Redes de Computadores

Relatório 2º Trabalho

Protocolo de Ligação de Dados

Turma 3LEIC06

- Diogo Silva up2021004794

- José Santos up202108673

Sumário

Para a UC de Redes de Computadores, criamos um projeto voltado para desenvolver um programa de download e explorar como configurar e usar uma rede de computadores.

Introdução

O projeto foi dividido em duas etapas distintas: na primeira, desenvolvemos um programa de download. Já na segunda parte, focamos na criação e configuração de uma rede de computadores, permitindo o acesso à internet para utilizar o programa e baixar arquivos online.

PARTE 1 – Desenvolvimento do programa de Download

O programa de download é desenvolvido em linguagem C e opera por meio do protocolo FTP para baixar arquivos. O funcionamento do programa ocorre em etapas sequenciais, descritas a seguir:

- Analisar o Url
- Criar um socket
- Fazer a autenticação com o FTP
- Entrar em modo passivo
- Criar outro socket
- Solicitar o ficheiro do FTP
- Fazer o download e guardar o ficheiro
- Fechar a ligação

Analisar o URL

Nesta fase, nossa tarefa consistia em analisar o URL em questão, uma vez que existiam dois possíveis formatos: podia ser do tipo 'ftp://<host>/<url-path>' ou do tipo 'ftp://[<user>:<password>@]<host>/<url-path>'. 'Host' representa o nome do servidor onde será estabelecida a comunicação; 'url-path' indica o caminho para acessar o arquivo desejado para download. Por fim, 'user' e 'password' são os campos destinados à autenticação no servidor (detalhes serão explicados posteriormente).

Criar um socket

A criação do socket é crucial porque é através dele que estabelecemos comunicação com o servidor. Criar um socket significa preparar uma interface de comunicação que viabiliza a troca de dados com o servidor.

Fazer a autenticação com FTP

Nesta etapa, o código realiza exclusivamente o processo de autenticação com o servidor FTP. Ele envia as credenciais de usuário ('user') e senha ('password') separadamente para autenticação no servidor, aguardando confirmação após cada etapa enviada.

Entrar em modo passivo

Nesta parte, o código é responsável por ativar o modo passivo na conexão com o servidor FTP. Ela envia o comando 'pasv' para o servidor, aguardando a resposta que confirma a ativação do modo passivo (resposta 227). Em seguida, extrai e calcula o IP e o número da porta a serem utilizados na conexão de dados.

Criar outro socket

A criação de um segundo socket é igualmente crucial, pois este será responsável pela transferência dos dados do arquivo do servidor para o cliente. O socketA controla a conexão e envia comandos, enquanto o socketB é dedicado exclusivamente à transferência de dados. Essa distinção possibilita uma troca mais fluida e eficiente de informações, garantindo uma transferência de arquivos sem interrupções.

Solicitar o ficheiro do FTP

A função *requestResource* neste contexto solicita explicitamente ao servidor FTP a disponibilidade e preparação para transferir o arquivo específico localizado no caminho (path) fornecido na URL. Essa ação é crucial para iniciar o processo de transferência do recurso desejado do servidor para o cliente, aguardando a confirmação do servidor para prosseguir com a transferência dos dados.

Fazer o download e guardar o ficheiro

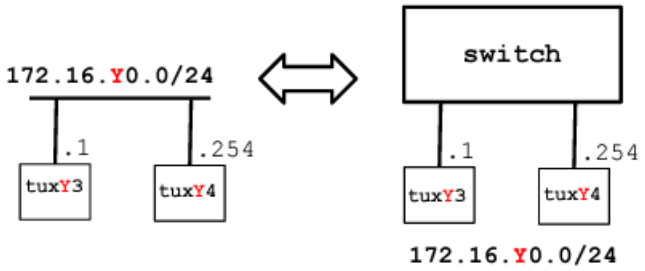
Nesta fase, o arquivo é baixado do servidor para o cliente por meio do segundo socket (socketB), garantindo que o arquivo seja corretamente transferido e salvo com seu nome original.

Fechar a ligação

Na etapa final, se tudo transcorreu sem problemas até então, ocorre o encerramento da conexão entre o cliente (nós) e o servidor. Essa fase é essencial para finalizar a comunicação de forma organizada e liberar os recursos associados à conexão, garantindo um fechamento adequado e eficiente entre as partes envolvidas.

PARTE 2 – Configuração e estudo de uma rede de computadores

Experiência 1 – Configurar uma rede IP



Pergunta 1: Quais são os comandos necessários para configurar esta experiência?

Os comandos necessários para configurar o ambiente descrito são no Tux43 "*ifconfig eth0 172.16.40.1/24"* e no Tux44 *"ifconfig eth0 172.16.40.254/24"*.

Pergunta 2: O que são os pacotes ARP e para que servem?

O Protocolo de Resolução de Endereços (ARP - Address Resolution Protocol) é utilizado para associar o endereço IP de uma máquina ao seu endereço físico, conhecido como MAC Address. Quando um computador em uma rede local tenta enviar pacotes para outro computador na mesma rede e não há entradas correspondentes na tabela ARP para o endereço IP de destino, um pacote ARP é enviado em modo de transmissão geral (broadcast) para todas as máquinas na rede. Esse procedimento tem o objetivo de identificar qual máquina possui o endereço MAC correspondente ao endereço IP do destinatário.

Posteriormente, a máquina de destino responde com um novo pacote ARP, fornecendo à máquina remetente o seu próprio endereço MAC. Com essa informação, a transmissão de pacotes entre as máquinas pode ocorrer de forma eficiente.

Pergunta 3: Quais são os endereços MAC e IP dos pacotes ARP e por quê?

No pacote ARP, serão incluídos os endereços IP tanto do Tux43 (emissor, 172.16.40.1) quanto do Tux44 (recetor, 172.16.40.254). Os endereços MAC enviados serão o do Tux43 (emissor, 00:21:5a:61:2f:13) e, como o endereço MAC do Tux recetor é desconhecido, será utilizado um valor genérico (00:00:00:00:00:00).

Exemplos podem ser vistos nas Figuras 1 e 2.

Pergunta 4: Que pacotes são gerados pelo comando ping?

O comando 'ping' começa enviando pacotes ARP para encontrar o endereço MAC do destinatário. Se necessário, a tabela ARP pode ser limpa com 'arp -d (ip)' para obter um novo endereço MAC. Em seguida, o 'ping' utiliza pacotes ICMP para testar a conectividade entre dispositivos, transmitindo solicitações e recebendo respostas de eco (echo request/reply).

Pergunta 5: Quais são os endereços MAC e IP dos pacotes de ping?

Do ping request temos do emissor o MAC (00:21:5a:61:2f:13) e o IP 172.16.40.1, do recetor o MAC (00:21:5a:c3:78:76) e o IP 172.16.40.254. (Figura 3)

Do ping reply temos do emissor o MAC (00:21:5a:c3:78:76) e o IP 172.16.40.254, do recetor o MAC (00:21:5a:61:2f:13) e o 172.16.40.1. (Figura 4)

Pergunta 6: Como se determina se uma trama Ethernet recebida é ARP, IP, ICMP?

Para identificar se uma trama é ARP, IP ou ICMP, é suficiente verificar a coluna 'Protocol' na captura.

Exemplos podem ser vistos nas Figuras 5 (ICMP) e 6 (ARP).

Pergunta 7: Como determinar o comprimento de uma trama recebida?

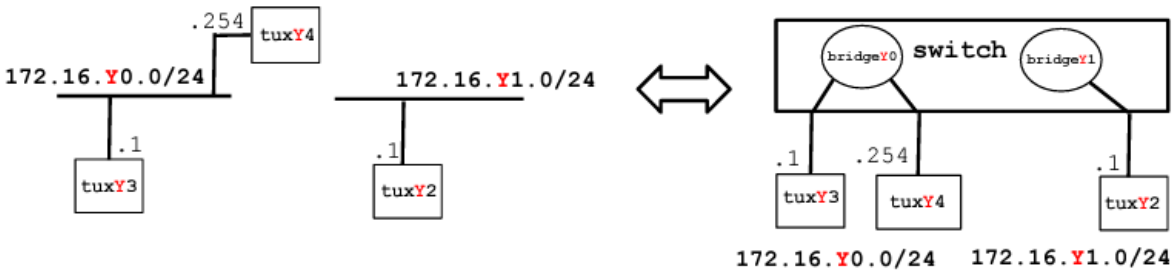
Para determinar o comprimento de uma trama recebida é suficiente verificar a coluna 'Length' na captura.

Exemplo pode ser visto na Figura 7.

