U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Computer Networks

2º trabalho laboratorial

FEUP
Redes de Computadores

Daniel dos Santos Ferreira - up202108771
Mansur Mustafin - up202102355

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Sumário

Este projeto foi realizado no âmbito da unidade curricular Redes de Computadores, e centra-se na implementação de um programa de download através de FTP, bem como na configuração e utilização de uma rede informática.

Através deste projeto, aplicamos os conhecimentos teóricos adquiridos nas aulas teóricas para implementar um programa utilizando o protocolo RFC 959 e configurar uma rede.

Introdução

O objetivo deste projeto foi desenvolver e testar um programa de download utilizando FTP e configurar uma rede de computadores conforme descrito no guião. O objetivo era transferir um ficheiro da Internet utilizando a rede configurada. Este relatório está organizado nas seguintes secções:

1. Aplicação de Download
    1.1 Arquitetura
    1.2 Download report
2. Configuração e análise de rede
    2.1 Configurar uma rede IP
    2.2 Implementar duas pontes num switch
    2.3 Configurar um router em Linux
    2.4 Configure a Commercial Router and Implement NAT
    2.5 DNS
    2.6 TCP connections
3. Conclusões
4. Referências
5. Anexos

Parte 1 - Aplicação de download

Arquitetura

O programa download é um cliente FTP para descarregar ficheiros de servidores FTP. Ele segue o protocolo de transferência de ficheiros definido na RFC 959 e oferece suporte a URLs formatados de acordo com a RFC 1738.

A implementação da programa download do protocolo FTP foi realizada em 2 módulos:

1. `main.c`

   Função e Responsabilidade:
   - Verifica e analisa os argumentos fornecidos na linha de comando. Espera um URL formatado de acordo com a RFC 1738 para conexão FTP.
   - Configura os detalhes de conexão FTP usando `Settings`, como utilizador, senha, host, caminho do URL e endereço IP, com base no URL fornecido.
   - Orquestra as operações do programa, chamando funções para estabelecer a conexão FTP, fazer login, entrar em modo passivo, transferir ficheiros e encerrar a conexão.
   - Utiliza as funções implementadas em `download.c` para realizar as tarefas de conexão e transferência de arquivos.

2. `download.h` e `download.c`
   Definições e Estruturas e Implementação das Funções
   - Estrutura `struct Settings`: Armazena detalhes da conexão FTP, como o utilizador, senha, host, caminho de URL e endereço IP.
   - Inclui funções para análise de URL `parse_ftp_url()`, conexão de socket `establish_ftp_connection()`, leitura de resposta do servidor `read_ftp_response()`, envio de comandos FTP `send_ftp_command()`, login `login_ftp()`, entrada em modo passivo `enter_ftp_passive_mode()`, download de ficheiros `download_file()` e encerramento da conexão `clouse_connection()`.

O programa Download segue um fluxo de execução estruturado, garantindo uma interação eficiente e segura com servidores FTP. Este fluxo é caracterizado pelas seguintes etapas:

1. **Inicialização**:
   Verificação de argumentos da linha de comando configurando os detalhes de conexão FTP, como utilizador, senha, host e caminho do URL.
2. **Estabelecimento da Conexão FTP**:
   O programa estabelece uma conexão de controlo com o servidor FTP, utilizando o endereço IP e a porta especificados. Uma vez estabelecida a conexão, o programa realiza o processo de autenticação do utilizador, enviando as credenciais ao servidor.
3. **Transferência de Ficheiros**:
   O programa solicita ao servidor para entrar em modo passivo. Após receber as informações do servidor, estabelece uma segunda conexão (socket) para a transferência de dados. Utiliza a conexão de dados para descarregar o ficheiro solicitado, conforme o caminho especificado no URL.
4. **Terminação**:
   Encerramento das conexões: após a conclusão da transferência, o programa fecha as duas conexões estabelecidas.

Download report

Para correr o programa basta utilizar o `Makefile` e correr `make run`. O url pode ser alterado mantendo a estrutura `URL=ftp://[<user>:<password>@]<host>/<url-path>`
Ou na forma alternativa: dentro de pasta `/bin` correr `./download <url>`
Output da programa contém a estrutura de configuração e as respostas do servidor. Os logs de Wireshark contém os pacotes FTP iniciais com os comandos e FTP-DATA com frames de ficheiro.
O ficheiro de exemplo foi baixado sem erros e com sucesso.
O programa foi testado com vários ficheiros de diferentes servidores e tamanhos.

Part 2 - Configuração e análise de uma rede

Experiência 1 - Configurar uma rede IP

1.1 Arquitetura da rede

No final desta experiência a rede deverá consistir nos computadores tux3 e tux4 conectados pelo switch Mikrotik.

1.2 Objetivos da experiência

Esta experiência tinha como objetivos ensinar a configurar o IP de um computador, fazer dois computadores comunicar através de um switch e analisar as tabelas de forwarding e ARP.

1.3 Comandos de configuração

```
~# ifconfig eth0 up                        // tux3
~# ifconfig eth0 172.16.10.1/24            // tux3
~# ifconfig eth0                           // tux4
~# ifconfig eth0 172.16.10.254/24          // tux4
```

Ambos os computadores foram ligados a uma porta qualquer do switch, visto que a porta do switch escolhida para cada computador não era importante para a realização da experiência.

1.4 Logs relevantes

Nesta experiência configuramos os computadores tux3 e tux4 com os comandos descritos acima e corremos o comando `ping 172.16.10.254` para testar a conectividade entre os dois computadores. O comando `ping` gera pacotes ICMP que contêm os endereços IP das máquinas de origem e destino e os endereços MAC da origem e destino do próximo salto do pacote. Como os endereços MAC são necessários para que o pacote seja transferido do tux3 para o tux4 no comando `ping` e o tux3 não sabe o endereço MAC do tux4, este envia um pacote de pedido ARP (usado para associar um endereço MAC ao respetivo endereço IP) em modo broadcast para que o computador tux4 lhe comunique o seu endereço MAC. Este pacote tem como endereços MAC de origem e destino o endereço MAC do tux3 (00:21:5a:5a:7d:16) e o endereço MAC broadcast (ff:ff:ff:ff:ff:ff para que consiga tomar qualquer salto da tabela de forwarding) respetivamente. Quando o tux4 recebe este pacote responde com um novo pacote em que os endereços MAC de origem e destino são o endereço MAC do tux4 (00:c0:df:25:13:65) e o endereço MAC do tux3 respetivamente. Após isso, este resultado é guardado na tabela ARP do tux3 pelo que o comando `ping` pode ser executado várias vezes sem ter de descobrir o endereço MAC do tux4 a cada novo `ping` como se pode verificar no seguinte log.
Uma máquina consegue diferenciar o tipo de trama de Ethernet que recebe através do parâmetro EtherType no cabeçalho da trama verificando se o pacote é ARP ou IP. Se o campo tiver o valor 0x0806 é uma trama ARP, se não, se tiver o valor 0x0800 é uma trama IP. Em tramas IP podemos também verificar o campo “Protocol” no seu cabeçalho. Se o valor for igual a 1, então a trama encapsulada na trama IP é uma trama ICMP.
Verificamos também que cada uma das máquinas continha uma interface loopback responsável por deixar uma máquina comunicar consigo mesma. Esta interface é útil para encontrar problemas numa rede e usar serviços hospedados na própria máquina.

Experiência 2 - Implementar duas pontes num switch

2.1 Arquitetura da rede

No final da experiência a rede deverá consistir no tux3 e tux4 conectados a uma mesma ponte chamada “bridge10” e o tux2 configurado e conectado a uma ponte chamada “bridge11”, usando o switch Mikrotik.

2.2 Objetivos da experiência

Esta experiência tinha como objetivos ensinar a construir uma ponte através da consola do switch Mikrotik, conectar as máquinas às pontes criadas e verificar que não existe qualquer forma de comunicação entre máquinas conectadas a pontes diferentes (sub-redes diferentes) por predefinição.

2.3 Comandos de configuração

Vamos assumir que o computadores tux2, tux3 e tux4 estão conectados, respetivamente, às portas 1, 2 e 3 do switch Mikrotik.

```
~# ifconfig eth0 up                                    // tux2
~# ifconfig eth0 172.16.11.1/24                        // tux2
```

Os seguintes comandos são usados na consola do Mikrotik:

```
> /interface bridge port remove [find interface = ether1]
> /interface bridge port remove [find interface = ether2]
> /interface bridge port remove [find interface = ether3]
> /interface bridge port add bridge=bridge11 interface=ether1
> /interface bridge port add bridge=bridge10 interface=ether2
> /interface bridge port add bridge=bridge10 interface=ether3
```

2.4 Logs relevantes

Após a configuração descrita acima, procedemos a testar lançar um ping broadcast (`ping -b 172.16.10.255`) para aferir quais máquinas se encontravam conectadas ao computador tux3. A partir deste log conseguimos concluir que o tux3 e tux4 se encontram conectados, pois estão ligados a uma mesma ponte no switch, a bridge10 (criando uma sub-rede). No log aparece que não foi encontrada nenhuma resposta pois o parâmetro net.ipv4.icmp_echo_ignore_broadcasts está ativo por predefinição, fazendo com que as máquinas não respondam a pings broadcast. No entanto, verifica-se embaixo que o pacote é recebido pelo tux4 com o endereço 172.16.10.254.
Em contraste, o tux2, que não está conectado à bridge10 mas sim isolado na bridge11, ou seja, noutra sub-rede, não recebe os pacotes ICMP provenientes do ping broadcast executado pelo tux3 como pode ser consultado no seguinte log vazio.
Com estes resultados podemos concluir que existem dois domínios de broadcast, um para cada bridge que foi criada, pois somente o tux4 recebeu os pacotes do ping em broadcast do tux3.

Experiência 3 - Configurar um router em Linux

3.1 Arquitetura da rede

No final da experiência a rede deverá consistir no tux3 e tux4 conectados a uma mesma ponte chamada “bridge10” e o tux2 e o tux4 conectados a uma mesma ponte chamada “bridge11” através do switch Mikrotik, sendo que tux4 age como um router na rede final.

3.2 Objetivos da experiência

Esta experiência tinha como objetivos transformar o tux4 num router, aprender a configurar uma rota entre duas máquinas para verificar a conectividade entre máquinas em sub-redes diferentes, analisar os endereços de IP e MAC nos pacotes ARP e ICMP, e analisar entradas da tabela de forwarding.

3.3 Comandos de configuração

Vamos considerar que ligamos o tux4 a uma porta adicional no switch Mikrotik, sendo esta a porta 4.

```
~# ifconfig eth1 up                                    // tux4
~# ifconfig eth1 172.16.11.253/24                      // tux4
> /interface bridge port remove [find interface=ether4]     // consola
> /interface bridge port add bridge=11 interface=ether4     // consola
~# sysctl net.ipv4.ip_forward=1                        // tux4
~# sysctl net.ipv4.icmp_echo_ignore_broadcasts=0       // tux4
~# route add -net 172.16.11.0/24 gw 172.16.10.254      // tux3
```

```
~# route add -net 172.16.10.0/24 gw 172.16.11.253          // tux2
```

3.4 Logs relevantes

Após a configuração descrita acima conseguimos garantir a conexão a todas as outras interfaces de rede: `172.16.10.254`, `172.16.11.253` e `172.16.11.1`. Assim, concluímos que configuramos bem as rotas acima, como pode ser verificado nos seguintes logs: para 172.16.10.254, para 172.16.11.253, para 172.16.11.1. Estes foram capturados com o comando `ping` do tux3 (`172.16.10.1`)
Após ter verificado esta conexão procedemos a eliminar as entradas da tabela ARP de cada um dos tuxs e verificamos a conexão a partir de um ping do tux3 ao tux2. A única diferença que se assistiu ao passo anterior foi o facto de terem de ser efetuados pedidos ARP para tomar conhecimento dos endereços MAC entre os tuxs para que os pacotes ICMP do comando `ping` pudessem ser transmitidos, verificando-se na mesma que a conexão entre os tuxs ainda se estabelece como se pode concluir pelos seguintes logs: eth0, eth1. Nestas pacotes ARP estão associados os endereços MAC de origem e destino dentro de um certo salto. Por exemplo, no salto do tux3 para o tux4 os endereços MAC são, respetivamente, o endereço MAC do tux3 e o endereço MAC do tux4, apesar de o destino final ser o tux2. Isto porque as mensagens ARP estão ao nível da camada de dados, ou seja, ao nível da ligação direta entre as máquinas. Além disso, verifica-se o envio de dois pacotes ICMP por cada pedido do ping, sendo um deles o pedido e o outro a resposta. Cada um destes pacotes guarda sempre os mesmo endereços IP, sendo estes os endereços de IP do tux3 e do tux2 (o do tux3 como origem e o do tux2 como destino no pedido e vice-versa na resposta). Os endereços MAC variam, como descritos nos pacotes ARP, de salto em salto.
Analisamos também a tabela de forwarding verificando que tanto o tux3 como o tux4 têm uma route em que o destino é qualquer máquina que esteja na subnet 172.16.20.0/24 como pode ser visto na primeira entrada do comando `route -n` no tux3 e no tux4. Além disso, tanto o tux2 como o tux4 têm uma route em que o destino é qualquer máquina que esteja na subnet 172.16.21.0/24 como pode ser visto na segunda entrada do comando `route -n` no tux2 e no tux4. Por fim, o tux3 tem uma route cujo destino é o tux2 em que o próximo *hop* é o tux4 e o tux2 tem também uma route cujo destino é o tux3 em que o próximo *hop* é também o tux4.
Por fim, verificamos que cada entrada da tabela de forwarding contém as seguintes informações: ip de destino do pacote, próximo salto (gateway), uma máscara (genmask) usada para verificar se um certo pacote pertence a uma certa subnet para ver se o ip que um pacote quer alcançar é o mesmo que o ip de destino numa entrada da tabela de forwarding; uma interface a ser usada para o próximo salto, um campo chamado *metric* associado ao custo de enviar um pacote no próximo salto, os campos *ref* e *use* que determinam o número de referências da entrada e número de pacotes enviados a partir da entrada correspondente e um conjunto de flags que determinam o tipo e estado da rota em questão.

Experiência 4 - Configurar um router comercial e implementar NAT

4.1 Arquitetura da rede

No final da experiência a rede deverá consistir no tux3 e tux4 conectados a uma mesma ponte chamada “bridge10”, o tux2, tux4 e o router Mikrotik conectados a uma outra ponte chamada “bridge11” pelo switch Mikrotik, e o router Mikrotik conectado à Internet com NAT configurado. O router Mikrotik deverá também ser o router por defeito do tux2 e tux4, continuando o tux4 a ser o router por defeito do tux3.

4.2 Objetivos da experiência

Esta experiência tinha como objetivos aprender a usar o router Mikrotik configurando endereços de IP e rotas, perceber como NAT funciona e conectar a nossa rede à Internet.

