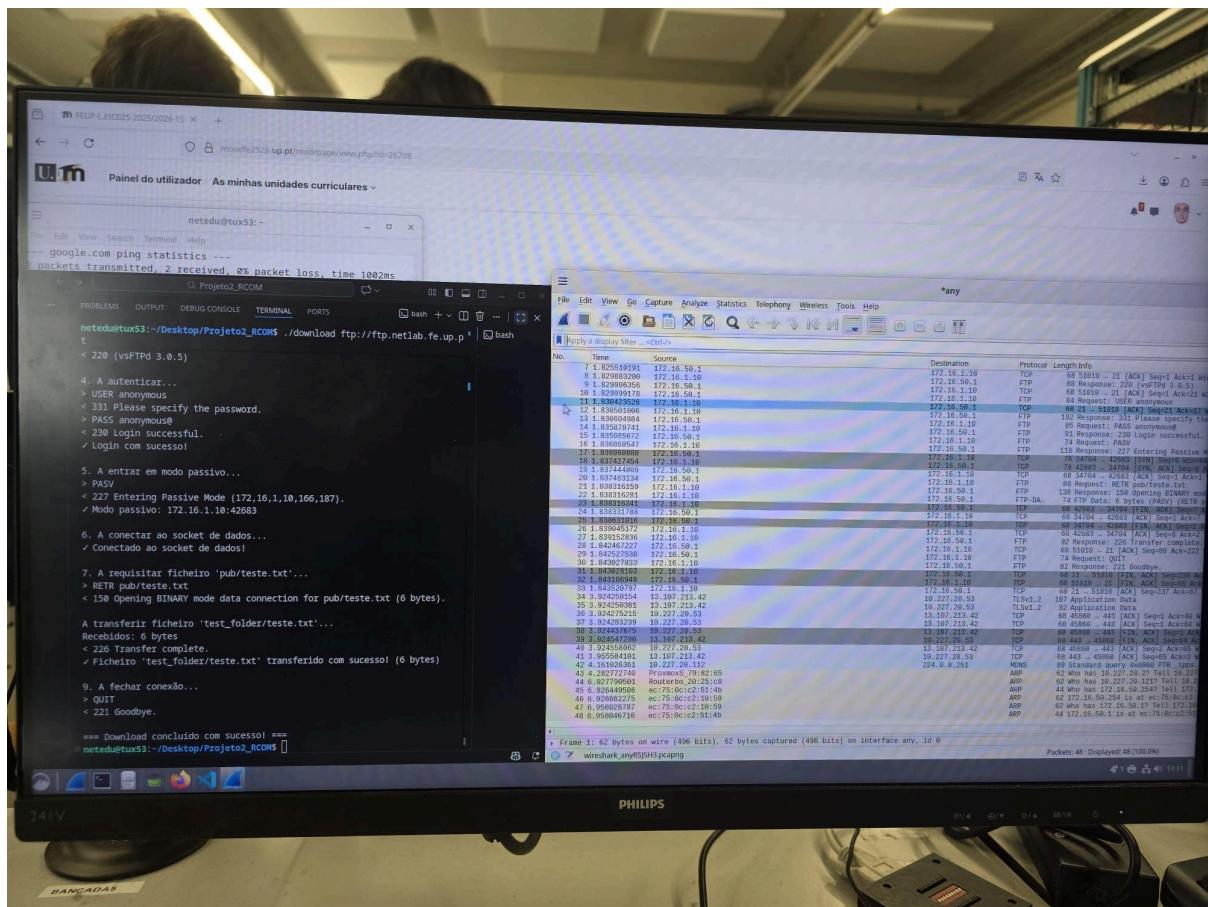
2.6.1



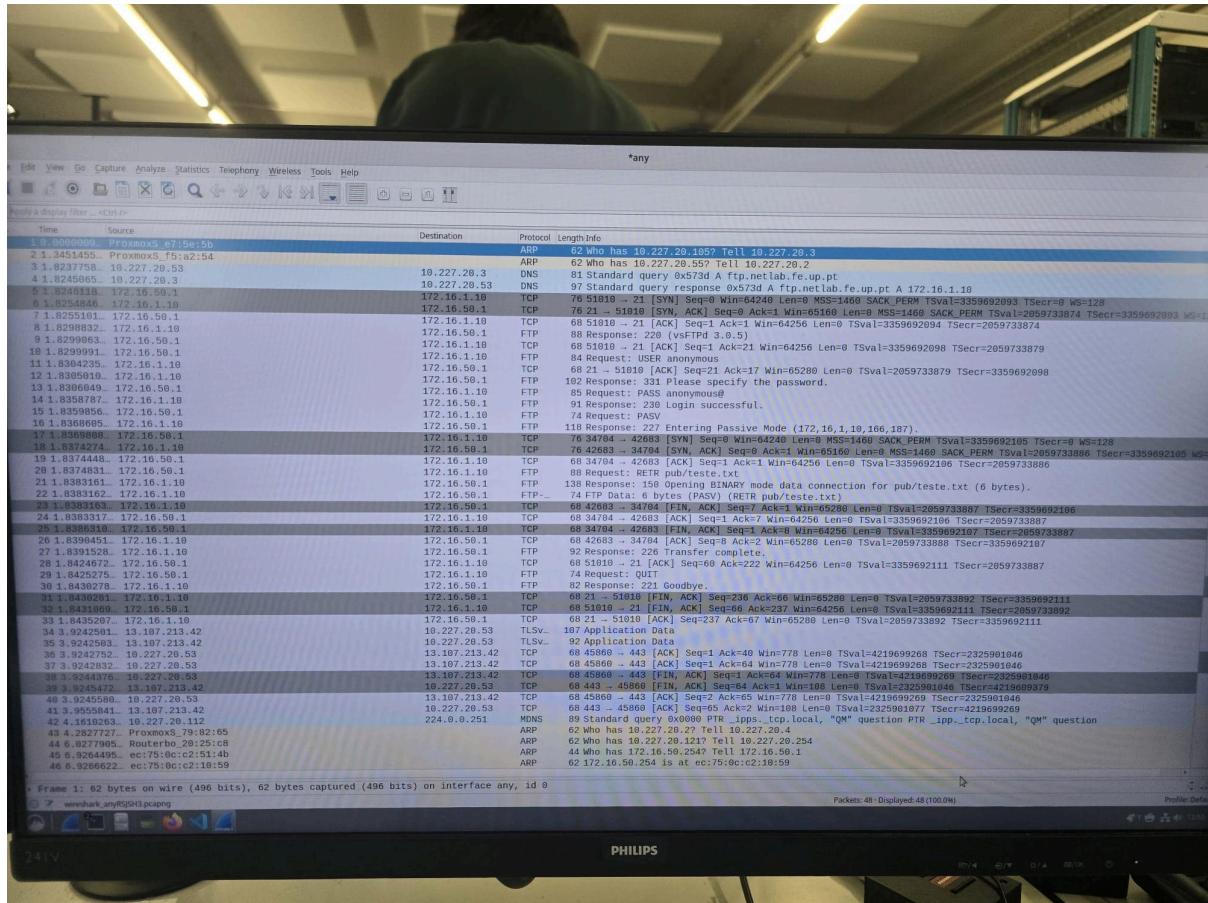
2.6.2



2.6.3.1



2.6.3.2



2.6.4