Pergunta 8: O que é a interface de loopback e qual é a sua importância?

A interface de loopback é uma construção virtual que permite que um dispositivo receba e processe informações enviadas a si mesmo. Ela é utilizada tanto para verificar o funcionamento interno de um sistema quanto para testar e assegurar a configuração adequada de uma rede.

Experiência 2 – Implementar duas *bridges* num *switch*



Pergunta 1: Como configurar a bridgeY0?

Tendo o Tux43 e Tux44 ligados à porta 23 e 24 do switch, respetivamente.

Comandos para configurar a bridge:

Criar bridge40:

/interface bridge add name=bridge40

Remover as portas conectadas à default bridge do Tux43 e do Tux44:

/interface bridge port remove [find interface=ether23]

/interface bridge port remove [find interface=ether24]

Adicionar as portas do Tux43 e do Tux44 à bridge40:

/interface bridge port add bridge=bridge40 interface=ether23

/interface bridge port add bridge=bridge40 interface=ether24

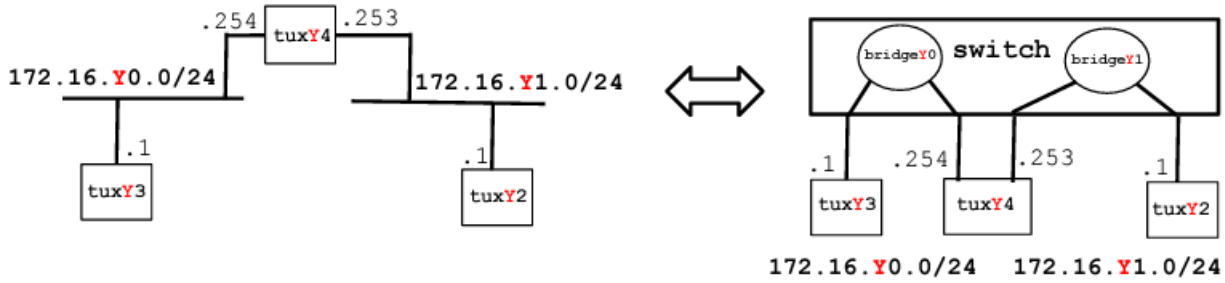
Pergunta 2: Quantos domínios de broadcast existem? Como se conclui isso a partir dos logs?

Existem dois domínios de broadcast, e ao realizar um ping de broadcast, somente as portas dentro de uma bridge específica são alcançadas.

Ao executar um ping do Tux43 para o endereço de broadcast (172.16.40.255), os pacotes são visíveis no Tux43 e Tux44 devido à sua presença na mesma bridge (Figuras 8 e 9). Infelizmente, perdemos as logs do Tux42 nesta experiência.

No entanto, quando é realizado um ping do Tux42 para o endereço de broadcast (172.16.41.255), os pacotes são vistos apenas no Tux42 (Figuras 10 e 11). Tux43 e Tux44 não recebem esses pacotes, pois estão em outro domínio de rede.

Experiência 3 – Configurar um *router* em Linux



Pergunta 1: Quais são os comandos necessários para configurar esta experiência?

Tendo o eth1 do Tux44 ligado à porta 20 do switch.

Configurar eth1 do Tux44:

ifconfig eth1 up

ifconfig eth1 172.16.41.253/24

Remover as portas da bridge default e adicionar à bridge que tem o Tux42 (bridge41):

/interface bridge port remove [find interface=ether20]

/interface bridge port add bridge=bridge41 interface=ether20

Ativar ip forwarding e desativar ICMP no Tux54:

sysctlnet.ipv4.ip_forward=1

sysctlnet.ipv4.icmp_echo_ignore_broadcasts=0

Configurar rotas nos Tux42 e Tux43 que permitam a comunicação com o Tux44, possibilitando que se alcancem mutuamente:

route add -net 172.16.40.0/24 gw 172.16.41.253

route add -net 172.16.41.0/24 gw 172.16.40.254

Pergunta 2: Que rotas existem nos tuxes? Qual o seu significado?

No Tux43, há uma rota para a bridge40, criada automaticamente, e uma rota anteriormente adicionada para a sub-rede 172.16.41.0/24 via gateway 172.16.40.254 (Tux44). Esta rota possibilita o envio de pacotes do Tux43 para o Tux42, que estão em bridges diferentes, sendo o Tux44 o ponto de conexão entre eles.

No Tux42, existe uma rota para a bridge41, criada automaticamente, e uma rota previamente configurada para a sub-rede 172.16.40.0/24 via gateway 172.16.40.253 (Tux44). Essa rota permite o envio de pacotes do Tux42 para o Tux43, ambos em bridges distintas, tendo o Tux44 como ponto intermediário para essa conexão.

Pergunta 3: Quais as informações que contém uma entrada das tabelas de encaminhamento?

Uma entrada típica em uma tabela de encaminhamento contém as seguintes informações:

- **Destino**: O endereço de destino para o qual o encaminhamento é direcionado, muitas vezes representado como um endereço IP ou um intervalo de endereços IP.
- **Máscara de sub-rede**: Indica a parte do endereço IP que define a rede, permitindo distinguir entre o endereço da rede e o endereço do host.
- **Gateway padrão**: O endereço IP do dispositivo que serve como ponto de acesso para alcançar redes fora daquela em que o dispositivo está localizado.
- **Interface de saída**: A interface de rede específica por onde o tráfego deve ser encaminhado para alcançar o destino especificado.
- **Métrica**: Valor numérico usado para determinar a preferência ou a prioridade entre rotas semelhantes.
- **Utilização**: Indica o número de vezes que essa rota específica foi usada para encaminhar pacotes.
- **Referências**: Representa o número de referências existentes para essa rota. Em alguns sistemas, essa informação pode não ser utilizada ou disponível.

Pergunta 4: Quais mensagens ARP e endereços MAC associados são observados e porquê?

Ao executar um ping do Tux43 para o Tux42, é iniciado um processo de troca de pacotes ARP para resolver os endereços MAC necessários para a comunicação na rede local. Nesse contexto, os pacotes ARP enviados e recebidos entre o Tux43 e o Tux44, que atua como gateway padrão, contêm apenas os endereços MAC desses dispositivos.

Essa limitação ocorre porque o Tux43, ao iniciar o processo de ARP, solicita ao Tux44 o endereço MAC associado ao endereço IP do Tux42. Entretanto, o Tux43 não possui o endereço MAC do Tux42 em sua tabela ARP. Consequentemente, os pacotes ARP refletem apenas os endereços MAC do emissor (Tux43) e do intermediário conhecido (Tux44), uma vez que é por meio do Tux44 que o Tux43 consegue alcançar o Tux42.

Dessa forma, devido à configuração da rota que direciona o tráfego através do Tux44 para atingir o Tux42, os pacotes ARP não incluem o endereço MAC do destino final (Tux42) durante a troca de informações entre o Tux43 e o Tux44.

Exemplo pode ser visto na Figura 12.

Pergunta 5: Que pacotes ICMP são observados e porquê?

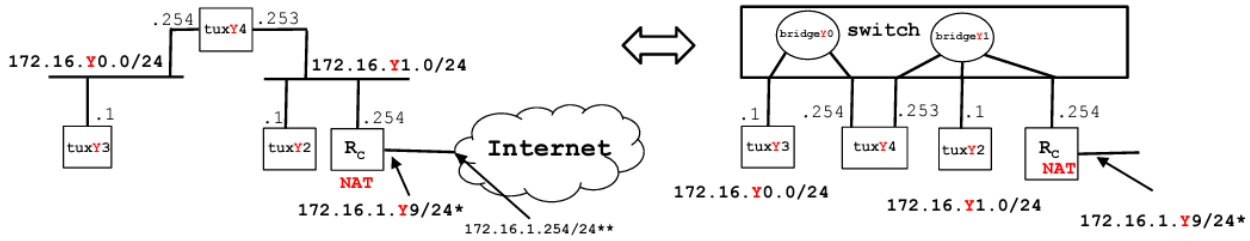
Todos os pacotes ICMP do tipo Request e Reply podem ser observados devido à configuração adequada das rotas, permitindo que todos os Tuxes sejam identificados entre si, conforme demonstrado na figura 13. Se a configuração não estivesse bem feita, os pacotes ICMP enviados teriam a mensagem “Host Unreachable”.

Pergunta 6: Quais são os endereços IP e MAC associados aos pacotes ICMP e porquê?