4.3 Comandos de configuração

Vamos considerar que o router Mikrotik está ligado ao switch Mikrotik na porta 5.

```
> /interface bridge remove port [find interface=ether5]           // switch
> /interface bridge add port bridge=bridge11 interface=ether5     // switch
> /ip address add address=172.16.2.19/24 interface=ether1         // router
> /ip address add address=172.16.11.254/24 interface=ether2       // router
> /ip route add dst-address=172.16.10.0/24 gateway=172.16.11.253  // router
> /ip route add dst-address=0.0.0.0/0 gateway=172.15.2.254        // router
~# route del -net 172.16.10.0 netmask 255.255.255.0 gw 172.16.11.253 //tux2
~# route add default gw 172.16.11.254                             // tux2
~# route add default gw 172.16.11.254                             // tux4
~# route add default gw 172.16.10.254                             // tux3
> /ip firewall nat disable 0        // desativar NAT no router
> /ip firewall nat enable 0         // ativar NAT no router
```

4.4 Logs relevantes

Em primeiro lugar, depois de efetuar a configuração descrita acima verificamos com sucesso que o tux3 conseguia chegar a todas as interfaces de rede pedidas: tux2, tux4, router Mikrotik e o endereço IP 172.16.2.254 para aceder à Internet (note-se que o NAT estava ligado neste parte da experiência).
Após isso, removemos a route que ligava o tux2 à sub-rede da ponte "bridge10" através do tux4, removemos os redirecionamentos ICMP, e efetuamos um ping ao tux3 através do tux2. Observámos que os pacotes ICMP seguem a seguinte rota: tux2 -> router Mikrotik -> tux4 -> tux3 -> tux4 -> tux2. Ou seja, não existe redirecionamento do tux2 diretamente para o tux4, que seria um caminho mais rápido. No entanto, com os redirecionamentos ICMP ativados, assistimos a uma mudança de rota, pois os pacotes partem diretamente do tux2 para o tux4. Isto pode ser observado usando o comando `traceroute` sem redirecionamentos ICMP e com redirecionamentos ICMP.
Por fim, tentamos efetuar um ping desde o tux3 para o router dos labs para testar a conexão à Internet. Testamos a conectividade com e sem NAT ativado. NAT (network address translation) é um mecanismo que serve para traduzir um conjuntos de endereços IPs de máquinas numa certa rede privada para um único IP público que pode ser utilizado fora da rede privada, por exemplo, na Internet, sendo assim possível reduzir o número de endereços públicos na Internet sem que haja colisões de endereços na mesma rede. Ao dar ping desde o tux3 para o endereço 172.16.2.254 concluímos que é estabelecida conexão à Internet com NAT ativado e não existe com NAT desativado.

Experiência 5 - DNS

5.1 Arquitetura da rede

A arquitetura deverá manter-se exatamente na mesma configuração que na experiência anterior, sendo simplesmente necessário configurar DNS no tux2, tux3 e tux4 editando o ficheiro /etc/resolv.conf em cada uma das máquinas.

5.2 Objetivos da experiência

Esta experiência tinha como objetivos perceber como funciona o serviço DNS e tentar dar ping a algum hostname aproveitando a configuração à Internet existente e o serviço DNS.

5.3 Comandos de configuração

```
~# echo 'nameserver 172.16.2.1' > /etc/resolv.conf    // tux2
```

```
~# echo 'nameserver 172.16.2.1' > /etc/resolv.conf    // tux3
~# echo 'nameserver 172.16.2.1' > /etc/resolv.conf    // tux4
```

5.4 Logs relevantes

Para testar a configuração realizada acima decidimos tentar dar ping ao google.com com o auxílio do serviço de DNS. O serviço de DNS (Domain Name System) é encarregue de traduzir nomes reconhecíveis pelo ser humanos como google.com ou amazon.com para endereços de IP concretos a serem usados na camada de rede. Como o tux3 não reconhece um endereço de IP chamado "google.com" ele realiza uma query DNS para tentar descobrir qual o endereço IP associado ao hostname "google.com". Este pedido é feito à lista de nameservers no ficheiros /etc/resolv.conf, que contém, neste caso, o endereço 172.16.2.1. Quando a resposta é obtida é enviado um DNS Query Response com o endereço IP do hostname solicitado. Assim, os pacotes permutados pelo DNS são pacotes DNS Query que fazem os pedidos de um endereço IP dado um hostname e os pacotes DNS Query Response que devolvem esse resultado. Os dois tipos de pacotes partilham uma grande quantidade de informação como os parâmetros "Questions", "Answer RRs", "Authority RRs", "Additional RRs" um conjunto de flags e o parâmetro "Queries" que define realmente o pedido DNS que contém o hostname cujo qual queremos saber o endereço de IP, o tamanho do hostname, entre outras. Os pacotes DNS Query Response têm um parâmetro a mais chamado "Answers" no qual está definida a resposta dada à query que recebeu, contendo o endereço IP do hostname fornecido, o próprio hostname, o tamanho dos dados, o tempo que o pacote tem até ser descartado (Time to Live), entre outros. Ao realizar o ping confirmamos a existência destes dois tipos de pacotes e dos parâmetros associados (DNS Query e DNS Query Response), sendo o ping realizado com sucesso após o pedido DNS que determinou o endereço IP de google.com 142.250.184.174.

Experiência 6 - TCP connections

6.1 Arquitetura da rede

A arquitetura deverá manter-se exatamente na mesma configuração que na experiência anterior, visto que o objetivo desta experiência é testar a aplicação desenvolvida na parte 1.

6.2 Objetivos da experiência

O objetivo desta experiência é integrar a aplicação desenvolvida na primeira parte do trabalho que descarrega um ficheiro através do protocolo FTP usando a rede que configuramos nesta segunda parte.

6.3 Logs relevantes

Assim como descrito no guião realizamos o download de um ficheiro do servidor FTP, analisando depois os logs que obtivemos.
Em primeiro lugar, observamos que foram abertas duas conexões TCP pela nossa aplicação pela troca dos pacotes [SYN], [SYN, ACK] e [ACK] duas vezes no início da aplicação entre as máquinas com endereços de IP 172.16.10.1 (o tux3) e 192.168.109.136 (o servidor do qual descarregamos o ficheiro), que marcam o estabelecimento da conexão entre as duas máquinas. Podemos observar esta fase de estabelecimento da conexão duas vezes no seguinte log.
Conseguimos também confirmar que a primeira conexão que é iniciada e configurada em modo passivo (que está associada à porta 47298 no tux3 e à porta 21 no servidor) é a responsável por transportar o controlo de informação pois é a que contém os pacotes TCP com comandos direcionados para o servidor e as respostas dadas pelo servidor a esses comandos. A outra conexão (associada à porta 51578 no tux3 e 43329 no servidor) é responsável por descarregar o ficheiro.

Conseguimos também visualizar as três fases da conexão TCP, o estabelecimento da conexão apresentada acima, a transferência de dados que pode ser vista neste log e o fecho da conexão com a troca de pacotes [FIN, ACK] entre as duas máquinas como pode ser visto neste log.
Verificámos também os parâmetros usados para o mecanismo ARQ que garante que não existe perda de informação, para garantir uma transferência confiável de dados, retransmitindo pacotes perdidos ou repetidos. Os parâmetros indispensáveis para este mecanismo são o número de sequência que identificam um certo pacote TCP e o número de acknowledgement que comunica à outra máquina o número do próximo pacote que pretende receber. Estes dois valores em adição com o tamanho da janela permitem o controlo de quais pacotes foram enviados e quais foram recebidos para que possa existir uma retransmissão caso algum erro ocorra. Podemos observar a maneira como variam esses valores em três pacotes TCP consecutivos em log1, log2 e log3. Como esperado, o primeiro pacote tem o número de acknowledgement igual ao número de sequência do segundo pacote (pois o segundo pacote é uma resposta ao primeiro pacote) e o segundo pacote tem o número de acknowledgment igual ao número de sequência do terceiro pacote.
Conseguimos também consultar a maneira como o processo de congestionamento funciona. Em primeiro lugar, a janela usada para a transmissão dos pacotes é pequena e vai aumentando gradualmente enquanto não ocorre nenhum erro (additive increase) aumentando cada vez mais o throughput da aplicação. Quando ocorre um erro pela perda ou corrupção de um pacote o tamanho da janela é diminuída à metade (multiplicative decrease) para diminuir o tráfego e evitar a perda de mais pacotes, assistindo-se a um comportamento do gráfico de throughput em forma de saw-tooth como pode ser visto no seguinte gráfico.
Por fim, realizamos a transferência do mesmo ficheiro no tux3 e tux2 em simultâneo verificando que as conexões são perturbadas pela segunda transferência no tux2. Isto porque o throughput do servidor continua a ser o mesmo mas dividido por duas conexões diferentes, assistindo-se à diminuição do throughput em ambas as conexões. Nos nossos logs assistimos a uma taxa de erros no envio de pacotes TCP duplicados que podem ser observados neste log e no gráfico associado.

Conclusões

Neste projeto, desenvolvemos com sucesso uma aplicação de download de FTP e configuramos uma rede, obtendo conhecimentos práticos sobre protocolos de rede e configuração de dispositivos. Para tal implementamos um cliente FTP robusto, tivemos experiências práticas com redes IP, bridges, Linux e routers comerciais, análise de pacotes usando Wireshark, compreendemos os sistemas NAT e DNS e ganhamos uma compreensão geral da camada de rede.

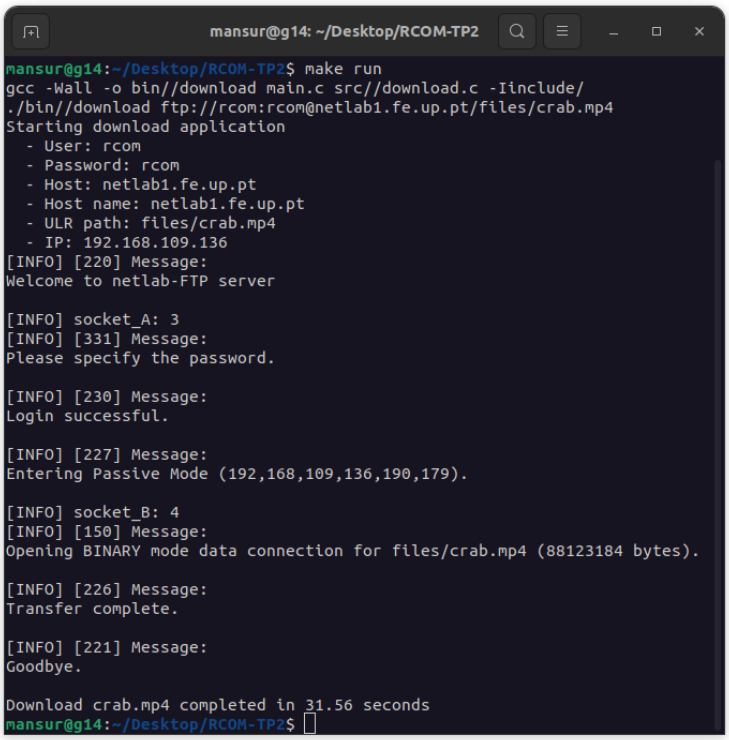
Referências

Guião 1ª aula - Experiência com Protocolos de Texto [Moodle]
Guide de trabalho 2 [Moodle]
RFC 959 [Moodle -> RFCs relevantes]

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Annexes

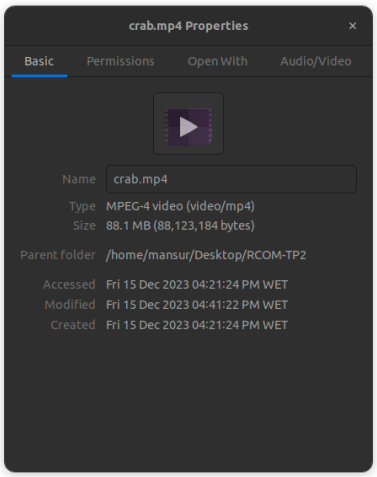
Part 1: output download url = `ftp://rcom:rcom@netlab1.fe.up.pt/files/crab.mp4`



Part 1: logs for download url = ftp://rcom:rcom@netlab1.fe.up.pt/files/crab.mp4

17	32.013798019	Routerbo_2b:84:75	Spanning-tree-(for-…	STP	60 RST. Root = 32768/0/c4:ad:34:2b:84:74 Cost = 0 Port = 0x8002
18	34.016036768	Routerbo_2b:84:75	Spanning-tree-(for-…	STP	60 RST. Root = 32768/0/c4:ad:34:2b:84:74 Cost = 0 Port = 0x8002
19	34.890913206	192.168.10.1	193.136.28.10	DNS	76 Standard query 0xfb98 A netlab1.fe.up.pt
20	34.891785662	193.136.28.10	192.168.10.1	DNS	286 Standard query response 0xfb98 A netlab1.fe.up.pt A 192.168.109.136 NS cns1.fe.up.pt NS cns2.fe.up.pt NS ns2.fe.up…
21	34.891934634	192.168.10.1	192.168.109.136	TCP	74 47298 → 21 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=3178966175 TSecr=0 WS=128
22	34.892592398	192.168.109.136	192.168.10.1	TCP	74 21 → 47298 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 TSval=2378755730 TSecr=3178966175 WS=128
23	34.892626620	192.168.10.1	192.168.109.136	TCP	66 47298 → 21 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=3178966176 TSecr=2378755730
24	34.894448429	192.168.109.136	192.168.10.1	FTP	100 Response: 220 Welcome to netlab-FTP server
25	34.894458207	192.168.10.1	192.168.109.136	TCP	66 47298 → 21 [ACK] Seq=1 Ack=35 Win=64256 Len=0 TSval=3178966178 TSecr=2378755732
26	34.894536010	192.168.10.1	192.168.109.136	FTP	77 Request: USER rcom
27	34.895028460	192.168.109.136	192.168.10.1	TCP	66 21 → 47298 [ACK] Seq=35 Ack=12 Win=65280 Len=0 TSval=2378755733 TSecr=3178966178
28	34.895084054	192.168.109.136	192.168.10.1	FTP	100 Response: 331 Please specify the password.
29	34.895154524	192.168.10.1	192.168.109.136	FTP	77 Request: PASS rcom
30	34.895644669	192.168.109.136	192.168.10.1	TCP	66 21 → 47298 [ACK] Seq=69 Ack=23 Win=65280 Len=0 TSval=2378755733 TSecr=3178966178
31	34.904217121	192.168.109.136	192.168.10.1	FTP	89 Response: 230 Login successful.
32	34.904283889	192.168.10.1	192.168.109.136	FTP	72 Request: pasv
33	34.905043412	192.168.109.136	192.168.10.1	TCP	66 21 → 47298 [ACK] Seq=92 Ack=29 Win=65280 Len=0 TSval=2378755743 TSecr=3178966188
34	34.905239527	192.168.109.136	192.168.10.1	FTP	119 Response: 227 Entering Passive Mode (192,168,109,136,169,65).
35	34.905347082	192.168.10.1	192.168.109.136	TCP	74 51578 → 43329 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=3178966189 TSecr=0 WS=128
36	34.905993044	192.168.109.136	192.168.10.1	TCP	74 43329 → 51578 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 TSval=2378755744 TSecr=3178966189 WS=128
37	34.906020072	192.168.10.1	192.168.109.136	TCP	66 51578 → 43329 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=3178966189 TSecr=2378755744
38	34.906039967	192.168.10.1	192.168.109.136	FTP	87 Request: retr files/crab.mp4
39	34.906773240	192.168.109.136	192.168.10.1	TCP	66 21 → 47298 [ACK] Seq=145 Ack=50 Win=65280 Len=0 TSval=2378755744 TSecr=3178966189
40	34.906975570	192.168.109.136	192.168.10.1	FTP	144 Response: 150 Opening BINARY mode data connection for files/crab.mp4 (88123184 bytes).
41	34.907353481	192.168.109.136	192.168.10.1	FTP-DATA	1514 FTP Data: 1448 bytes (PASV) (retr files/crab.mp4)
42	34.907370382	192.168.10.1	192.168.109.136	TCP	66 51578 → 43329 [ACK] Seq=1 Ack=1449 Win=64128 Len=0 TSval=3178966191 TSecr=2378755745
43	34.907458941	192.168.109.136	192.168.10.1	FTP-DATA	1514 FTP Data: 1448 bytes (PASV) (retr files/crab.mp4)
44	34.907470884	192.168.10.1	192.168.109.136	TCP	66 51578 → 43329 [ACK] Seq=1 Ack=2897 Win=63488 Len=0 TSval=3178966191 TSecr=2378755745
45	34.907597227	192.168.109.136	192.168.10.1	FTP-DATA	1514 FTP Data: 1448 bytes (PASV) (retr files/crab.mp4)
46	34.907606096	192.168.10.1	192.168.109.136	TCP	66 51578 → 43329 [ACK] Seq=1 Ack=4345 Win=62464 Len=0 TSval=3178966191 TSecr=2378755745
47	34.907719868	192.168.109.136	192.168.10.1	FTP-DATA	1514 FTP Data: 1448 bytes (PASV) (retr files/crab.mp4)
48	34.907728738	192.168.10.1	192.168.109.136	TCP	66 51578 → 43329 [ACK] Seq=1 Ack=5793 Win=61440 Len=0 TSval=3178966191 TSecr=2378755745
49	34.907842230	192.168.109.136	192.168.10.1	FTP-DATA	1514 FTP Data: 1448 bytes (PASV) (retr files/crab.mp4)
50	34.907849493	192.168.10.1	192.168.109.136	TCP	66 51578 → 43329 [ACK] Seq=1 Ack=7241 Win=60544 Len=0 TSval=3178966191 TSecr=2378755745

Part 1: crab.mp4 file



Experiência 1: Ping de tux3 para tux4

```
 11 16.005644014  HewlettPacka_5a:7d…  Broadcast            ARP      42 Who has 172.16.10.254? Tell 172.16.10.1
 12 16.005747728  KYE_25:13:65         HewlettPacka_5a:7d…  ARP      60 172.16.10.254 is at 00:c0:df:25:13:65
 13 16.005765608  172.16.10.1          172.16.10.254        ICMP     98 Echo (ping) request  id=0x11e9, seq=1/256, ttl=64 (reply in 14)
 14 16.005854795  172.16.10.254        172.16.10.1          ICMP     98 Echo (ping) reply    id=0x11e9, seq=1/256, ttl=64 (request in 13)
 15 16.017447331  Routerboardc_2b:84…  Spanning-tree-(for…  STP      60 RST. Root = 32768/0/c4:ad:34:2b:84:74  Cost = 0  Port = 0x8012
 16 17.019822563  172.16.10.1          172.16.10.254        ICMP     98 Echo (ping) request  id=0x11e9, seq=2/512, ttl=64 (reply in 17)
 17 17.019928512  172.16.10.254        172.16.10.1          ICMP     98 Echo (ping) reply    id=0x11e9, seq=2/512, ttl=64 (request in 16)
 18 18.019646353  Routerboardc_2b:84…  Spanning-tree-(for…  STP      60 RST. Root = 32768/0/c4:ad:34:2b:84:74  Cost = 0  Port = 0x8012
 19 18.043816256  172.16.10.1          172.16.10.254        ICMP     98 Echo (ping) request  id=0x11e9, seq=3/768, ttl=64 (reply in 20)
 20 18.043940783  172.16.10.254        172.16.10.1          ICMP     98 Echo (ping) reply    id=0x11e9, seq=3/768, ttl=64 (request in 19)
```

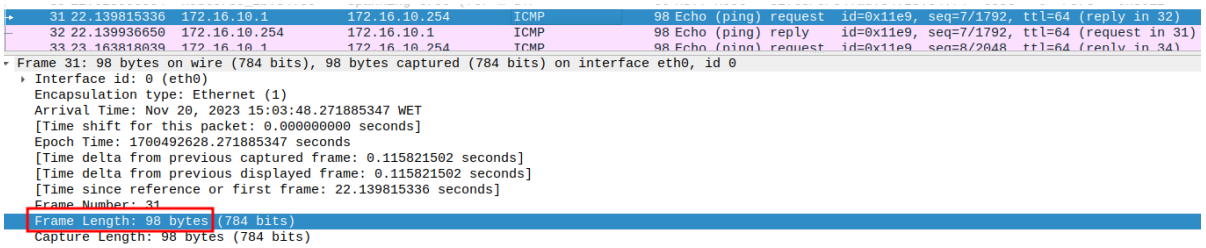
Experiência 1: pacote ICMP (1) 0x0800

```
 19 18.043816256  172.16.10.1          172.16.10.254        ICMP     98 Echo (ping) request  id=0x11e9, seq=3/768, ttl=64 (reply in 20)
 20 18.043940783  172.16.10.254        172.16.10.1          ICMP     98 Echo (ping) reply    id=0x11e9, seq=3/768, ttl=64 (request in 19)
 21 19.067819728  172.16.10.1          172.16.10.254        ICMP     98 Echo (ping) request  id=0x11e9, seq=4/1024, ttl=64 (reply in 22)
 22 19.067919042  172.16.10.254        172.16.10.1          ICMP     98 Echo (ping) reply    id=0x11e9, seq=4/1024, ttl=64 (request in 21)
 23 20.021822259  Routerbo_2b:84:85     Spanning-tree-(for-… STP      60 RST. Root = 32768/0/c4:ad:34:2b:84:74  Cost = 0  Port = 0x8012
 24 20.091816774  172.16.10.1          172.16.10.254        ICMP     98 Echo (ping) request  id=0x11e9, seq=5/1280, ttl=64 (reply in 25)
 25 20.091907079  172.16.10.254        172.16.10.1          ICMP     98 Echo (ping) reply    id=0x11e9, seq=5/1280, ttl=64 (request in 24)
 26 21.115818639  172.16.10.1          172.16.10.254        ICMP     98 Echo (ping) request  id=0x11e9, seq=6/1536, ttl=64 (reply in 27)
 27 21.115918512  172.16.10.254        172.16.10.1          ICMP     98 Echo (ping) reply    id=0x11e9, seq=6/1536, ttl=64 (request in 26)
 28 21.188488914  KYE_25:13:65         HewlettP_5a:7d:16    ARP      60 Who has 172.16.10.1? Tell 172.16.10.254
 29 21.188497504  HewlettP_5a:7d:16    KYE_25:13:65         ARP      42 172.16.10.1 is at 00:21:5a:5a:7d:16
▸ Frame 19: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface eth0, id 0
▾ Ethernet II, Src: HewlettP_5a:7d:16 (00:21:5a:5a:7d:16), Dst: KYE_25:13:65 (00:c0:df:25:13:65)
  ▸ Destination: KYE_25:13:65 (00:c0:df:25:13:65)
  ▸ Source: HewlettP_5a:7d:16 (00:21:5a:5a:7d:16)
    Type: IPv4 (0x0800)
▾ Internet Protocol Version 4, Src: 172.16.10.1, Dst: 172.16.10.254
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  ▸ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 84
    Identification: 0x4e52 (20050)
  ▸ Flags: 0x40, Don't fragment
    ...0 0000 0000 0000 = Fragment Offset: 0
    Time to Live: 64
    Protocol: ICMP (1)
```

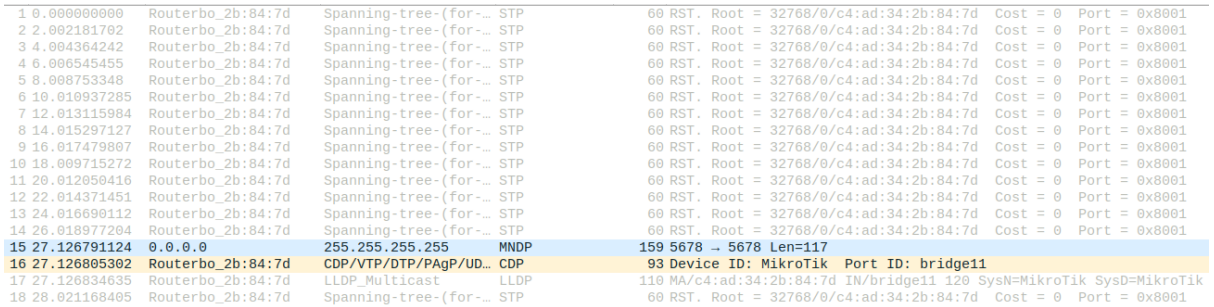
Experiência 1: pacote ARP 0x0806

```
 28 21.188488914  KYE_25:13:65         HewlettP_5a:7d:16    ARP      60 Who has 172.16.10.1? Tell 172.16.10.254
 29 21.188497504  HewlettP_5a:7d:16    KYE_25:13:65         ARP      42 172.16.10.1 is at 00:21:5a:5a:7d:16
 30 22.023993834  Routerbo_2b:84:85     Spanning-tree-(for-… STP      60 RST. Root = 32768/0/c4:ad:34:2b:84:74  Cost = 0  Port = 0x8012
 31 22.139815336  172.16.10.1          172.16.10.254        ICMP     98 Echo (ping) request  id=0x11e9, seq=7/1792, ttl=64 (reply in 32)
 32 22.139936650  172.16.10.254        172.16.10.1          ICMP     98 Echo (ping) reply    id=0x11e9, seq=7/1792, ttl=64 (request in 31)
▸ Frame 29: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface eth0, id 0
▾ Ethernet II, Src: HewlettP_5a:7d:16 (00:21:5a:5a:7d:16), Dst: KYE_25:13:65 (00:c0:df:25:13:65)
  ▸ Destination: KYE_25:13:65 (00:c0:df:25:13:65)
  ▸ Source: HewlettP_5a:7d:16 (00:21:5a:5a:7d:16)
    Type: ARP (0x0806)
▸ Address Resolution Protocol (reply)
```

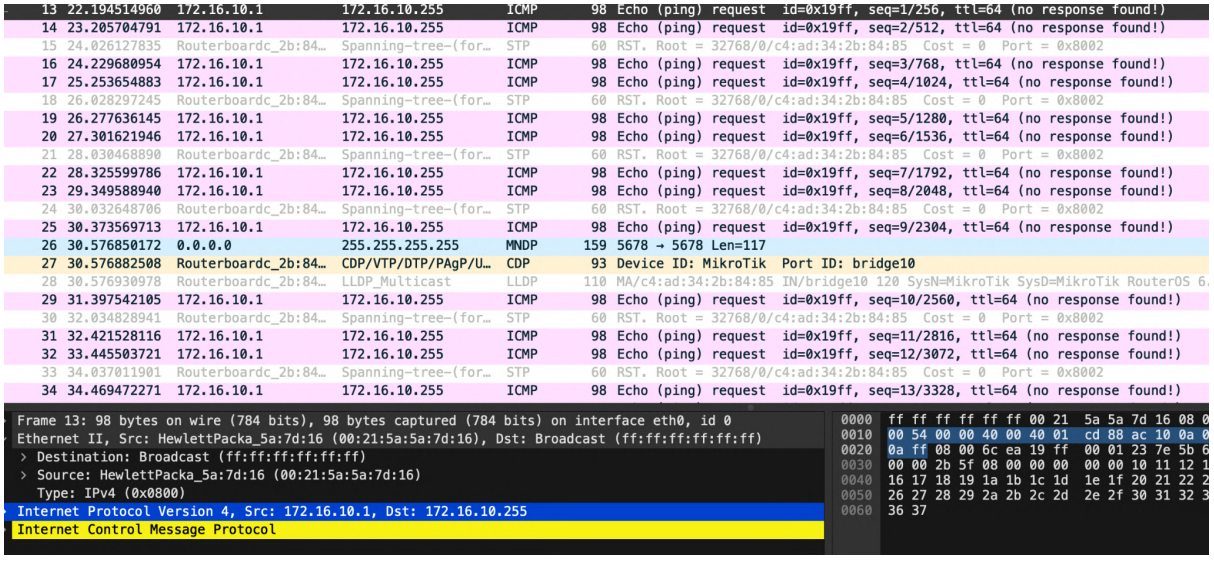
Experiência 1: “Frame Length” ICMP



Experiência 2: Broadcast do tux3 para o tux2



Experiência 2: Broadcast do tux3 para o tux4



* Na experiência foi usada bancada 2, todos IP tem forma 172.16.2x.x

Experiência 3: 172.16.20.254

8	14.005922891	Routerbo_2b:fa:11	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:2b:fa:11 Cost = 0 Port = 0x8001
9	16.008193990	Routerbo_2b:fa:11	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:2b:fa:11 Cost = 0 Port = 0x8001
10	18.010466974	Routerbo_2b:fa:11	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:2b:fa:11 Cost = 0 Port = 0x8001
11	20.012740448	Routerbo_2b:fa:11	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:2b:fa:11 Cost = 0 Port = 0x8001
12	21.179176347	0.0.0.0	255.255.255.255	MNDP	159	5678 → 5678 Len=117
13	21.179202189	Routerbo_2b:fa:11	CDP/VTP/DTP/PAgP/UD…	CDP	93	Device ID: MikroTik Port ID: bridge20
14	21.179248703	Routerbo_2b:fa:11	LLDP_Multicast	LLDP	110	MA/c4:ad:34:2b:fa:11 IN/bridge20 120 SysN=MikroTik SysD=MikroTik RouterOS
15	22.005012999	Routerbo_2b:fa:11	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:2b:fa:11 Cost = 0 Port = 0x8001
16	24.007291920	Routerbo_2b:fa:11	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:2b:fa:11 Cost = 0 Port = 0x8001
17	25.208921890	172.16.20.1	172.16.20.254	ICMP	98	Echo (ping) request id=0x0971, seq=1/256, ttl=64 (reply in 18)
18	25.209123242	172.16.20.254	172.16.20.1	ICMP	98	Echo (ping) reply id=0x0971, seq=1/256, ttl=64 (request in 17)
19	26.009555825	Routerbo_2b:fa:11	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:2b:fa:11 Cost = 0 Port = 0x8001
20	26.225161176	172.16.20.1	172.16.20.254	ICMP	98	Echo (ping) request id=0x0971, seq=2/512, ttl=64 (reply in 21)
21	26.225310496	172.16.20.254	172.16.20.1	ICMP	98	Echo (ping) reply id=0x0971, seq=2/512, ttl=64 (request in 20)
22	27.249142857	172.16.20.1	172.16.20.254	ICMP	98	Echo (ping) request id=0x0971, seq=3/768, ttl=64 (reply in 23)
23	27.249288336	172.16.20.254	172.16.20.1	ICMP	98	Echo (ping) reply id=0x0971, seq=3/768, ttl=64 (request in 22)
24	28.011823432	Routerbo_2b:fa:11	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:2b:fa:11 Cost = 0 Port = 0x8001