Quando enviamos pacotes do Tux43 para o Tux42, que não estão na mesma rede, não há uma conexão direta. Para os pacotes alcançarem o Tux42, eles precisam primeiro passar pelo Tux44, que está conectado a ambas as bridges (bridge40 e bridge41). Como resultado, os endereços IP do emissor serão do Tux43 (172.16.40.1) e do recetor serão do Tux42 (172.16.41.1). No entanto, os endereços MAC serão do Tux43 (00:21:5a:61:2f:13) para o emissor e do Tux44 (00:21:5a:c3:78:76) para o recetor, conforme observado na figura 14.

Quando o pacote enviado pelo Tux43 chega ao Tux44, este encaminha-o para o Tux42, mantendo o mesmo endereço IP, mas alterando apenas os endereços MAC. Assim, a interface eth1 do Tux44 passa a ser o emissor e o Tux42 o recetor do pacote.

Experiência 4 – Configurar um *router comercial* com NAT



Pergunta 1: Como configurar uma rota estática num router comercial?

Inicialmente, ainda com o cabo ligado a consola do switch, ligamos a porta eth2 do router a uma porta do switch, que na nossa experiênica foi ligada à porta 7. De seguida abrimos o GTK, e removemos a interface ether 7 de qualquer bridge a que esteja associada, e adicionamo-la à bridge41 com os comandos:

/interface bridge port remove [find interface=ether7];
/interface bridge port add bridge=bridge41 interface=ether7;

De seguida começamos por ligar a porta eth1 do router à porta 4.1 da régua e ligamos ainda o cabo S0 do Tux43 à porta identificada como "Router MTIK". De seguida, abrimos o GTKTerminal no Tux43, configuramos o baudrate para 115200 e fazemos login. Por segurança, resetamos todas as configurações feitas no router com o comando /system reset-configuration.

Agora que temos todas as condições reunidas, configuramos uma rota estática com o seguinte comando:

- /ip route add dst-address=172.16.40.0/24 gateway=172.16.41.253;

Pergunta 2: Quais são os caminhos percorridos pelos pacotes nos testes realizdos, e porquê?

Como a default gateway do Tux44 e do Tux42 é um ip que se encontra no router, 172.16.41.254, todos os pacotes destinados a redes que não 172.16.40.0/24 ou 172.16.41.0/24 são direcionados para o Router RC. Isto acontece porque a default gateway do Tux43 é o ip do Tux44, enquanto que a do Tux42 e a do Tux44 é o ip 172.16.41.254. Assim, todo o tráfego do Tux43 é encaminhado para o Tux44, que por sua vez é direcionado para o router.

No caso em que o ou o Tux42, ou o router queiram acessar a rede 172.16.40.0/24, eles fazem-no através de uma rota localizada na interface eth1 do Tux44, em que o gateway é 172.16.41.253.

Pergunta 3: Como configurar o NAT num router comercial?

Para ligar (ou desligar) a NAT num router corremos na consola do router o comando /ip firewall nat (enable/disable) 0.

Para adicionar regras à NAT, corremos o comando:

- /ip firewall nat add chain=srcnat action=masquerade out-interface=ether1;

Parâmetros do comando:

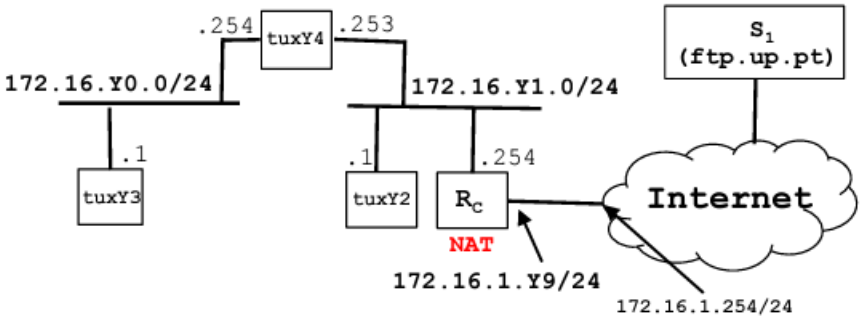
1. **chain=srcnat**: Especifica a chain na qual a regra de NAT será aplicada. Neste caso, srcnat refere-se à Source NAT (cadeira de origem), onde ocorre a tradução de endereços de origem.

2. **action=masquerade**: Define a ação a ser realizada pela regra de NAT. Neste caso, a ação é "masquerade", que é uma forma de fazer NAT dinâmico. Isso substitui o endereço de origem dos pacotes pelo endereço IP da interface de saída(out-interface)
3. **out-interface=ether1**: Especifica a interface de saída para a qual a regra de NAT se aplica. Aqui, "ether1" é o nome da interface da rede de saída. A regra masquerade será aplicada apenas aos pacotes que saem por esta interface, substituindo o endereço de origem deles.

Pergunta 4: O que é o NAT?

NAT signifca "Network Adress Translation" (Tradução de Endereço de Rede), e é uma técnica usada em routers e dispositivos de rede para mapear endereços IP entre redes diferentes. É usado em grande parte para permitir que vários dispositivos numa rede local partilhem o mesmo endereço IP público para aceder à Internet.

Experiência 5 – DNS



Pergunta 1: Como configurar o serviço de DNS num host?

Para configurar o serviço DNS (Domain Name Service) em um sistema Linux, é necessário editar o arquivo "resolv.conf" localizado no diretório /etc/ utilizando um editor de texto, como o nano. Adicionamos ao arquivo os IP's dos servidores DNS desejados, inserindo uma linha da seguinte maneira:

nameserver xxx.xxx.xxx.xxx (substituindo, pelo IP desejado)

Após adicionarmos os endereços IP desejados, guardamos o arquivo e reiniciamos o serviço DNS com o comando "**service network-manager restart**", para que as alterações sejam efetuadas. Agora, se tudo tiver sido feito corretamente, conseguimos acessar um site na internet, confirmando assim que configuramos o DNS corretamente.

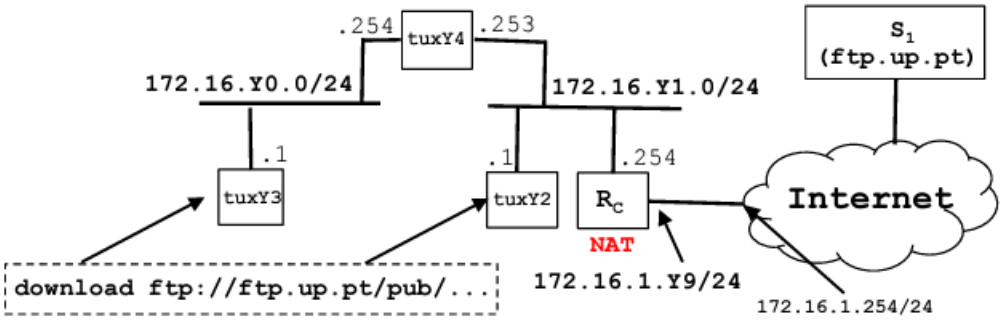
Pergunta 2: Quais pacotes são trocados pelo DNS e que informações são transportadas.

É enviado para o servidor um pacote com o nome do host que fica à espera que este retorne o seu endereço IP. O servidor recebe o pacote e verifica se o hostname pedido está registado. Se tal acontecer, o servidor retorna um pacote com o endereço IP correspondente, caso contrário o servidor consulta de forma recursiva outros servidores até encontrar o registo ou determinar que não pode ser resolvido.

Quando o servidor recebe a resposta, envia de volta um pacote para o host com o IP encontrado ou um código de erro, indicando que o nome não foi possível de ser resolvido.

Os pacotes DNS contém o nome do donímio solicitado, o tipo de registo, o classificador de recurso, o endereço IP correspondente ao nome do domínio(se encontrado) ou o código de erro, em caso negativo, e o tempo de vida do registo.

Experiência 6 – Conexões TCP



Pergunta 1: Quantas conexões TCP são abertas pela nossa aplicação FTP?

A nossa aplicação TCP abre duas ligações, em que uma é feita ao entrar em contacto com o servidor, para efetuar o envio e receção de comandos de modo a preparar a transferência(efetuar login, entrar em modo passivo e pedir o ficheiro), enquanto que a outra é dedicada á transferência de dados do ficheiro.

Pergunta 2: Em qual conexão é transportada a informação de controlo FTP?

A informação de controlo FTP é transportada através da primeira ligação.

Pergunta 3: Quais são as fases de uma conexão TCP?

Uma conexão TCP pode ser divida em 3 fases: A primeira fase para estabelecer a ligação, a segunda para transmitir dados e a terceira para encerrar a conexão.