Experiência 3: 172.16.21.253

3	3.993887281	Routerbo_2b:fa:11	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:2b:fa:11 Cost = 0 Port = 0x8001
4	5.996221447	Routerbo_2b:fa:11	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:2b:fa:11 Cost = 0 Port = 0x8001
5	7.998550305	Routerbo_2b:fa:11	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:2b:fa:11 Cost = 0 Port = 0x8001
6	10.000872737	Routerbo_2b:fa:11	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:2b:fa:11 Cost = 0 Port = 0x8001
7	11.201718529	172.16.20.1	172.16.21.253	ICMP	98	Echo (ping) request id=0x09d4, seq=1/256, ttl=64 (reply in 8)
8	11.201905145	172.16.21.253	172.16.20.1	ICMP	98	Echo (ping) reply id=0x09d4, seq=1/256, ttl=64 (request in 7)
9	11.993201929	Routerbo_2b:fa:11	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:2b:fa:11 Cost = 0 Port = 0x8001
10	12.226193150	172.16.20.1	172.16.21.253	ICMP	98	Echo (ping) request id=0x09d4, seq=2/512, ttl=64 (reply in 11)
11	12.226336604	172.16.21.253	172.16.20.1	ICMP	98	Echo (ping) reply id=0x09d4, seq=2/512, ttl=64 (request in 10)
12	13.250188519	172.16.20.1	172.16.21.253	ICMP	98	Echo (ping) request id=0x09d4, seq=3/768, ttl=64 (reply in 13)
13	13.250367523	172.16.21.253	172.16.20.1	ICMP	98	Echo (ping) reply id=0x09d4, seq=3/768, ttl=64 (request in 12)
14	13.995542590	Routerbo_2b:fa:11	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:2b:fa:11 Cost = 0 Port = 0x8001
15	14.274185705	172.16.20.1	172.16.21.253	ICMP	98	Echo (ping) request id=0x09d4, seq=4/1024, ttl=64 (reply in 16)
16	14.274328600	172.16.21.253	172.16.20.1	ICMP	98	Echo (ping) reply id=0x09d4, seq=4/1024, ttl=64 (request in 15)
17	15.298189805	172.16.20.1	172.16.21.253	ICMP	98	Echo (ping) request id=0x09d4, seq=5/1280, ttl=64 (reply in 18)
18	15.298336192	172.16.21.253	172.16.20.1	ICMP	98	Echo (ping) reply id=0x09d4, seq=5/1280, ttl=64 (request in 17)
19	15.997886394	Routerbo_2b:fa:11	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:2b:fa:11 Cost = 0 Port = 0x8001

Experiência 3: 172.16.21.1

5	8.008846726	Routerbo_2b:fa:11	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:2b:fa:11 Cost = 0 Port = 0x8001
6	10.011052453	Routerbo_2b:fa:11	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:2b:fa:11 Cost = 0 Port = 0x8001
7	12.013275501	Routerbo_2b:fa:11	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:2b:fa:11 Cost = 0 Port = 0x8001
8	14.015482835	Routerbo_2b:fa:11	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:2b:fa:11 Cost = 0 Port = 0x8001
9	15.798835802	172.16.20.1	172.16.21.1	ICMP	98	Echo (ping) request id=0x0a0a, seq=1/256, ttl=64 (reply in 10)
10	15.799308418	172.16.21.1	172.16.20.1	ICMP	98	Echo (ping) reply id=0x0a0a, seq=1/256, ttl=63 (request in 9)
11	16.017682277	Routerbo_2b:fa:11	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:2b:fa:11 Cost = 0 Port = 0x8001
12	16.815204155	172.16.20.1	172.16.21.1	ICMP	98	Echo (ping) request id=0x0a0a, seq=2/512, ttl=64 (reply in 13)
13	16.815465431	172.16.21.1	172.16.20.1	ICMP	98	Echo (ping) reply id=0x0a0a, seq=2/512, ttl=63 (request in 12)
14	17.839184997	172.16.20.1	172.16.21.1	ICMP	98	Echo (ping) request id=0x0a0a, seq=3/768, ttl=64 (reply in 15)
15	17.839446483	172.16.21.1	172.16.20.1	ICMP	98	Echo (ping) reply id=0x0a0a, seq=3/768, ttl=63 (request in 14)
16	18.019891008	Routerbo_2b:fa:11	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:2b:fa:11 Cost = 0 Port = 0x8001

Experiencia 3 eth0: with ARP

106	188.205687264	Routerbo_2b:fa:13	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:2b:fa:11 Cost = 0 Port = 0x8002
107	190.207889779	Routerbo_2b:fa:13	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:2b:fa:11 Cost = 0 Port = 0x8002
108	191.543154061	HewlettP_5a:78:c7	Broadcast	ARP	60	Who has 172.16.20.254? Tell 172.16.20.1
109	191.543181648	HewlettP_a7:26:a2	HewlettP_5a:78:c7	ARP	42	172.16.20.254 is at 00:22:64:a7:26:a2
110	191.543309108	172.16.20.1	172.16.21.1	ICMP	98	Echo (ping) request id=0x0b4e, seq=1/256, ttl=64 (reply in 111)
111	191.543607121	172.16.21.1	172.16.20.1	ICMP	98	Echo (ping) reply id=0x0b4e, seq=1/256, ttl=63 (request in 110)
112	192.210090338	Routerbo_2b:fa:13	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:2b:fa:11 Cost = 0 Port = 0x8002
113	192.558317162	172.16.20.1	172.16.21.1	ICMP	98	Echo (ping) request id=0x0b4e, seq=2/512, ttl=64 (reply in 114)
114	192.558475283	172.16.21.1	172.16.20.1	ICMP	98	Echo (ping) reply id=0x0b4e, seq=2/512, ttl=63 (request in 113)
115	193.582316373	172.16.20.1	172.16.21.1	ICMP	98	Echo (ping) request id=0x0b4e, seq=3/768, ttl=64 (reply in 116)
116	193.582503548	172.16.21.1	172.16.20.1	ICMP	98	Echo (ping) reply id=0x0b4e, seq=3/768, ttl=63 (request in 115)
117	194.212292713	Routerbo_2b:fa:13	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:2b:fa:11 Cost = 0 Port = 0x8002
118	194.606324315	172.16.20.1	172.16.21.1	ICMP	98	Echo (ping) request id=0x0b4e, seq=4/1024, ttl=64 (reply in 119)
119	194.606486556	172.16.21.1	172.16.20.1	ICMP	98	Echo (ping) reply id=0x0b4e, seq=4/1024, ttl=63 (request in 118)
120	195.630322827	172.16.20.1	172.16.21.1	ICMP	98	Echo (ping) request id=0x0b4e, seq=5/1280, ttl=64 (reply in 121)
121	195.630488211	172.16.21.1	172.16.20.1	ICMP	98	Echo (ping) reply id=0x0b4e, seq=5/1280, ttl=63 (request in 120)
122	196.023761965	0.0.0.0	255.255.255.255	MNDP	159	5678 → 5678 Len=117
123	196.023789133	Routerbo_2b:fa:11	CDP/VTP/DTP/PAgP/UD…	CDP	93	Device ID: MikroTik Port ID: bridge20
124	196.023838860	Routerbo_2b:fa:11	LLDP_Multicast	LLDP	110	MA/c4:ad:34:2b:fa:11 IN/bridge20 120 SysN=MikroTik SysD=MikroTik RouterOS
125	196.214499488	Routerbo_2b:fa:13	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:2b:fa:11 Cost = 0 Port = 0x8002
126	196.654346413	172.16.20.1	172.16.21.1	ICMP	98	Echo (ping) request id=0x0b4e, seq=6/1536, ttl=64 (reply in 127)
127	196.654506280	172.16.21.1	172.16.20.1	ICMP	98	Echo (ping) reply id=0x0b4e, seq=6/1536, ttl=63 (request in 126)
128	196.698290549	HewlettP_a7:26:a2	HewlettP_5a:78:c7	ARP	42	Who has 172.16.20.1? Tell 172.16.20.254
129	196.698416892	HewlettP_5a:78:c7	HewlettP_a7:26:a2	ARP	60	172.16.20.1 is at 00:21:5a:5a:78:c7
130	197.678323065	172.16.20.1	172.16.21.1	ICMP	98	Echo (ping) request id=0x0b4e, seq=7/1792, ttl=64 (reply in 131)

Experiencia 3 eth1: with ARP

```
101 180.196885516 Routerbo_2b:fa:19  Spanning-tree-(for-… STP   60 RST. Root = 32768/0/c4:ad:34:2b:fa:0d  Cost = 0  Port = 0x8002
102 182.199089218 Routerbo_2b:fa:19  Spanning-tree-(for-… STP   60 RST. Root = 32768/0/c4:ad:34:2b:fa:0d  Cost = 0  Port = 0x8002
103 183.534577202 EncoreNe_c8:7c:55  Broadcast             ARP   42 Who has 172.16.21.1? Tell 172.16.21.253
104 183.534711227 HewlettP_5a:76:a8  EncoreNe_c8:7c:55     ARP   60 172.16.21.1 is at 00:21:5a:5a:76:a8
105 183.534729107 172.16.20.1        172.16.21.1           ICMP  98 Echo (ping) request  id=0x0b4e, seq=1/256, ttl=63 (reply in 106)
106 183.534847138 172.16.21.1        172.16.20.1           ICMP  98 Echo (ping) reply    id=0x0b4e, seq=1/256, ttl=64 (request in 105)
107 184.201291244 Routerbo_2b:fa:19  Spanning-tree-(for-… STP   60 RST. Root = 32768/0/c4:ad:34:2b:fa:0d  Cost = 0  Port = 0x8002
108 184.549590843 172.16.20.1        172.16.21.1           ICMP  98 Echo (ping) request  id=0x0b4e, seq=2/512, ttl=63 (reply in 109)
109 184.549708456 172.16.21.1        172.16.20.1           ICMP  98 Echo (ping) reply    id=0x0b4e, seq=2/512, ttl=64 (request in 108)
110 185.573591730 172.16.20.1        172.16.21.1           ICMP  98 Echo (ping) request  id=0x0b4e, seq=3/768, ttl=63 (reply in 111)
111 185.573733438 172.16.21.1        172.16.20.1           ICMP  98 Echo (ping) reply    id=0x0b4e, seq=3/768, ttl=64 (request in 110)
112 186.203494527 Routerbo_2b:fa:19  Spanning-tree-(for-… STP   60 RST. Root = 32768/0/c4:ad:34:2b:fa:0d  Cost = 0  Port = 0x8002
113 186.597598833 172.16.20.1        172.16.21.1           ICMP  98 Echo (ping) request  id=0x0b4e, seq=4/1024, ttl=63 (reply in 114)
114 186.597720008 172.16.21.1        172.16.20.1           ICMP  98 Echo (ping) reply    id=0x0b4e, seq=4/1024, ttl=64 (request in 113)
115 187.621597625 172.16.20.1        172.16.21.1           ICMP  98 Echo (ping) request  id=0x0b4e, seq=5/1280, ttl=63 (reply in 116)
116 187.621720546 172.16.21.1        172.16.20.1           ICMP  98 Echo (ping) reply    id=0x0b4e, seq=5/1280, ttl=64 (request in 115)
117 188.015207246 0.0.0.0            255.255.255.255       MNDP  159 5678 → 5678 Len=117
118 188.015225125 Routerbo_2b:fa:0d  CDP/VTP/DTP/PAgP/UD… CDP   93 Device ID: MikroTik  Port ID: bridge21
119 188.015264446 Routerbo_2b:fa:0d  LLDP_Multicast       LLDP  110 MA/c4:ad:34:2b:fa:0d IN/bridge21 120 SysN=MikroTik SysD=MikroTik RouterOS
120 188.205701791 Routerbo_2b:fa:19  Spanning-tree-(for-… STP   60 RST. Root = 32768/0/c4:ad:34:2b:fa:0d  Cost = 0  Port = 0x8002
121 188.560146585 HewlettP_5a:76:a8  EncoreNe_c8:7c:55     ARP   60 Who has 172.16.21.253? Tell 172.16.21.1
122 188.560165931 EncoreNe_c8:7c:55  HewlettP_5a:76:a8     ARP   42 172.16.21.253 is at 00:e0:7d:c8:7c:55
123 188.645621630 172.16.20.1        172.16.21.1           ICMP  98 Echo (ping) request  id=0x0b4e, seq=6/1536, ttl=63 (reply in 124)
124 188.645738405 172.16.21.1        172.16.20.1           ICMP  98 Echo (ping) reply    id=0x0b4e, seq=6/1536, ttl=64 (request in 123)
125 189.669597584 172.16.20.1        172.16.21.1           ICMP  98 Echo (ping) request  id=0x0b4e, seq=7/1792, ttl=63 (reply in 126)
```

Experiência 3: routes de pc3 exp3

```
root@gnu23:~# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
172.16.20.0     0.0.0.0         255.255.255.0   U     0      0        0 eth0
172.16.21.0     172.16.20.254   255.255.255.0   UG    0      0        0 eth0
root@gnu23:~#
```

Experiência 3: routes de pc2 exp3

```
root@gnu22:~# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
172.16.20.0     172.16.21.253   255.255.255.0   UG    0      0        0 eth0
172.16.21.0     0.0.0.0         255.255.255.0   U     0      0        0 eth0
root@gnu22:~#
```

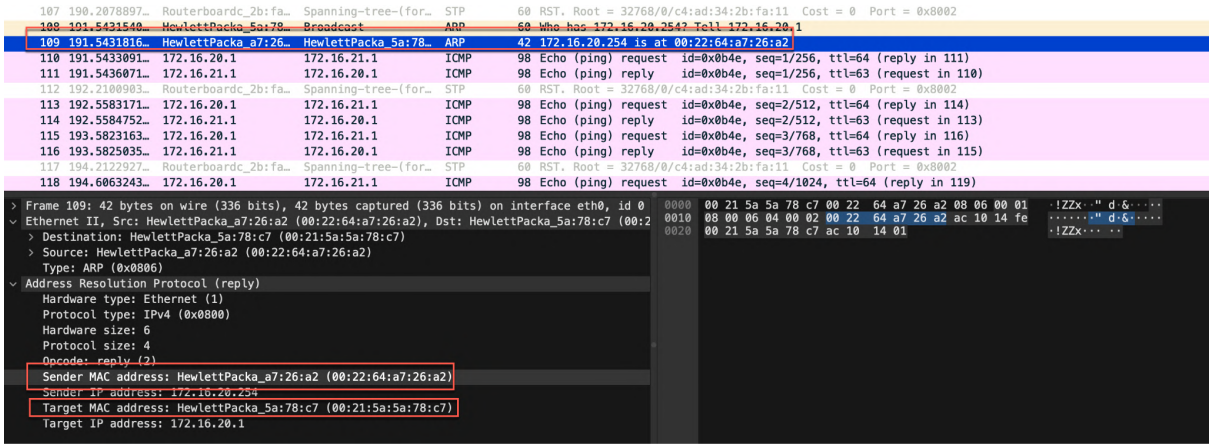
Experiência 3: routes de pc4 exp3

```
root@gnu24:~# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
172.16.20.0     0.0.0.0         255.255.255.0   U     0      0        0 eth0
172.16.21.0     0.0.0.0         255.255.255.0   U     0      0        0 eth1
root@gnu24:~#
```

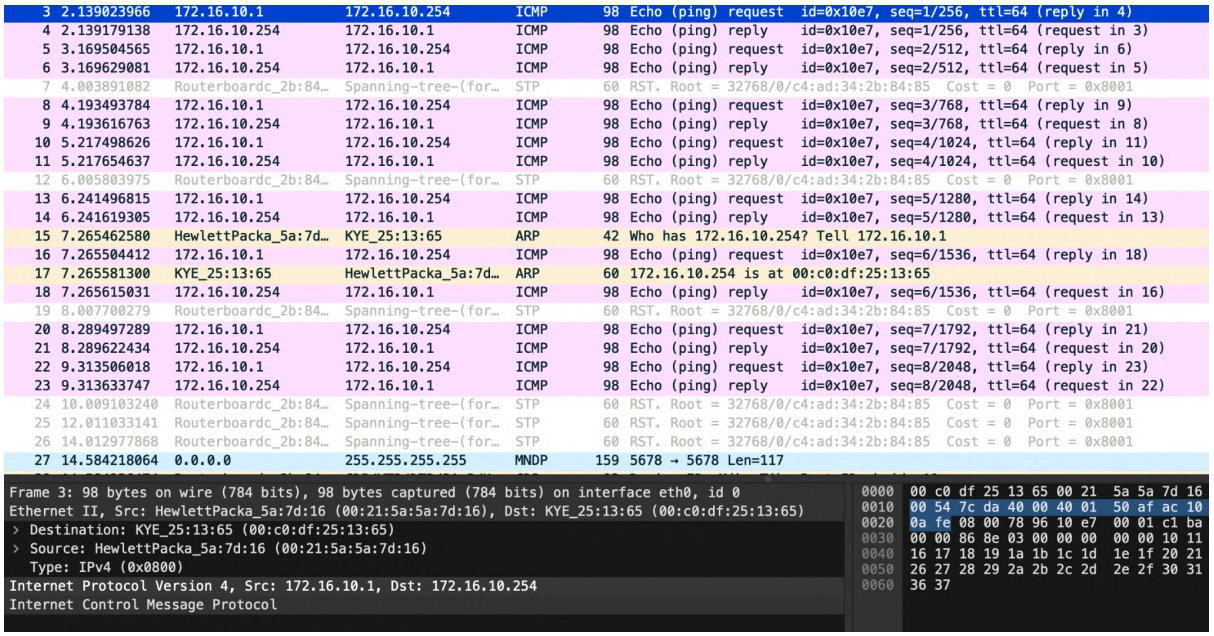
Experiência 3: ARP request de tux3 para tux4

```
107 190.2078897… Routerboardc_2b:fa… Spanning-tree-(for… STP   60 RST. Root = 32768/0/c4:ad:34:2b:fa:11  Cost = 0  Port = 0x8002
108 191.5431540… HewlettPacka_5a:78… Broadcast            ARP   60 Who has 172.16.20.254? Tell 172.16.20.1
109 191.5431816… HewlettPacka_a7:26… HewlettPacka_5a:78… ARP   42 172.16.20.254 is at 00:22:64:a7:26:a2
110 191.5433091… 172.16.20.1         172.16.21.1           ICMP  98 Echo (ping) request  id=0x0b4e, seq=1/256, ttl=64 (reply in 111)
111 191.5436071… 172.16.21.1         172.16.20.1           ICMP  98 Echo (ping) reply    id=0x0b4e, seq=1/256, ttl=63 (request in 110)
112 192.2100903… Routerboardc_2b:fa… Spanning-tree-(for… STP   60 RST. Root = 32768/0/c4:ad:34:2b:fa:11  Cost = 0  Port = 0x8002
113 192.5583171… 172.16.20.1         172.16.21.1           ICMP  98 Echo (ping) request  id=0x0b4e, seq=2/512, ttl=64 (reply in 114)
114 192.5584752… 172.16.21.1         172.16.20.1           ICMP  98 Echo (ping) reply    id=0x0b4e, seq=2/512, ttl=63 (request in 113)
115 193.5823163… 172.16.20.1         172.16.21.1           ICMP  98 Echo (ping) request  id=0x0b4e, seq=3/768, ttl=64 (reply in 116)
116 193.5825035… 172.16.21.1         172.16.20.1           ICMP  98 Echo (ping) reply    id=0x0b4e, seq=3/768, ttl=63 (request in 115)
117 194.2122927… Routerboardc_2b:fa… Spanning-tree-(for… STP   60 RST. Root = 32768/0/c4:ad:34:2b:fa:11  Cost = 0  Port = 0x8002
118 194.6063243… 172.16.20.1         172.16.21.1           ICMP  98 Echo (ping) request  id=0x0b4e, seq=4/1024, ttl=64 (reply in 119)

> Frame 108: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface eth0, id 0
v Ethernet II, Src: HewlettPacka_5a:78:c7 (00:21:5a:5a:78:c7), Dst: Broadcast (ff:ff:ff:ff:ff:f
  > Destination: Broadcast (ff:ff:ff:ff:ff:ff)
  > Source: HewlettPacka_5a:78:c7 (00:21:5a:5a:78:c7)
    Type: ARP (0x0806)
    Padding: 00000000000000000000000000000000000000
v Address Resolution Protocol (request)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (1)
    Sender MAC address: HewlettPacka_5a:78:c7 (00:21:5a:5a:78:c7)
    Sender IP address: 172.16.20.1
    Target MAC address: Xerox_00:00:00 (00:00:00:00:00:00)
    Target IP address: 172.16.20.254

0000  ff ff ff ff ff ff 00 21  5a 5a 78 c7 08 06 00 0
0010  08 00 06 04 00 01 00 21  5a 5a 78 c7 ac 10 14 0
0020  00 00 00 00 00 00 ac 10  14 fe 00 00 00 00 00 0
0030  00 00 00 00 00 00 00 00  00 00 00 00
```

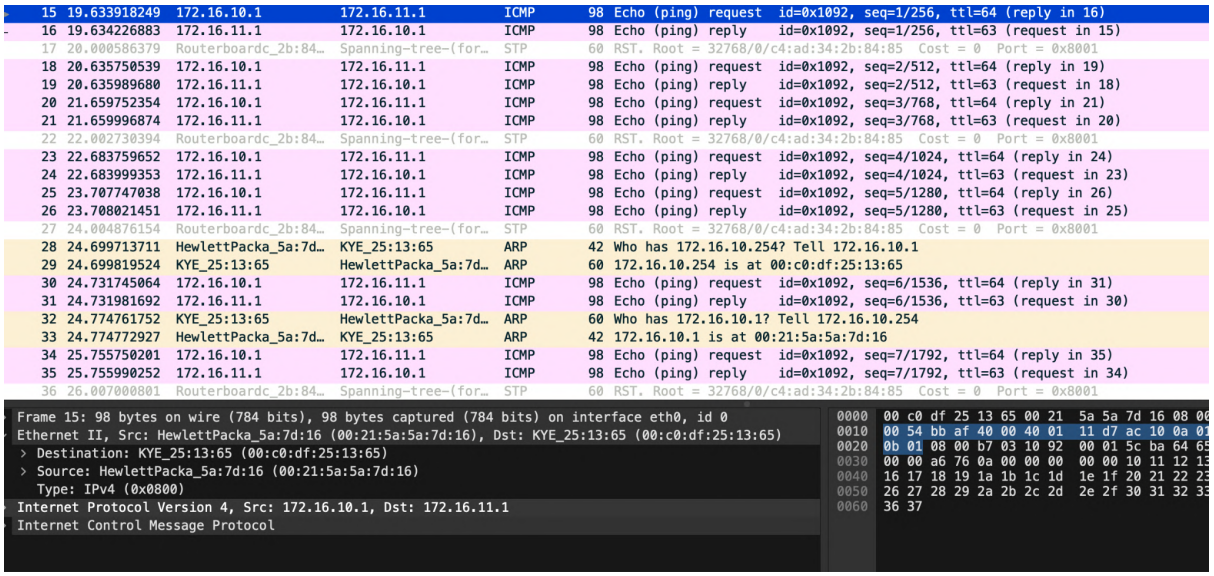
Experiência 3: ARP reply de tux4 para tux3



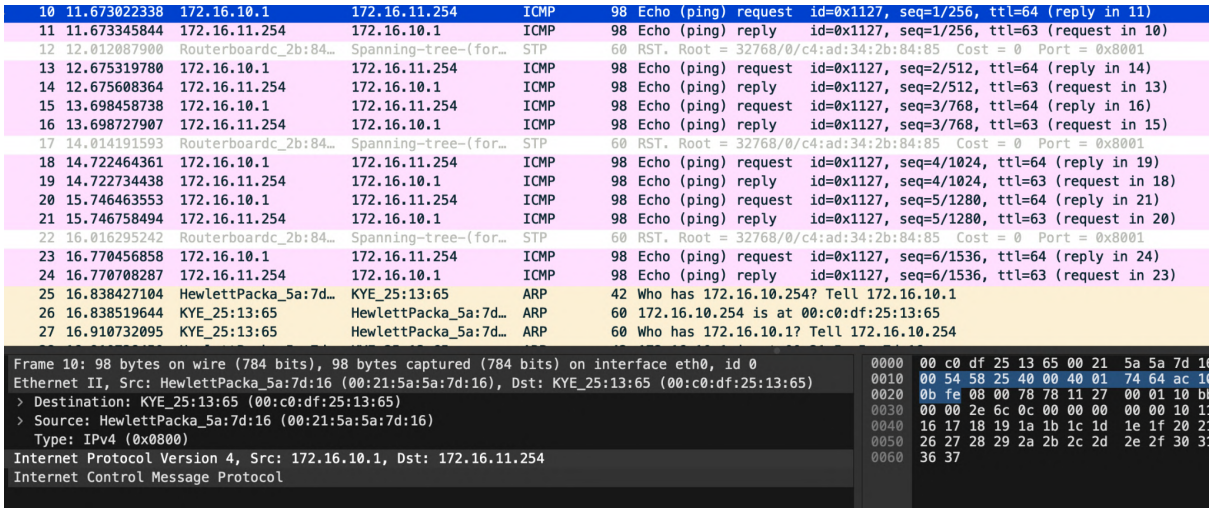
Experiência 4: ping to tux3 para o tux4 com IP 172.16.10.254



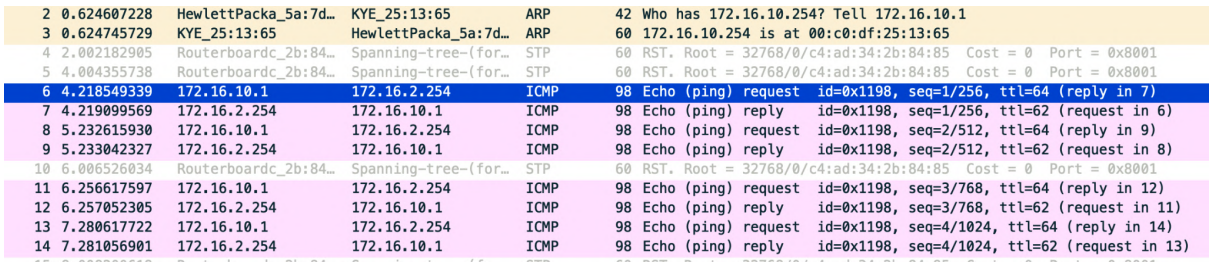
Experiência 4: ping to tux3 para o tux2 com IP 172.16.11.1



Experiência 4: ping to tux3 para o router Mikrotik com o IP 172.16.11.254



Experiência 4: ping to tux3 para o IP 172.16.2.254



Experiência 4: traceroute do tux2 para tux3 sem redirecionamentos ICMP

```
root@gnu12:~# traceroute 172.16.10.1
traceroute to 172.16.10.1 (172.16.10.1), 30 hops max, 60 byte packets
 1  172.16.11.254 (172.16.11.254)  0.197 ms  0.179 ms  0.197 ms
 2  172.16.11.253 (172.16.11.253)  0.331 ms  0.320 ms  0.306 ms
 3  172.16.10.1 (172.16.10.1)  0.484 ms  0.471 ms  0.449 ms
root@gnu12:~#
```

Experiência 4: traceroute do tux2 para tux3 com redirecionamentos ICMP

```
root@gnu12:~# traceroute -n 172.16.10.1
traceroute to 172.16.10.1 (172.16.10.1), 30 hops max, 60 byte packets
 1  172.16.11.253  0.157 ms  0.135 ms  0.128 ms
 2  172.16.10.1  0.239 ms  0.254 ms  0.238 ms
root@gnu12:~#
```