Pergunta 4: Como funciona o mecanismo ARP TCP? Quais são os campos TCP relevantes? Que informações podem ser observadas nos registos?

O protocolo TCP gere erros na transmissão de dados usando o método ARP que opera por meio de uma técnica conhecida como “janela deslizante”. Esse mecanismo envolve o uso de números sequenciais para identificar as tramas a serem enviadas, os números ACK para confirmar a correta receção das tramas, incluindo as anteriores, e o tamanho da janela que controla a quantidade de dados na rede. Esses valores referenciados como SEQ, ACK e WS podem ser encontrados nos logs (Figura 15).

Pergunta 5: Como funciona o mecanismo de controlo de congestão TCP? Quais são os campos relevantes. Como evolui a taxa de transferência da conexão de dados ao longo do tempo? Está de acordo com o mecanismo de controlo de congestão TCP?

O mecanismo de controlo de congestão TCP é fundamentado nos números de ACK recebidos durante a transmissão dos pacotes, os quais desempenham o papel de um relógio de origem na transmissão. Para regular a janela de transmissão de pacotes, levando em consideração a congestão da conexão, é essencial introduzir uma nova variável por conexão chamada “CongestionWindow”.

A gestão dessa variável envolve ajustes incrementais quando a congestão diminui e decrementos quando a congestão aumenta. Geralmente, isso é realizado por meio de um algoritmo conhecido como Aumento Aditivo/Diminuição Multiplicativa. Seguindo esse algoritmo, a cada Round Trip Time, a CongestionWindow é aumentada em 1. Quando é detectada a perda de um pacote, seu valor é reduzido pela metade.

Adicionalmente, pode ocorrer uma fase de "slow start" no início da conexão, cujo propósito é determinar rapidamente a capacidade disponível. Durante essa fase, a CongestionWindow é inicialmente configurada com um valor menor e, à medida que a transmissão avança sem perdas significativas, é aumentada exponencialmente para explorar eficientemente a largura de banda disponível. Essas estratégias colaborativas visam otimizar o desempenho da transmissão TCP, adaptando-se dinamicamente às condições da rede.

Ao iniciarmos o download vemos que logo no início aumenta exponencialmente, estabilizando depois a velocidade, aumentando apenas ligeiramente ao longo do tempo. (Figura 16)

Pergunta 6: A taxa de transferência de conexões de dados TCP é perturbada pela aparição de uma segunda conexão TCP? Como?

Sim, a taxa de transferência de conexões de dados TCP pode ser afetada pela aparição de uma segunda conexão TCP devido à competição por largura de banda. Conexões concorrentes podem levar a congestionamento e, consequentemente, a uma redução na taxa de transferência de ambas as conexões devido ao compartilhamento do mesmo canal de comunicação.

Conclusões

Este projeto foi uma experiência rica e enriquecedora, que nos permitiu aprofundar os nossos conhecimentos sobre redes de computadores. Trabalhamos juntos para desenvolver um programa de download e configurar uma rede de computadores, o que nos permitiu entender melhor como funcionam as redes e como elas são fundamentais para a comunicação e a transferência de dados.

Consideramos que ambos os membros do grupo contribuíram igualmente para o sucesso do projeto. A abordagem gradual e a repetição da experiência a cada aula nos ajudaram a entender melhor o funcionamento da rede. Finalmente, achamos que este projeto foi extremamente enriquecedor e que nos deu as bases necessárias para futuramente aprofundarmos nossos conhecimentos em redes.