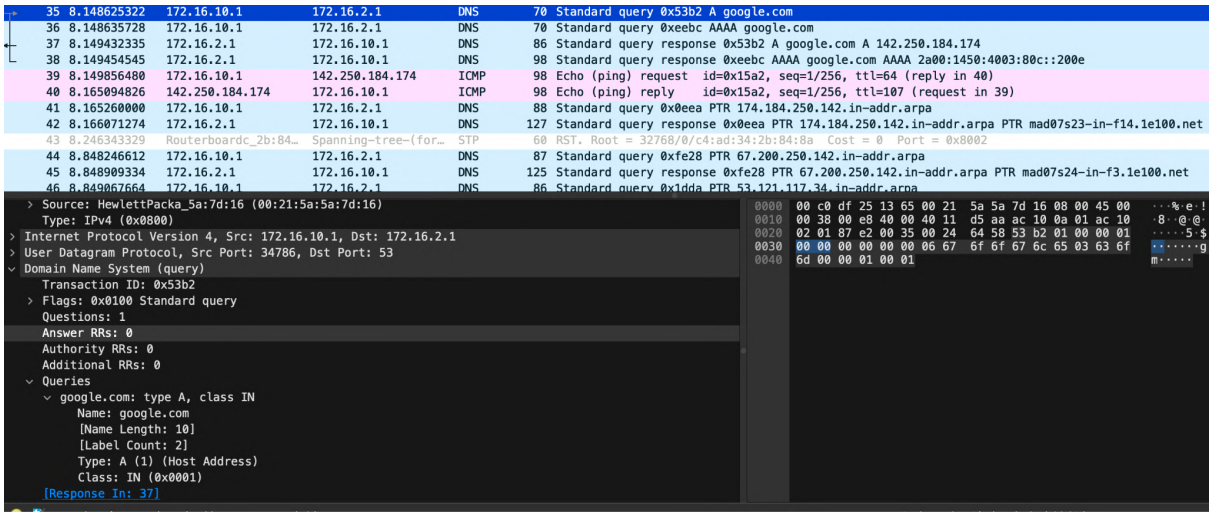
Experiência 4: conexão à Internet com NAT desativado

```
root@gnu13:~# ping 172.16.2.254
PING 172.16.2.254 (172.16.2.254) 56(84) bytes of data.
^C
--- 172.16.2.254 ping statistics ---
5 packets transmitted, 0 received, 100% packet loss, time 108ms

root@gnu13:~#
```

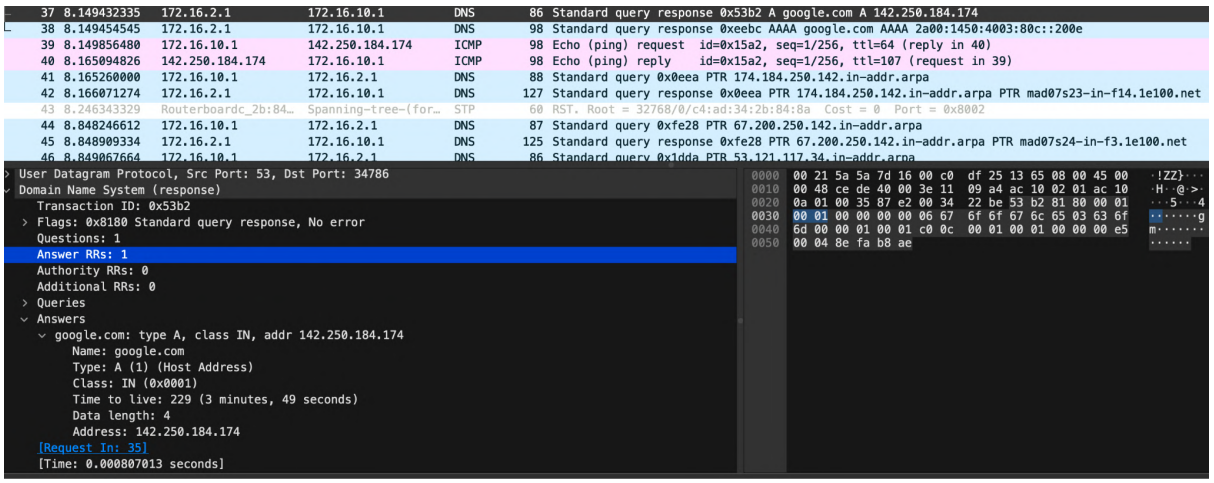
Experiência 4: conexão à Internet com NAT ativado

```
--- 172.16.2.254 ping statistics ---
7 packets transmitted, 4 received, +3 errors, 42.8571% packet loss, time 76ms
rtt min/avg/max/mdev = 0.428/249.904/998.245/432.054 ms, pipe 3
root@gnu13:~# ping 172.16.2.254
PING 172.16.2.254 (172.16.2.254) 56(84) bytes of data.
64 bytes from 172.16.2.254: icmp_seq=1 ttl=62 time=0.549 ms
64 bytes from 172.16.2.254: icmp_seq=2 ttl=62 time=0.480 ms
64 bytes from 172.16.2.254: icmp_seq=3 ttl=62 time=0.460 ms
64 bytes from 172.16.2.254: icmp_seq=4 ttl=62 time=0.461 ms
64 bytes from 172.16.2.254: icmp_seq=5 ttl=62 time=0.462 ms
64 bytes from 172.16.2.254: icmp_seq=6 ttl=62 time=0.473 ms
64 bytes from 172.16.2.254: icmp_seq=7 ttl=62 time=0.450 ms
64 bytes from 172.16.2.254: icmp_seq=8 ttl=62 time=0.445 ms
64 bytes from 172.16.2.254: icmp_seq=9 ttl=62 time=0.433 ms
64 bytes from 172.16.2.254: icmp_seq=10 ttl=62 time=0.452 ms
64 bytes from 172.16.2.254: icmp_seq=11 ttl=62 time=0.509 ms
64 bytes from 172.16.2.254: icmp_seq=12 ttl=62 time=0.433 ms
64 bytes from 172.16.2.254: icmp_seq=13 ttl=62 time=0.457 ms
64 bytes from 172.16.2.254: icmp_seq=14 ttl=62 time=0.442 ms
64 bytes from 172.16.2.254: icmp_seq=15 ttl=62 time=0.448 ms
64 bytes from 172.16.2.254: icmp_seq=16 ttl=62 time=0.488 ms
^C
--- 172.16.2.254 ping statistics ---
```

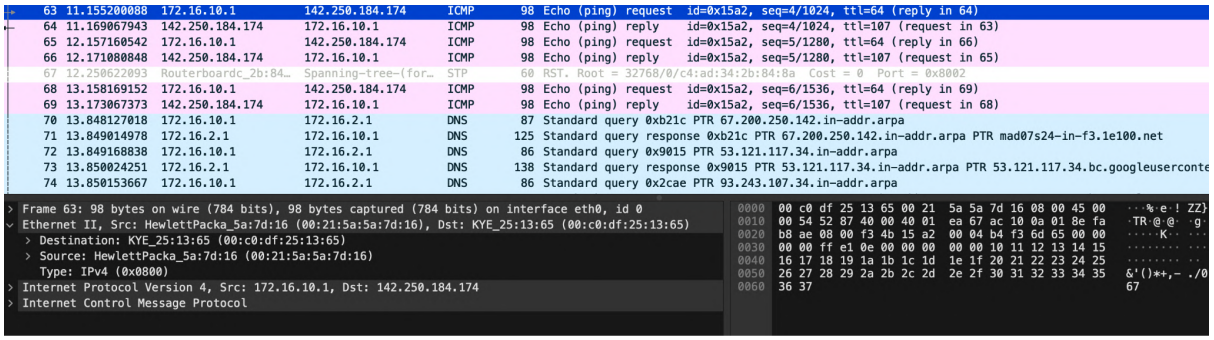
Experiência 5: pacotes DNS Query no ping do tux3 ao google.com



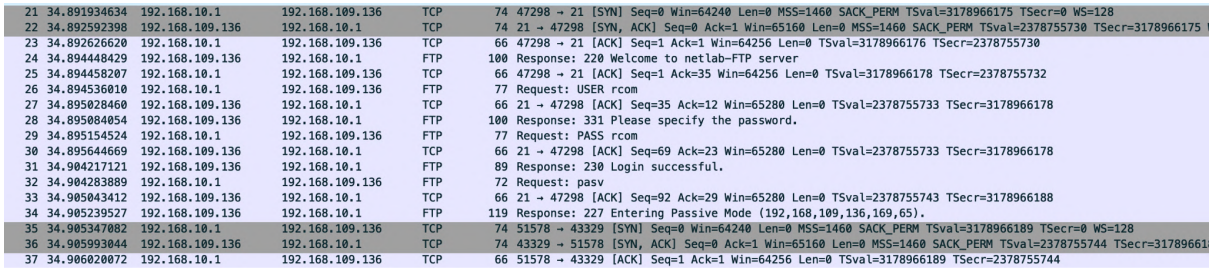
Experiência 5: pacotes DNS Query Response no ping do tux3 ao google.com



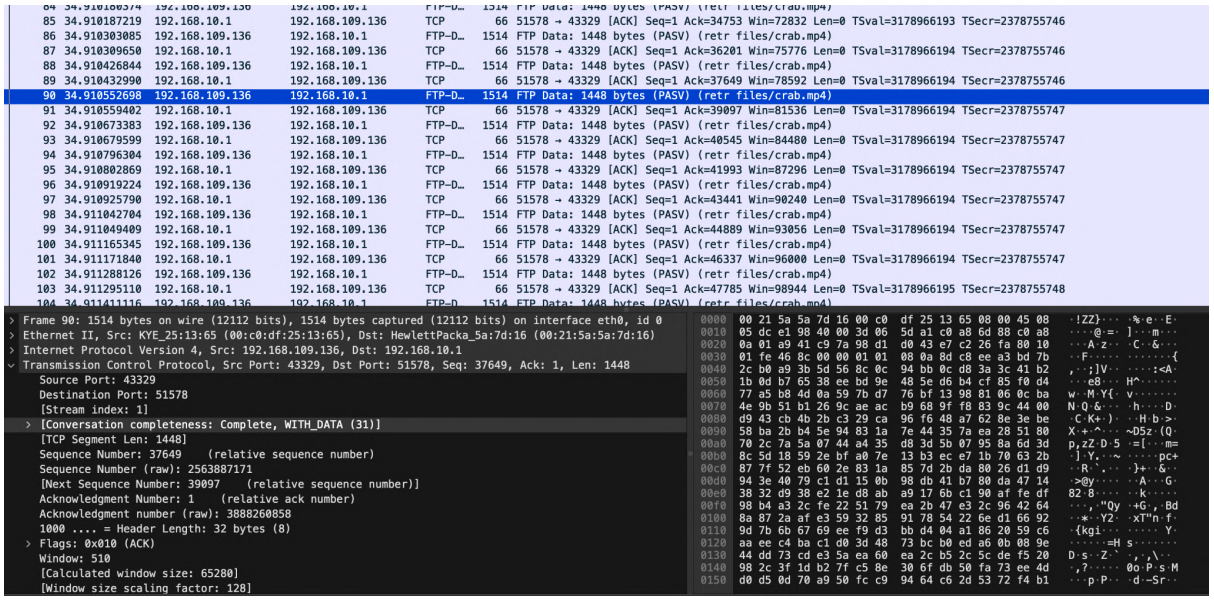
Experiência 5: Sucesso do ping do tux3 ao google.com



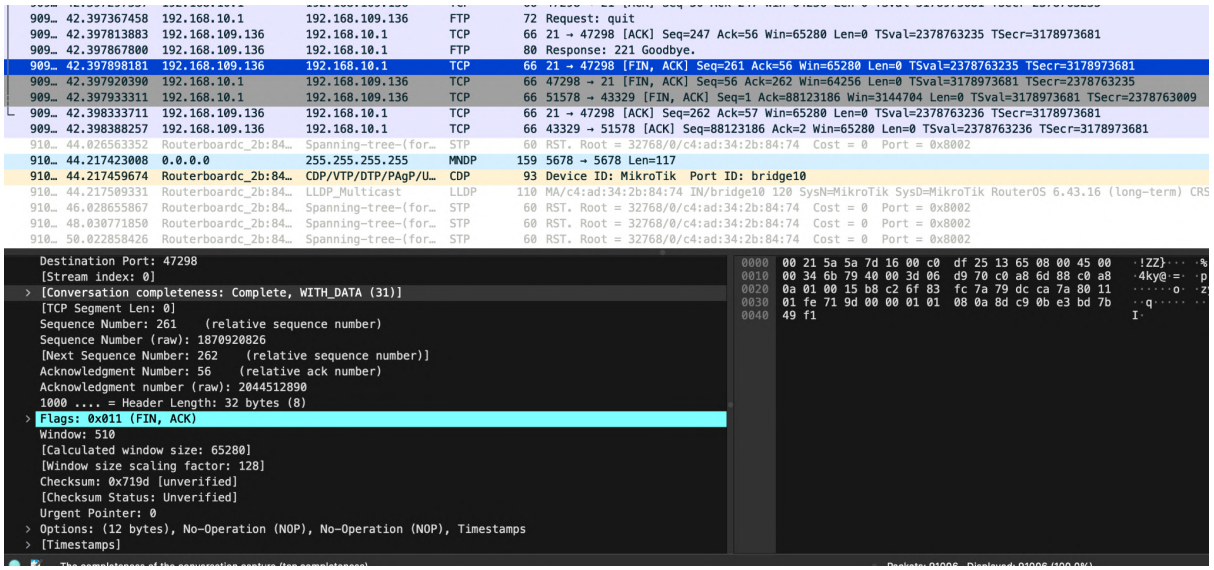
Experiência 6: Dois estabelecimentos de conexão



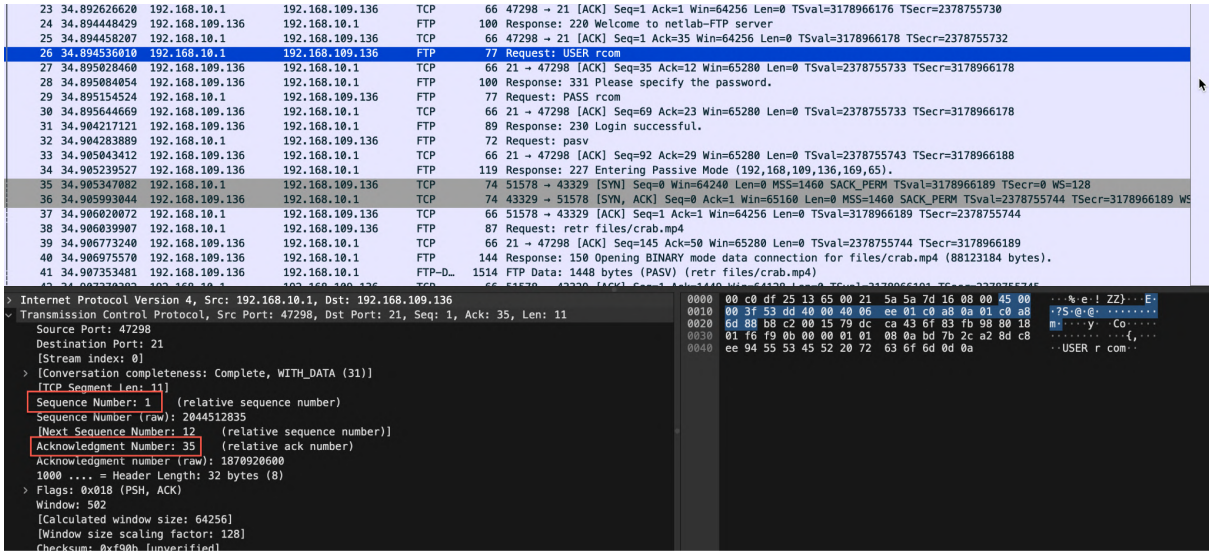
Experiência 6: Transferência de dados



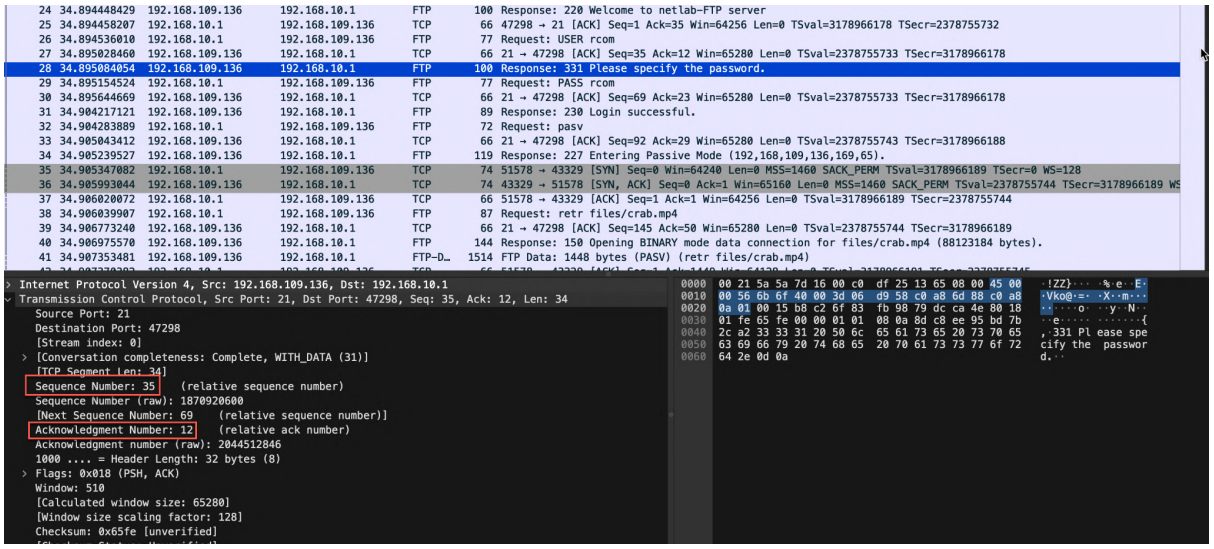
Experiência 6: Fecho da conexão TCP



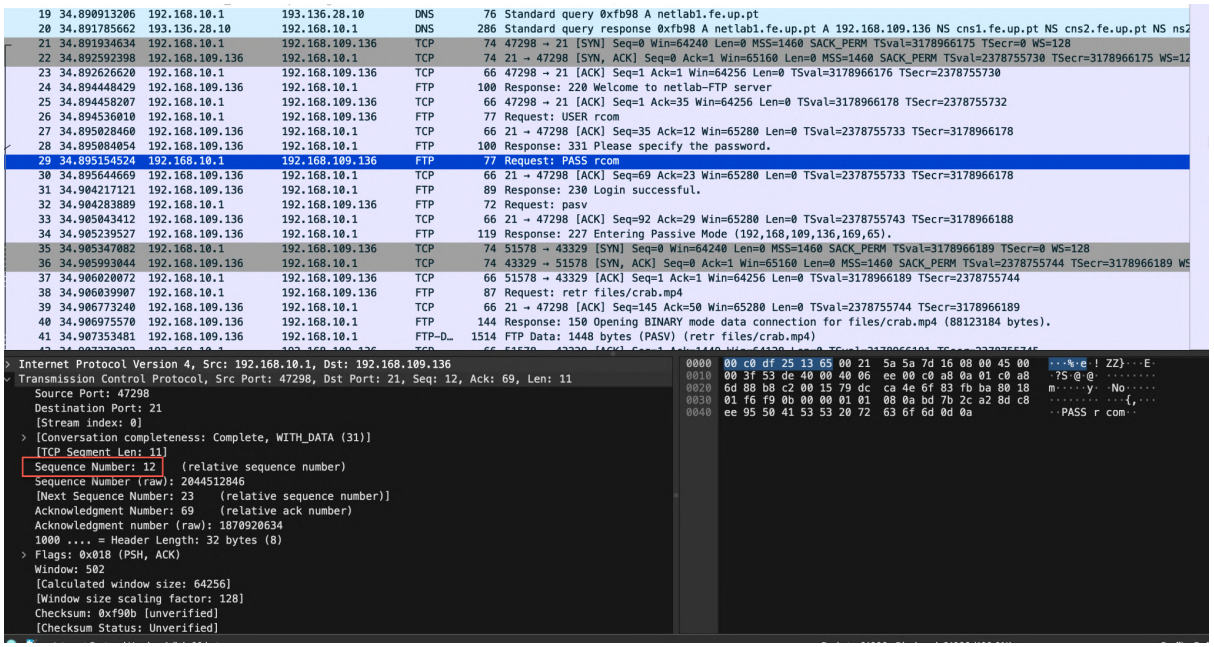
Experiência 6: Análise ARQ 1



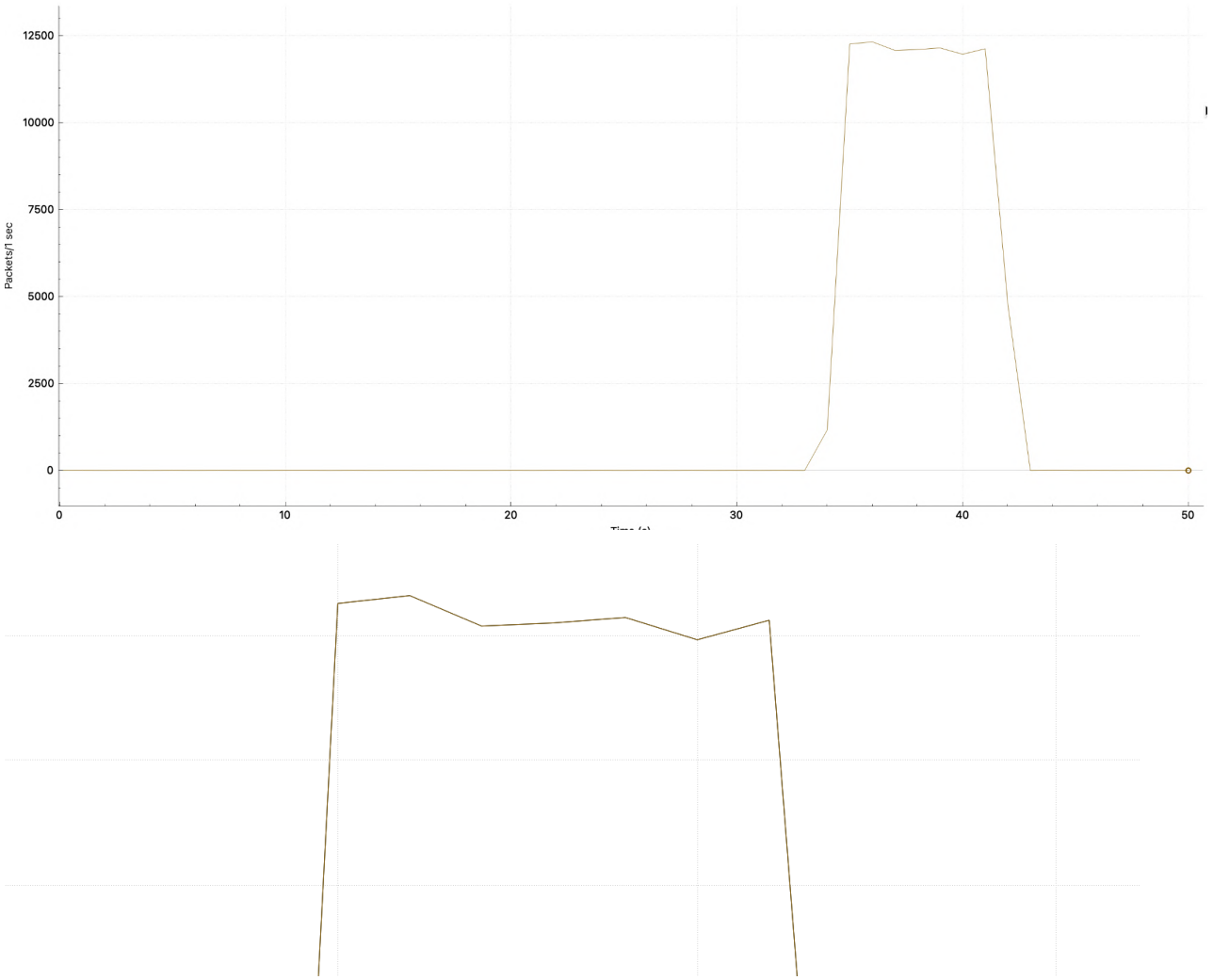
Experiência 6: Análise ARQ 2



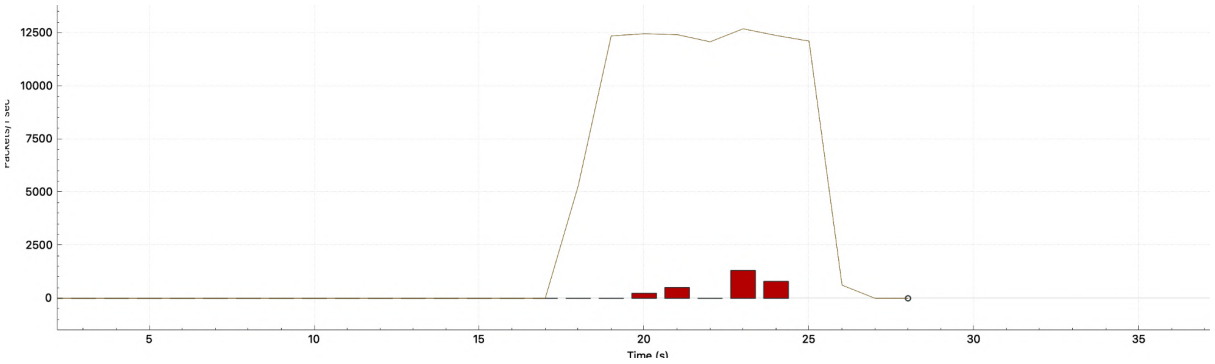
Experiência 6: Análise ARQ 3



Experiência 6: Throughput da conexão em forma de saw-tooth



Experiência 6: Throughput da conexão no tux3 com transferência simultânea no tux2



Experiência 6: logs de ARQ

```
29658 20.970461486  192.168.109.136  192.168.10.1     FTP-DATA  1514 FTP Data: 1448 bytes (PASV) (retr files/crab.mp4)
29659 20.970591670  192.168.109.136  192.168.10.1     FTP-DATA  1514 [TCP Previous segment not captured] FTP Data: 1448 bytes (PASV) (retr files/crab.mp4)
29660 20.970601588  192.168.10.1     192.168.109.136  TCP         78 48176 → 40003 [ACK] Seq=1 Ack=28337361 Win=2896384 Len=0 TSval=3180922755 TSecr=2380712235 SLE=28532841 SRE=285342…
29661 20.970689448  192.168.109.136  192.168.10.1     FTP-DATA  1514 FTP Data: 1448 bytes (PASV) (retr files/crab.mp4)
29662 20.970699991  192.168.10.1     192.168.109.136  TCP         78 [TCP Dup ACK 29660#1] 48176 → 40003 [ACK] Seq=1 Ack=28337361 Win=2896384 Len=0 TSval=3180922755 TSecr=2380712235 S…
29663 20.970833949  192.168.109.136  192.168.10.1     FTP-DATA  1514 FTP Data: 1448 bytes (PASV) (retr files/crab.mp4)
29664 20.970840445  192.168.10.1     192.168.109.136  TCP         78 [TCP Dup ACK 29660#2] 48176 → 40003 [ACK] Seq=1 Ack=28337361 Win=2896384 Len=0 TSval=3180922755 TSecr=2380712235 S…
29665 20.970947232  192.168.109.136  192.168.10.1     FTP-DATA  1514 FTP Data: 1448 bytes (PASV) (retr files/crab.mp4)
29666 20.970953517  192.168.10.1     192.168.109.136  TCP         78 [TCP Dup ACK 29660#3] 48176 → 40003 [ACK] Seq=1 Ack=28337361 Win=2896384 Len=0 TSval=3180922755 TSecr=2380712235 S…
29667 20.971065892  192.168.109.136  192.168.10.1     FTP-DATA  1514 FTP Data: 1448 bytes (PASV) (retr files/crab.mp4)
```

main.c

```c
#include "download.h"
#include <sys/time.h>
#include <time.h>

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

    if (argc != 2) {
        printf("Usage: %s ftp://[<user>:<password>@]<host>/<url-path>\n",
argv[0]);
        exit(-1);
    }

    struct Settings settings;
    memset(&settings, 0, sizeof(settings));
    struct timeval start, end;
    double time_used;

    if(parse_ftp_url(argv[1], &settings)){
        printf("Usage: %s ftp://[<user>:<password>@]<host>/<url-path>\n",
argv[0]);
        exit(-1);
    }

    printf("Starting download application\n"
           "  - User: %s\n"
           "  - Password: %s\n"
           "  - Host: %s\n"
           "  - Host name: %s\n"
           "  - ULR path: %s\n"
           "  - IP: %s\n",
           settings.user,
           settings.password,
           settings.host,
           settings.host_name,
           settings.url_path,
           settings.ip);

    /*Criar socket A, connectar ele, e verificar resposta do servidor*/
    int socket_A;
    if(establish_ftp_connection(settings.ip, SERVER_PORT, &socket_A)){
        exit(-1);
```

```
    }
    printf("[INFO] socket_A: %d\n", socket_A);

    if(login_ftp(socket_A, settings.user, settings.password)){
        clouse_connection(socket_A, NOT_CONNECTED);
        exit(-1);
    }

    char* data_ip = malloc(MAX_SIZE);       // ip to connect socket_B
    int data_port = 0;                      // port to connect socket_B
    if(enter_ftp_passive_mode(socket_A, data_ip, &data_port)){
        clouse_connection(socket_A, NOT_CONNECTED);
        exit(-1);
    }

    int socket_B;
    if(connect_socket(data_ip, data_port, &socket_B)){
        clouse_connection(socket_A, NOT_CONNECTED);
        exit(-1);
    }
    printf("[INFO] socket_B: %d\n", socket_B);

    // TODO: temos que ler algo depois de conectar o socket? (Eu verifiquei
tem nada la).

    if(gettimeofday(&start, NULL)) {
        perror("gettimeofday");
        if(clouse_connection(socket_A, socket_B)) exit(-1);
        exit(-1);
    }

    if(download_file(socket_A, socket_B, settings.url_path,
settings.filename)){
        if(clouse_connection(socket_A, socket_B)) exit(-1);
        exit(-1);
    }

    if(gettimeofday(&end, NULL)) {
        perror("gettimeofday");
        if(clouse_connection(socket_A, socket_B)) exit(-1);
        exit(-1);
    }
```

```
    if(clouse_connection(socket_A, socket_B)) exit(-1);

    time_used = (end.tv_sec - start.tv_sec) + (end.tv_usec - start.tv_usec)
/ 1e6;
    printf("Download %s completed in %.2f seconds\n", settings.filename,
time_used);

    return 0;
}
```

Download.h

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


#define TRUE 1
#define FALSE 0

#define MAX_SIZE 256
#define MAX_RESPONSE_SIZE 1024

#define SERVER_PORT 21 // TPF tranfer
#define NOT_CONNECTED -1 // while socket are not connected

#define h_addr h_addr_list[0]   // The first address in h_addr_list.