Anexos

Figuras

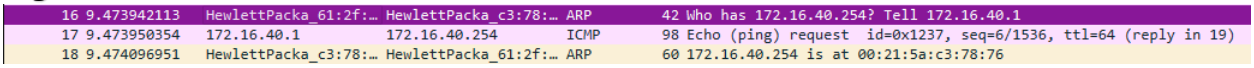


Figura 1 - ARP Request and Reply

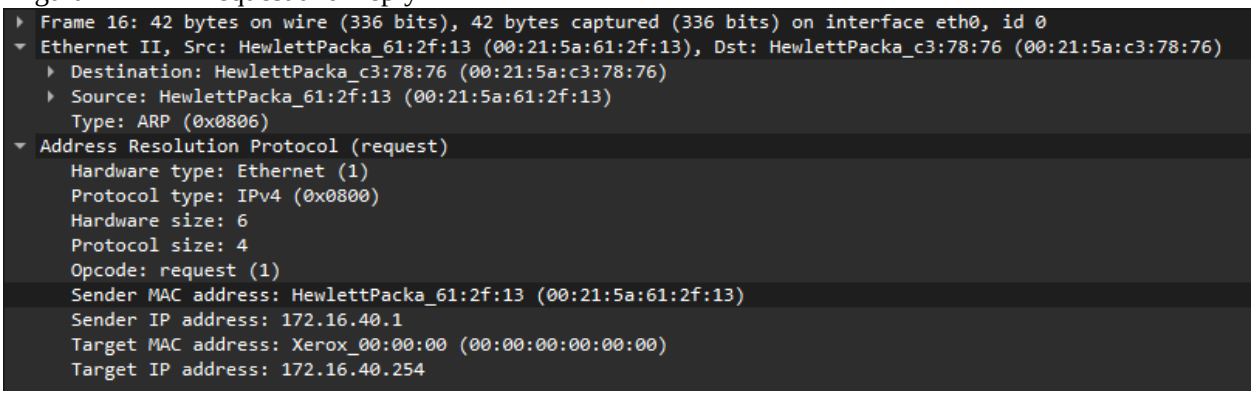


Figura 2 - ARP Request info

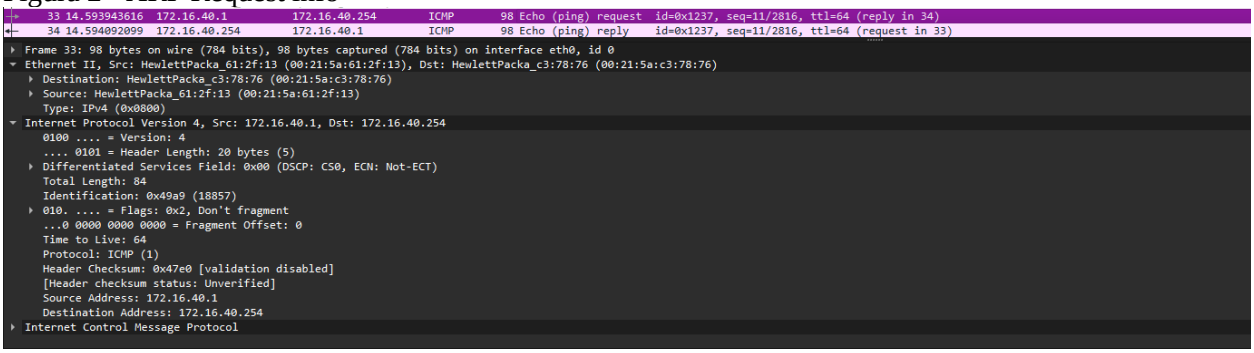


Figura 3 - ARP Request info

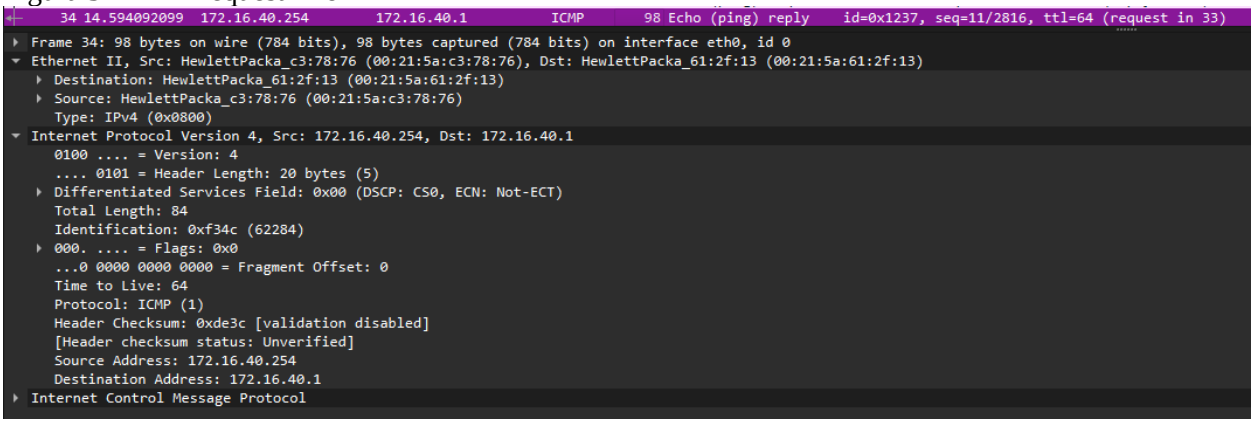


Figura 4 - ARP Reply info

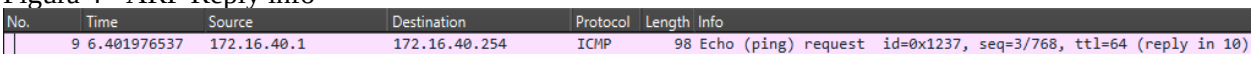


Figura 5 – Trama ICMP

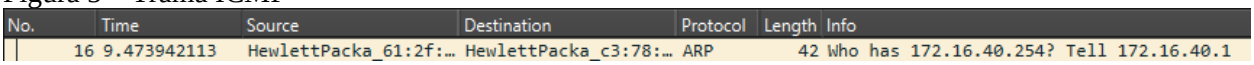


Figura 6 – Trama ARP

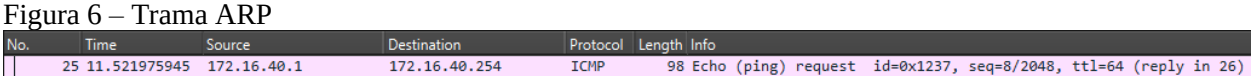
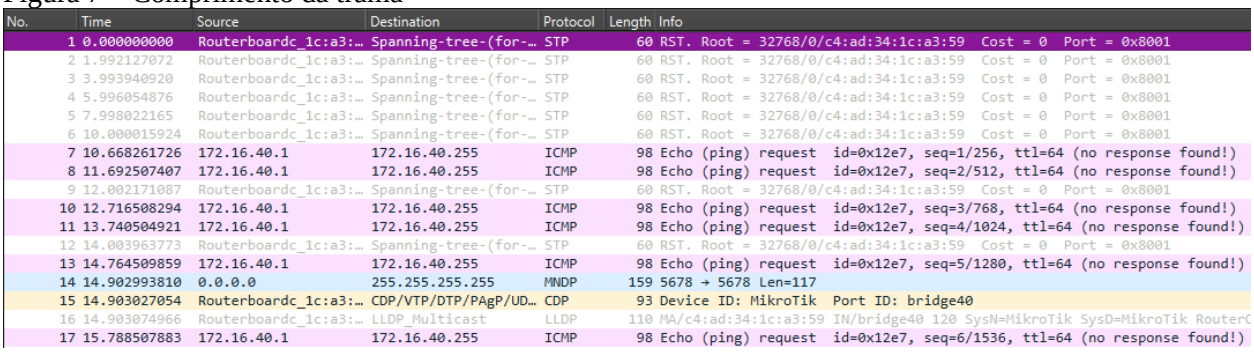
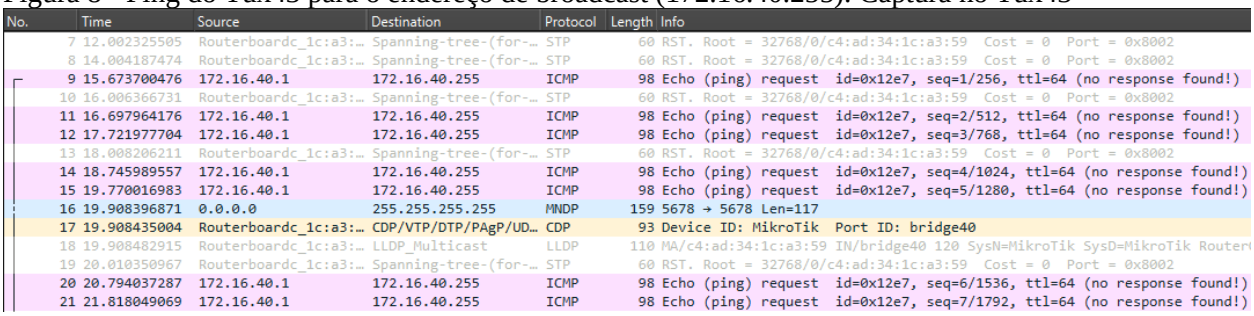


Figura 7 – Comprimento da trama



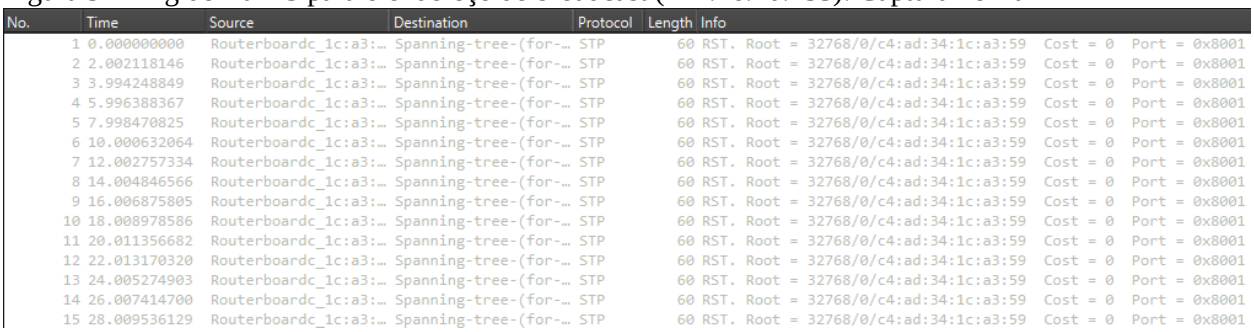
No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	Routerboardc_1c:a3:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:a3:59 Cost = 0 Port = 0x8001
2	1.992127072	Routerboardc_1c:a3:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:a3:59 Cost = 0 Port = 0x8001
3	3.993940920	Routerboardc_1c:a3:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:a3:59 Cost = 0 Port = 0x8001
4	5.996054876	Routerboardc_1c:a3:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:a3:59 Cost = 0 Port = 0x8001
5	7.998022165	Routerboardc_1c:a3:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:a3:59 Cost = 0 Port = 0x8001
6	10.000015924	Routerboardc_1c:a3:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:a3:59 Cost = 0 Port = 0x8001
7	10.668261726	172.16.40.1	172.16.40.255	ICMP	98	Echo (ping) request id=0x12e7, seq=1/256, ttl=64 (no response found!)
8	11.692507407	172.16.40.1	172.16.40.255	ICMP	98	Echo (ping) request id=0x12e7, seq=2/512, ttl=64 (no response found!)
9	12.002171087	Routerboardc_1c:a3:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:a3:59 Cost = 0 Port = 0x8001
10	12.716508294	172.16.40.1	172.16.40.255	ICMP	98	Echo (ping) request id=0x12e7, seq=3/768, ttl=64 (no response found!)
11	13.740504921	172.16.40.1	172.16.40.255	ICMP	98	Echo (ping) request id=0x12e7, seq=4/1024, ttl=64 (no response found!)
12	14.003963773	Routerboardc_1c:a3:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:a3:59 Cost = 0 Port = 0x8001
13	14.764509859	172.16.40.1	172.16.40.255	ICMP	98	Echo (ping) request id=0x12e7, seq=5/1280, ttl=64 (no response found!)
14	14.902993810	0.0.0.0	255.255.255.255	MNDP	159	5678 → 5678 Len=117
15	14.903027054	Routerboardc_1c:a3:…	CDP/VTP/DTP/PAgP/UD…	CDP	93	Device ID: MikroTik Port ID: bridge40
16	14.903074966	Routerboardc_1c:a3:…	LLDP_Multicast	LLDP	110	MA/c4:ad:34:1c:a3:59 IN/bridge40 120 SysN=MikroTik SysD=MikroTik RouterC
17	15.788507883	172.16.40.1	172.16.40.255	ICMP	98	Echo (ping) request id=0x12e7, seq=6/1536, ttl=64 (no response found!)

Figura 8 - Ping do Tux43 para o endereço de broadcast (172.16.40.255). Captura no Tux43



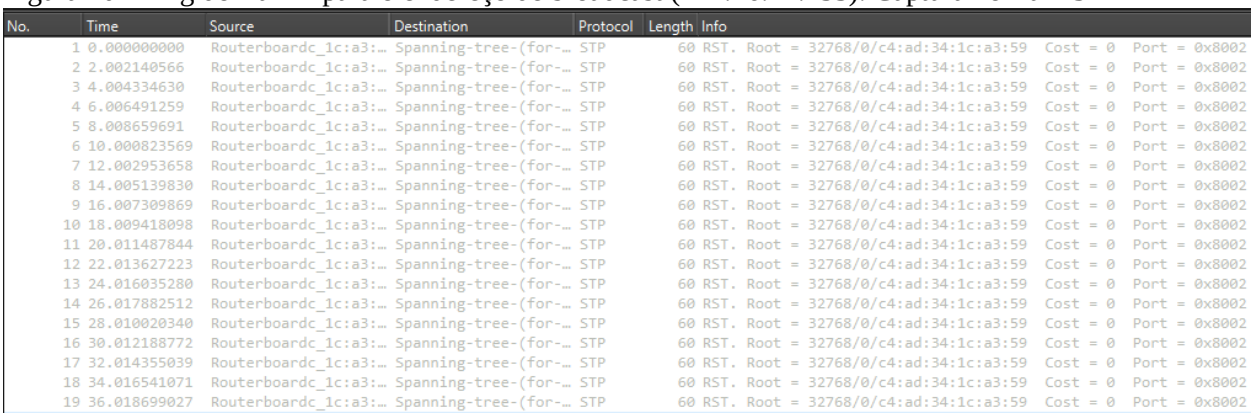
No.	Time	Source	Destination	Protocol	Length	Info
7	12.002325505	Routerboardc_1c:a3:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:a3:59 Cost = 0 Port = 0x8002
8	14.004187474	Routerboardc_1c:a3:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:a3:59 Cost = 0 Port = 0x8002
9	15.673700476	172.16.40.1	172.16.40.255	ICMP	98	Echo (ping) request id=0x12e7, seq=1/256, ttl=64 (no response found!)
10	16.006366731	Routerboardc_1c:a3:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:a3:59 Cost = 0 Port = 0x8002
11	16.697964176	172.16.40.1	172.16.40.255	ICMP	98	Echo (ping) request id=0x12e7, seq=2/512, ttl=64 (no response found!)
12	17.721977704	172.16.40.1	172.16.40.255	ICMP	98	Echo (ping) request id=0x12e7, seq=3/768, ttl=64 (no response found!)
13	18.008206211	Routerboardc_1c:a3:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:a3:59 Cost = 0 Port = 0x8002
14	18.745989557	172.16.40.1	172.16.40.255	ICMP	98	Echo (ping) request id=0x12e7, seq=4/1024, ttl=64 (no response found!)
15	19.770016983	172.16.40.1	172.16.40.255	ICMP	98	Echo (ping) request id=0x12e7, seq=5/1280, ttl=64 (no response found!)
16	19.908396871	0.0.0.0	255.255.255.255	MNDP	159	5678 → 5678 Len=117
17	19.908435004	Routerboardc_1c:a3:…	CDP/VTP/DTP/PAgP/UD…	CDP	93	Device ID: MikroTik Port ID: bridge40
18	19.908482915	Routerboardc_1c:a3:…	LLDP_Multicast	LLDP	110	MA/c4:ad:34:1c:a3:59 IN/bridge40 120 SysN=MikroTik SysD=MikroTik Router
19	20.010350967	Routerboardc_1c:a3:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:a3:59 Cost = 0 Port = 0x8002
20	20.794037287	172.16.40.1	172.16.40.255	ICMP	98	Echo (ping) request id=0x12e7, seq=6/1536, ttl=64 (no response found!)
21	21.818049069	172.16.40.1	172.16.40.255	ICMP	98	Echo (ping) request id=0x12e7, seq=7/1792, ttl=64 (no response found!)

Figura 9 - Ping do Tux43 para o endereço de broadcast (172.16.40.255). Captura no Tux44



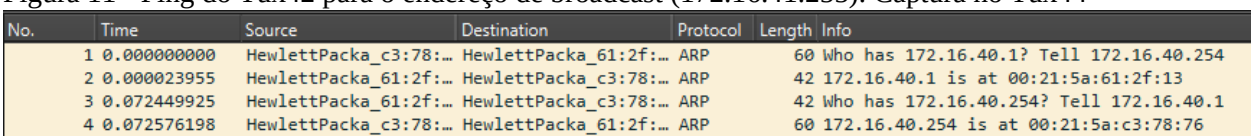
No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	Routerboardc_1c:a3:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:a3:59 Cost = 0 Port = 0x8001
2	2.002118146	Routerboardc_1c:a3:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:a3:59 Cost = 0 Port = 0x8001
3	3.994248849	Routerboardc_1c:a3:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:a3:59 Cost = 0 Port = 0x8001
4	5.996388367	Routerboardc_1c:a3:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:a3:59 Cost = 0 Port = 0x8001
5	7.998470825	Routerboardc_1c:a3:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:a3:59 Cost = 0 Port = 0x8001
6	10.000632064	Routerboardc_1c:a3:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:a3:59 Cost = 0 Port = 0x8001
7	12.002757334	Routerboardc_1c:a3:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:a3:59 Cost = 0 Port = 0x8001
8	14.004846566	Routerboardc_1c:a3:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:a3:59 Cost = 0 Port = 0x8001
9	16.006875805	Routerboardc_1c:a3:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:a3:59 Cost = 0 Port = 0x8001
10	18.008978586	Routerboardc_1c:a3:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:a3:59 Cost = 0 Port = 0x8001
11	20.011356682	Routerboardc_1c:a3:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:a3:59 Cost = 0 Port = 0x8001
12	22.013170320	Routerboardc_1c:a3:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:a3:59 Cost = 0 Port = 0x8001
13	24.005274903	Routerboardc_1c:a3:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:a3:59 Cost = 0 Port = 0x8001
14	26.007414700	Routerboardc_1c:a3:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:a3:59 Cost = 0 Port = 0x8001
15	28.009536129	Routerboardc_1c:a3:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:a3:59 Cost = 0 Port = 0x8001

Figura 10 - Ping do Tux42 para o endereço de broadcast (172.16.41.255). Captura no Tux43



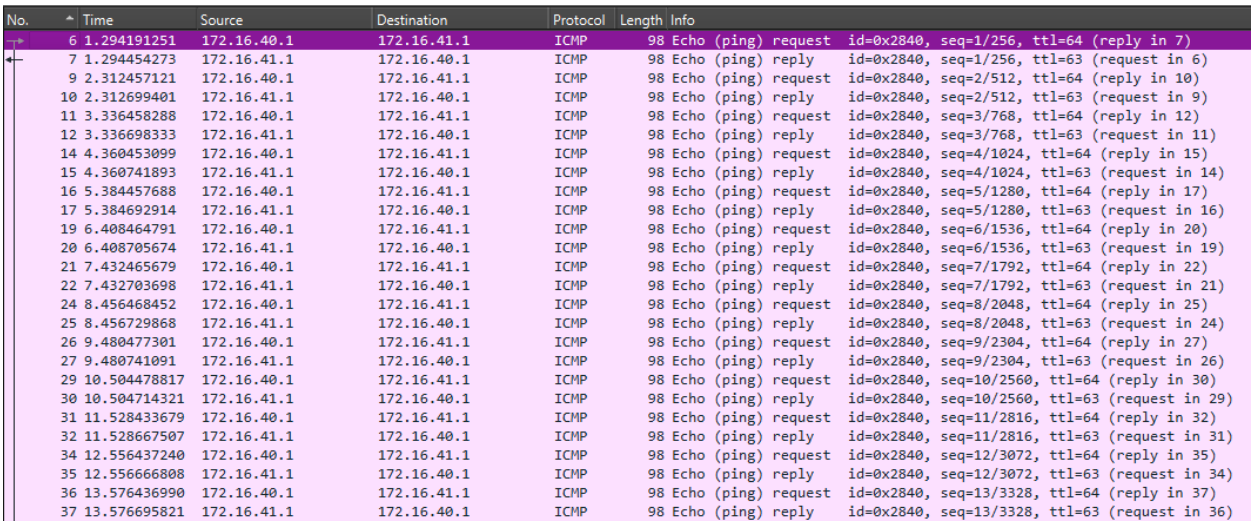
No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	Routerboardc_1c:a3:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:a3:59 Cost = 0 Port = 0x8002
2	2.002140566	Routerboardc_1c:a3:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:a3:59 Cost = 0 Port = 0x8002
3	4.004334630	Routerboardc_1c:a3:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:a3:59 Cost = 0 Port = 0x8002
4	6.006491259	Routerboardc_1c:a3:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:a3:59 Cost = 0 Port = 0x8002
5	8.008659691	Routerboardc_1c:a3:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:a3:59 Cost = 0 Port = 0x8002
6	10.000823569	Routerboardc_1c:a3:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:a3:59 Cost = 0 Port = 0x8002
7	12.002953658	Routerboardc_1c:a3:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:a3:59 Cost = 0 Port = 0x8002
8	14.005139830	Routerboardc_1c:a3:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:a3:59 Cost = 0 Port = 0x8002
9	16.007309869	Routerboardc_1c:a3:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:a3:59 Cost = 0 Port = 0x8002
10	18.009418098	Routerboardc_1c:a3:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:a3:59 Cost = 0 Port = 0x8002
11	20.011487844	Routerboardc_1c:a3:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:a3:59 Cost = 0 Port = 0x8002
12	22.013627223	Routerboardc_1c:a3:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:a3:59 Cost = 0 Port = 0x8002
13	24.016035280	Routerboardc_1c:a3:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:a3:59 Cost = 0 Port = 0x8002
14	26.017882512	Routerboardc_1c:a3:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:a3:59 Cost = 0 Port = 0x8002
15	28.010020340	Routerboardc_1c:a3:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:a3:59 Cost = 0 Port = 0x8002
16	30.012188772	Routerboardc_1c:a3:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:a3:59 Cost = 0 Port = 0x8002
17	32.014355039	Routerboardc_1c:a3:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:a3:59 Cost = 0 Port = 0x8002
18	34.016541071	Routerboardc_1c:a3:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:a3:59 Cost = 0 Port = 0x8002
19	36.018699027	Routerboardc_1c:a3:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:a3:59 Cost = 0 Port = 0x8002