#define CODE_220 220 // Response code for "Service ready for new user."
#define CODE_331 331 // Response code for "User name okay, need password."
#define CODE_230 230 // Response code for "User logged in, proceed."
#define CODE_227 227 // Response code for "Entering Passive Mode
(h1,h2,h3,h4,p1,p2)."
#define CODE_150 150 // Response code for "File status okay; about to open
data connection."
```

```
#define CODE_125 125 // Response code for "Data connection already open."
#define CODE_226 226 // Response code for "Closing data connection."
#define CODE_221 221 // Response code for "Service closing control
connection."

struct Settings {
   char user[MAX_SIZE];
   char password[MAX_SIZE];
   char host[MAX_SIZE];
   char host_name[MAX_SIZE];    // oficial host name.
   char url_path[MAX_SIZE];
   char ip[MAX_SIZE];
   char filename[MAX_SIZE];
};

enum state{
   START,
   CODE,    // state of receive code
   MESSAGE,    // state of receive message
   FEUP_MOMENT, // case quando pqp tem 200-text
   STOP
};

// Parcer of text (getip.c)
int parse_ftp_url(const char *text, struct Settings *settings);

// Create socket and connect it
int connect_socket(const char *IP, const int port, int *socket_fd);

// Create socket and connect it and verify response the server
int establish_ftp_connection(const char *IP, const int port, int
*socket_fd);

// Leia de socket_fd o response_code e response_message
int read_ftp_response(const int socket_fd, char* response_message, int*
response_code);

// Envia commando ao servidor
int send_ftp_command(const int socket_fd, const char* command);

// Faz Login ao servidor (USER anonymous; PASS anonymous)
```

```
int login_ftp(const int socket_fd, const char* username, const char*
password);

// send command pasv and cacl the data_port
int enter_ftp_passive_mode(const int socket_fd, char* data_ip, int*
data_port);

// Baixa o arquivo.
int download_file(const int socket_fd_A, const int socket_fd_B, const char*
url_path, const char* filename);

// Fecha a conexao do A, e B se foram abertas
int clouse_connection(const int socket_fd_A, const int socket_fd_B);

// Funcao auxiliar que faz free()
void free_resources(char* buf1, char* buf2, char* buf3);
```

Download.c

```
#include "download.h"

int parse_ftp_url(const char *text, struct Settings *settings){

   // Check if URL contains user and password info
   const char *at_sign = strchr(text, '@');
   int result;
   if(at_sign){
       // URL contains user and possibly a password
       char user_pass[512];
       result = sscanf(text, "ftp://%511[^@]@%255[^/]/%255[^\n]",
                               user_pass,
                               settings->host,
                               settings->url_path);

       if (result != 3) return -1;

       char *colon = strchr(user_pass, ':');
       if (colon) {
           *colon = '\0';
           strncpy(settings->user, user_pass, MAX_SIZE - 1);
           strncpy(settings->password, colon + 1, MAX_SIZE - 1);
       } else {
```

```
            return -1;
        }
    } else {
        // URL does not contain user and password, parse accordingly
        result = sscanf(text, "ftp://%255[^/]/%255[^\n]",
                                settings->host,
                                settings->url_path);

        if (result != 2) return -1;

        strncpy(settings->user, "anonymous", MAX_SIZE - 1);
        strncpy(settings->password, "anonymous", MAX_SIZE - 1);
    }

    if(strlen(settings->host) == 0) return -1;
    struct hostent *h;
    if((h = gethostbyname(settings->host)) == NULL){
        herror("gethostbyname()");
        return -1;
    }

    /*save the ip*/
    strncpy(settings->ip, inet_ntoa(*((struct in_addr *) h->h_addr)),
MAX_SIZE - 1);
    strncpy(settings->host_name, h->h_name, MAX_SIZE - 1);

    /*save the filename*/
    const char *last_slash = strrchr(settings->url_path, '/');
    if (last_slash != NULL) {
        strncpy(settings->filename, last_slash + 1, MAX_SIZE - 1);
    } else {
        strncpy(settings->filename, settings->url_path, MAX_SIZE - 1);
    }
    settings->filename[MAX_SIZE - 1] = '\0';

    return EXIT_SUCCESS;
}

int connect_socket(const char *IP, const int port, int *socket_fd){
    if(IP == NULL || socket_fd == NULL) return -1;

    int sockfd;
```

```
    struct sockaddr_in server_addr;

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

    /*open a TCP socket*/
    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        perror("socket()");
        return -1;
    }
    /*connect to the server*/
    if (connect(sockfd,
                (struct sockaddr *) &server_addr,
                sizeof(server_addr)) < 0) {
        perror("connect()");
        return -1;
    }

    *socket_fd = sockfd;

    return EXIT_SUCCESS;
}

int establish_ftp_connection(const char *IP, const int port, int
*socket_fd){

    if(connect_socket(IP, port, socket_fd)) return -1;

    /*read response from server*/
    char* response_buffer = malloc(MAX_RESPONSE_SIZE);
    int ftp_response_code = 0;

    if(read_ftp_response((*socket_fd), response_buffer,
&ftp_response_code)){
        printf("ERROR failed read response message\n");
        free(response_buffer);
        return -1;
```

```
    }

    if(ftp_response_code != CODE_220){
        printf("[ERROR] failed onnection to %s\n", IP);
        return -1;
    }
    free(response_buffer);
    return EXIT_SUCCESS;
}

int read_ftp_response(const int socket_fd, char* response_buffer, int*
response_code){
    if(response_buffer == NULL || response_code == NULL) return -1;

    enum state state = START;
    int indx = 0;
    *response_code = 0;
    int prev_code = 0;

    while(state != STOP){
        char byte = 0;
        int read_status = read(socket_fd, &byte, sizeof(byte));

        if(read_status == -1) return -1;

        // printf("flag: %d; byte: %d; byte: %c\n", read_status, byte,
byte);

        switch (state)
        {
        case START:
            state = CODE;
        case CODE:
            if(read_status == 0 || byte == '\n') state = STOP;
            else if(read_status == 1 && (byte >= 48 && byte <= 57)){    //
case if is digit
                *response_code += byte - 48;    // Transform ASCII to
decimal number
                *response_code *= 10;           // Left shift
            }
            else if(read_status == 1 && byte == ' ') state = MESSAGE;
            else if(read_status == 1 && byte == '-') state = FEUP_MOMENT;
```

```
            break;
        case MESSAGE:
            if(read_status == 0 || byte == '\n'){
                state = STOP;
                response_buffer[indx] = '\0';
            }
            else if(read_status == 1){
                response_buffer[indx++] = byte;
            }
            break;
        case FEUP_MOMENT:
            if(byte == '\n'){
                state = CODE;
                if(prev_code != 0 && prev_code != *response_code) return -1;
// On new line not the same code.

                prev_code = *response_code;
                *response_code = 0;
            }
            response_buffer[indx++] = byte;
            break;
        default:
            state = START;
            break;
        }
    }

    *response_code /= 10;

    printf("[INFO] [%d] ", *response_code);
    printf("Message:\n%s\n\n", response_buffer);
    return EXIT_SUCCESS;
}

int send_ftp_command(const int socket_fd, const char* command){
    size_t bytes = write(socket_fd, command, strlen(command));
    if(bytes < 0){
        perror("[ERROR] filed sending command to FTP server\n");
        return -1;
    }
    return EXIT_SUCCESS;
}
```

```c
int login_ftp(const int socket_fd, const char* username, const char*
password){
    if(username == NULL || password == NULL) return -1;

    char *command = malloc(MAX_RESPONSE_SIZE);
    char *response = malloc(MAX_RESPONSE_SIZE);
    int response_code = 0;

    // Send USER command
    snprintf(command, MAX_RESPONSE_SIZE, "USER %s\r\n", username);
    if(send_ftp_command(socket_fd, command)){
        free_resources(response, command, NULL);
        return -1;
    }

    if(read_ftp_response(socket_fd, response, &response_code)){
        free_resources(response, command, NULL);
        return -1;
    }

    if(response_code != CODE_331){
        printf("[ERROR] login failed with user: %s\n", username);
        free_resources(response, command, NULL);
        return -1;
    }

    // Send PASS command
    snprintf(command, MAX_RESPONSE_SIZE, "PASS %s\r\n", password);
    if(send_ftp_command(socket_fd, command)){
        free_resources(response, command, NULL);
        return -1;
    }

    if(read_ftp_response(socket_fd, response, &response_code)){
        free_resources(response, command, NULL);
        return -1;
    }

    if(response_code != CODE_230){
        printf("[ERROR] login failed with password: %s\n", password);
        free_resources(response, command, NULL);
```

```
        return -1;
    }

    free_resources(response, command, NULL);

    // printf("[INFO] FTP login successful\n");
    return EXIT_SUCCESS;
}

int enter_ftp_passive_mode(const int socket_fd, char* data_ip, int*
data_port){
    if(data_ip == NULL || data_port == NULL) return -1;

    char *pasv_command = "pasv\r\n";
    char *response = malloc(MAX_RESPONSE_SIZE);
    int response_code = 0;

    if(send_ftp_command(socket_fd, pasv_command)) {
        free(response);
        return -1;
    }

    if(read_ftp_response(socket_fd, response, &response_code)){
        free(response);
        return -1;
    }

    if(response_code != CODE_227){
        printf("[ERROR] failed pasv command\n");
        free(response);
        return -1;
    }

    int ip1, ip2, ip3, ip4, port1, port2;
    int result = sscanf(response,
                        "Entering Passive Mode (%d,%d,%d,%d,%d,%d)",
                            &ip1, &ip2, &ip3, &ip4, &port1, &port2);

    if(result != 6){
        printf("[ERROR] response pasv\n");
        return -1;
    }
```

```
    snprintf(data_ip, MAX_SIZE, "%d.%d.%d.%d", ip1, ip2, ip3, ip4);

    *data_port = (port1 << 8) + port2;  // port1 * 256 + port2
    free(response);
    return EXIT_SUCCESS;
}

int download_file(const int socket_fd_A, const int socket_fd_B, const char*
url_path, const char* filename){

    char *command = malloc(MAX_RESPONSE_SIZE);
    char *response = malloc(MAX_RESPONSE_SIZE);
    int response_code = 0;

    // Send retr command.
    snprintf(command, MAX_RESPONSE_SIZE, "retr %s\r\n", url_path);
    if(send_ftp_command(socket_fd_A, command) < 0){
        free_resources(response, command, NULL);
        return -1;
    }

    if(read_ftp_response(socket_fd_A, response, &response_code) < 0){
        free_resources(response, command, NULL);
        return -1;
    }

    /*if(response_code != CODE_150 && response_code != CODE_125){
        printf("[ERROR] retr command with URL path: %s\n", url_path);
        free_resources(response, command, NULL);
        return -1;
    }*/

    if(response_code / 100 != 1){
        printf("[ERROR] retr command with URL path: %s\n", url_path);
        free_resources(response, command, NULL);
        return -1;
    }

    // Download file.
    FILE *file = fopen(filename, "wb");
```

```
    if(file == NULL){
        printf("[ERROR] open file\n");
        return -1;
    }

    char *buf = malloc(MAX_SIZE);
    int bytes_read = 0;

    while( (bytes_read = read(socket_fd_B, buf, MAX_SIZE)) > 0 ){
        if(fwrite(buf, bytes_read, 1, file) < 0){
            free_resources(buf, response, command);
            return -1;
        }
    }
    fclose(file);

    // Check if its was finished.

    do{
        if(read_ftp_response(socket_fd_A, response, &response_code) < 0){
            free_resources(buf, response, command);
            return -1;
        }

        if(response_code != CODE_226){
            printf("[ERROR] transfer was not complete\n");
            free_resources(buf, response, command);
            return -1;
        }
    } while (response_code / 100 < 2);

    free_resources(buf, response, command);
    return EXIT_SUCCESS;
}

int clouse_connection(const int socket_fd_A, const int socket_fd_B){

    char *pasv_command = "quit\r\n";
    char *response = malloc(MAX_RESPONSE_SIZE);
    int response_code = 0;

    if(send_ftp_command(socket_fd_A, pasv_command)) {
```

```
        free(response);
        return -1;
    }

    if(read_ftp_response(socket_fd_A, response, &response_code)){
        free(response);
        return -1;
    }

    if(response_code != CODE_221){
        printf("[ERROR] failed quit command\n");
        free(response);
        return -1;
    }

    if (close(socket_fd_A)<0) {
        perror("close()");
        return -1;
    }
    if (socket_fd_B != NOT_CONNECTED && close(socket_fd_B)) {
        perror("close()");
        return -1;
    }
    return EXIT_SUCCESS;
}

void free_resources(char* buf1, char* buf2, char* buf3){
    free(buf1);
    free(buf2);
    free(buf3);
}
```

Makefile

```
# Makefile to build the project
# NOTE: This file must not be changed.

# Parameters
CC = gcc
CFLAGS = -Wall

SRC = src/
```

```
INCLUDE = include/
BIN = bin/

# We can change this URL.
URL = ftp://rcom:rcom@netlab1.fe.up.pt/pipe.txt

# ftp://netlab1.fe.up.pt/pub.txt

# ftp://rcom:rcom@netlab1.fe.up.pt/pipe.txt
# ftp://rcom:rcom@netlab1.fe.up.pt/files/pic1.jpg
# ftp://rcom:rcom@netlab1.fe.up.pt/files/pic2.png
# ftp://rcom:rcom@netlab1.fe.up.pt/files/crab.mp4

# ftp://ftp.up.pt/pub/kodi/timestamp.txt
# ftp://ftp.up.pt/pub/gnu/emacs/elisp-manual-21-2.8.tar.gz
# ftp://ftp.up.pt/debian/ls-lR.gz

# ftp://demo:password@test.rebex.net/readme.txt

# ftp://anonymous:anonymous@ftp.bit.nl/speedtest/100mb.bin


# Targets
.PHONY: all
all: $(BIN)/download

$(BIN)/download: main.c $(SRC)/*.c
	$(CC) $(CFLAGS) -o $@ $^ -I$(INCLUDE)

.PHONY: run
run: $(BIN)/download
	./$(BIN)/download $(URL)

.PHONY: clean
clean:
	rm -f $(BIN)/download
```

Estrutura da pasta

```
.
├── bin
│   └── download
```

```
├── include
│   └── download.h
├── main.c
├── Makefile
└── src
    └── download.c
```