Figura 11 - Ping do Tux42 para o endereço de broadcast (172.16.41.255). Captura no Tux44



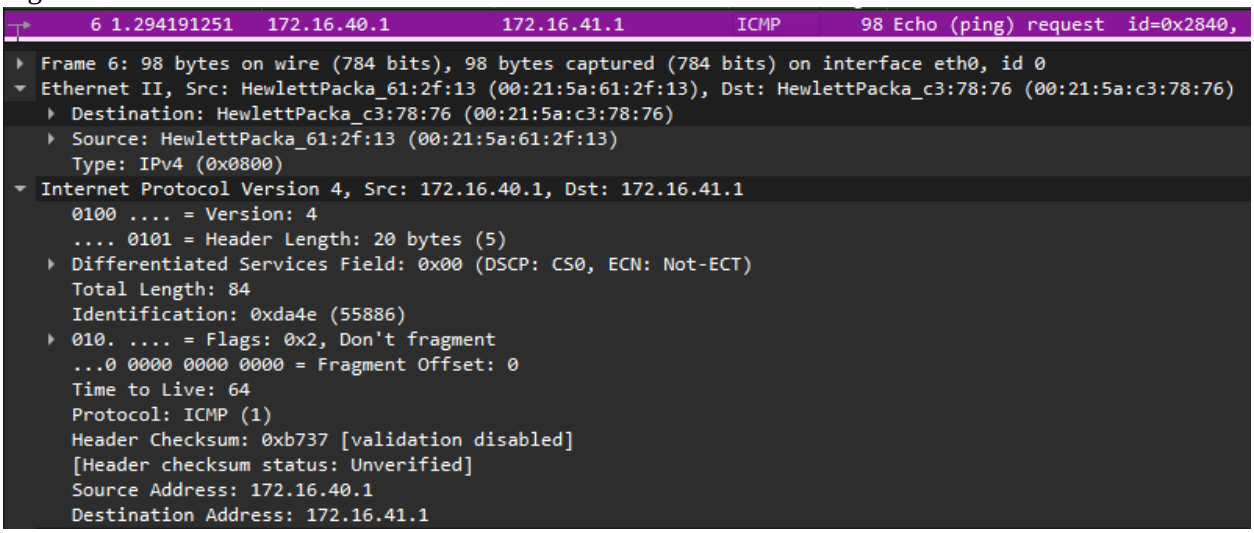
No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	HewlettPacka_c3:78:…	HewlettPacka_61:2f:…	ARP	60	Who has 172.16.40.1? Tell 172.16.40.254
2	0.000023955	HewlettPacka_61:2f:…	HewlettPacka_c3:78:…	ARP	42	172.16.40.1 is at 00:21:5a:61:2f:13
3	0.072449925	HewlettPacka_61:2f:…	HewlettPacka_c3:78:…	ARP	42	Who has 172.16.40.254? Tell 172.16.40.1
4	0.072576198	HewlettPacka_c3:78:…	HewlettPacka_61:2f:…	ARP	60	172.16.40.254 is at 00:21:5a:c3:78:76

Figura 12 – Ping do Tux43 para o Tux42



No.	Time	Source	Destination	Protocol	Length	Info
6	1.294191251	172.16.40.1	172.16.41.1	ICMP	98	Echo (ping) request id=0x2840, seq=1/256, ttl=64 (reply in 7)
7	1.294454273	172.16.41.1	172.16.40.1	ICMP	98	Echo (ping) reply id=0x2840, seq=1/256, ttl=63 (request in 6)
9	2.312457121	172.16.40.1	172.16.41.1	ICMP	98	Echo (ping) request id=0x2840, seq=2/512, ttl=64 (reply in 10)
10	2.312699401	172.16.41.1	172.16.40.1	ICMP	98	Echo (ping) reply id=0x2840, seq=2/512, ttl=63 (request in 9)
11	3.336458288	172.16.40.1	172.16.41.1	ICMP	98	Echo (ping) request id=0x2840, seq=3/768, ttl=64 (reply in 12)
12	3.336698333	172.16.41.1	172.16.40.1	ICMP	98	Echo (ping) reply id=0x2840, seq=3/768, ttl=63 (request in 11)
14	4.360453099	172.16.40.1	172.16.41.1	ICMP	98	Echo (ping) request id=0x2840, seq=4/1024, ttl=64 (reply in 15)
15	4.360741893	172.16.41.1	172.16.40.1	ICMP	98	Echo (ping) reply id=0x2840, seq=4/1024, ttl=63 (request in 14)
16	5.384457688	172.16.40.1	172.16.41.1	ICMP	98	Echo (ping) request id=0x2840, seq=5/1280, ttl=64 (reply in 17)
17	5.384692914	172.16.41.1	172.16.40.1	ICMP	98	Echo (ping) reply id=0x2840, seq=5/1280, ttl=63 (request in 16)
19	6.408464791	172.16.40.1	172.16.41.1	ICMP	98	Echo (ping) request id=0x2840, seq=6/1536, ttl=64 (reply in 20)
20	6.408705674	172.16.41.1	172.16.40.1	ICMP	98	Echo (ping) reply id=0x2840, seq=6/1536, ttl=63 (request in 19)
21	7.432465679	172.16.40.1	172.16.41.1	ICMP	98	Echo (ping) request id=0x2840, seq=7/1792, ttl=64 (reply in 22)
22	7.432703698	172.16.41.1	172.16.40.1	ICMP	98	Echo (ping) reply id=0x2840, seq=7/1792, ttl=63 (request in 21)
24	8.456468452	172.16.40.1	172.16.41.1	ICMP	98	Echo (ping) request id=0x2840, seq=8/2048, ttl=64 (reply in 25)
25	8.456729868	172.16.41.1	172.16.40.1	ICMP	98	Echo (ping) reply id=0x2840, seq=8/2048, ttl=63 (request in 24)
26	9.480477301	172.16.40.1	172.16.41.1	ICMP	98	Echo (ping) request id=0x2840, seq=9/2304, ttl=64 (reply in 27)
27	9.480741091	172.16.41.1	172.16.40.1	ICMP	98	Echo (ping) reply id=0x2840, seq=9/2304, ttl=63 (request in 26)
29	10.504478817	172.16.40.1	172.16.41.1	ICMP	98	Echo (ping) request id=0x2840, seq=10/2560, ttl=64 (reply in 30)
30	10.504714321	172.16.41.1	172.16.40.1	ICMP	98	Echo (ping) reply id=0x2840, seq=10/2560, ttl=63 (request in 29)
31	11.528433679	172.16.40.1	172.16.41.1	ICMP	98	Echo (ping) request id=0x2840, seq=11/2816, ttl=64 (reply in 32)
32	11.528667507	172.16.41.1	172.16.40.1	ICMP	98	Echo (ping) reply id=0x2840, seq=11/2816, ttl=63 (request in 31)
34	12.556437240	172.16.40.1	172.16.41.1	ICMP	98	Echo (ping) request id=0x2840, seq=12/3072, ttl=64 (reply in 35)
35	12.556666808	172.16.41.1	172.16.40.1	ICMP	98	Echo (ping) reply id=0x2840, seq=12/3072, ttl=63 (request in 34)
36	13.576436990	172.16.40.1	172.16.41.1	ICMP	98	Echo (ping) request id=0x2840, seq=13/3328, ttl=64 (reply in 37)
37	13.576695821	172.16.41.1	172.16.40.1	ICMP	98	Echo (ping) reply id=0x2840, seq=13/3328, ttl=63 (request in 36)

Figura 13 – Pacotes ICMP



```
6 1.294191251   172.16.40.1          172.16.41.1          ICMP     98 Echo (ping) request  id=0x2840,
Frame 6: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface eth0, id 0
Ethernet II, Src: HewlettPacka_61:2f:13 (00:21:5a:61:2f:13), Dst: HewlettPacka_c3:78:76 (00:21:5a:c3:78:76)
  Destination: HewlettPacka_c3:78:76 (00:21:5a:c3:78:76)
  Source: HewlettPacka_61:2f:13 (00:21:5a:61:2f:13)
  Type: IPv4 (0x0800)
Internet Protocol Version 4, Src: 172.16.40.1, Dst: 172.16.41.1
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
  Total Length: 84
  Identification: 0xda4e (55886)
  010. .... = Flags: 0x2, Don't fragment
  ...0 0000 0000 0000 = Fragment Offset: 0
  Time to Live: 64
  Protocol: ICMP (1)
  Header Checksum: 0xb737 [validation disabled]
  [Header checksum status: Unverified]
  Source Address: 172.16.40.1
  Destination Address: 172.16.41.1
```

Figura 14 - Ping do Tux43 para o Tux42. Captura no Tux43

1	0.000000000	Routerbo_1c:95:ca	Spanning-tree-(for-…	STP	60 RST. Root = 32768/0/c4:ad:34:1c:95:ca Cost = 0 Port = 0x8001
2	3.650871256	Routerbo_1c:95:ca	Spanning-tree-(for-…	STP	60 RST. Root = 32768/0/c4:ad:34:1c:95:ca Cost = 0 Port = 0x8001
3	3.650876460	172.16.50.1	172.16.2.1	DNS	86 Standard query 0x2cf9 PTR 26.114.82.140.in-addr.arpa
4	3.650816771	172.16.2.1	172.16.50.1	DNS	131 Standard query response 0x2cf9 PTR 26.114.82.140.in-addr.arpa PTR lb-140-82-114-26-iad.github.com
5	3.650967766	172.16.50.1	172.16.2.1	DNS	86 Standard query 0x631f PTR 194.26.161.35.in-addr.arpa
6	3.160669286	172.16.2.1	172.16.50.1	DNS	149 Standard query response 0x631f PTR 194.26.161.35.in-addr.arpa PTR ec2-35-161-26-194.us-west-2.compute.amazonaws.com
7	3.681110511	172.16.50.1	172.16.2.1	DNS	69 Standard query 0x42ac A ftp.up.pt
8	3.681936027	172.16.2.1	172.16.50.1	DNS	107 Standard query response 0x42ac A ftp.up.pt CNAME mirrors.up.pt A 193.137.29.15
9	3.682816693	172.16.50.1	193.137.29.15	TCP	74 36340 → 21 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=356591175 TSecr=0 WS=128
10	3.685510822	193.137.29.15	172.16.50.1	TCP	74 21 → 36340 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1380 SACK_PERM=1 TSval=2531393048 TSecr=356591175 WS=128
11	3.685528561	172.16.50.1	193.137.29.15	TCP	66 36340 → 21 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=356591179 TSecr=2531393048
12	3.691836151	193.137.29.15	172.16.50.1	FTP	139 Response: 220-Welcome to the University of Porto's mirror archive (mirrors.up.pt)
13	3.691860734	172.16.50.1	193.137.29.15	TCP	66 36340 → 21 [ACK] Seq=1 Ack=74 Win=64256 Len=0 TSval=356591185 TSecr=2531393052
14	3.691879731	193.137.29.15	172.16.50.1	FTP	141 Response: 220--
15	3.691867274	172.16.50.1	193.137.29.15	TCP	66 36340 → 21 [ACK] Seq=1 Ack=149 Win=64256 Len=0 TSval=356591185 TSecr=2531393052
16	3.691952505	193.137.29.15	172.16.50.1	FTP	310 Response: 220-All connections and transfers are logged. The max number of connections is 200.
17	3.691960118	172.16.50.1	193.137.29.15	TCP	66 36340 → 21 [ACK] Seq=1 Ack=393 Win=64128 Len=0 TSval=356591185 TSecr=2531393052
18	3.692742543	172.16.50.1	193.137.29.15	FTP	81 Request: user anonymous
19	3.694545689	193.137.29.15	172.16.50.1	TCP	66 21 → 36340 [ACK] Seq=393 Ack=16 Win=65280 Len=0 TSval=2531393057 TSecr=356591186
20	3.697444565	193.137.29.15	172.16.50.1	FTP	100 Response: 331 Please specify the password.
21	3.697479416	172.16.50.1	193.137.29.15	FTP	72 Request: pass
22	3.701095346	193.137.29.15	172.16.50.1	FTP	89 Response: 230 Login successful.
23	3.701133619	172.16.50.1	193.137.29.15	FTP	71 Request: pasv
24	3.704425349	193.137.29.15	172.16.50.1	FTP	117 Response: 227 Entering Passive Mode (193,137,29,15,225,91).
25	3.704491278	172.16.50.1	193.137.29.15	TCP	74 47600 → 57691 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=356591198 TSecr=0 WS=128
26	3.708141012	193.137.29.15	172.16.50.1	TCP	74 57691 → 47600 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1380 SACK_PERM=1 TSval=2531393070 TSecr=356591198 WS=128
27	3.708161335	172.16.50.1	193.137.29.15	TCP	66 47600 → 57691 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=356591202 TSecr=2531393070
28	3.708182707	172.16.50.1	193.137.29.15	FTP	94 Request: retr pub/kodi/timestamp.txt
29	3.710046544	193.137.29.15	172.16.50.1	FTP-DA…	77 FTP Data: 11 bytes (PASV) (retr pub/kodi/timestamp.txt)
30	3.710070081	172.16.50.1	193.137.29.15	TCP	66 47600 → 57691 [ACK] Seq=1 Ack=12 Win=64256 Len=0 TSval=356591203 TSecr=2531393072
31	3.710075109	193.137.29.15	172.16.50.1	TCP	66 57691 → 47600 [FIN, ACK] Seq=12 Ack=1 Win=65280 Len=0 TSval=2531393072 TSecr=356591202
32	3.710141807	193.137.29.15	172.16.50.1	FTP	146 Response: 150 Opening BINARY mode data connection for pub/kodi/timestamp.txt (11 bytes).
33	3.753627606	172.16.50.1	193.137.29.15	TCP	66 36340 → 21 [ACK] Seq=55 Ack=581 Win=64128 Len=0 TSval=356591247 TSecr=2531393072
34	3.753638012	172.16.50.1	193.137.29.15	TCP	66 47600 → 57691 [ACK] Seq=1 Ack=13 Win=64256 Len=0 TSval=356591247 TSecr=2531393072
35	3.756831197	193.137.29.15	172.16.50.1	FTP	90 Response: 226 Transfer complete.
36	3.756847400	172.16.50.1	193.137.29.15	TCP	66 36340 → 21 [ACK] Seq=55 Ack=605 Win=64128 Len=0 TSval=356591250 TSecr=2531393118
37	3.756902225	172.16.50.1	193.137.29.15	TCP	66 47600 → 57691 [FIN, ACK] Seq=1 Ack=13 Win=64256 Len=0 TSval=356591250 TSecr=2531393072
38	3.756918288	172.16.50.1	193.137.29.15	TCP	66 36340 → 21 [FIN, ACK] Seq=55 Ack=605 Win=64128 Len=0 TSval=356591250 TSecr=2531393118
39	3.760198355	193.137.29.15	172.16.50.1	TCP	66 57691 → 47600 [ACK] Seq=13 Ack=2 Win=65280 Len=0 TSval=2531393122 TSecr=356591250
40	3.760445991	193.137.29.15	172.16.50.1	TCP	66 21 → 36340 [FIN, ACK] Seq=605 Ack=56 Win=65280 Len=0 TSval=2531393122 TSecr=356591250
41	3.760456835	172.16.50.1	193.137.29.15	TCP	66 36340 → 21 [ACK] Seq=56 Ack=606 Win=64128 Len=0 TSval=356591254 TSecr=2531393122

Figura 15 – Aplicação de Download

(Nota: Este teste foi efetuado numa bancada diferente)

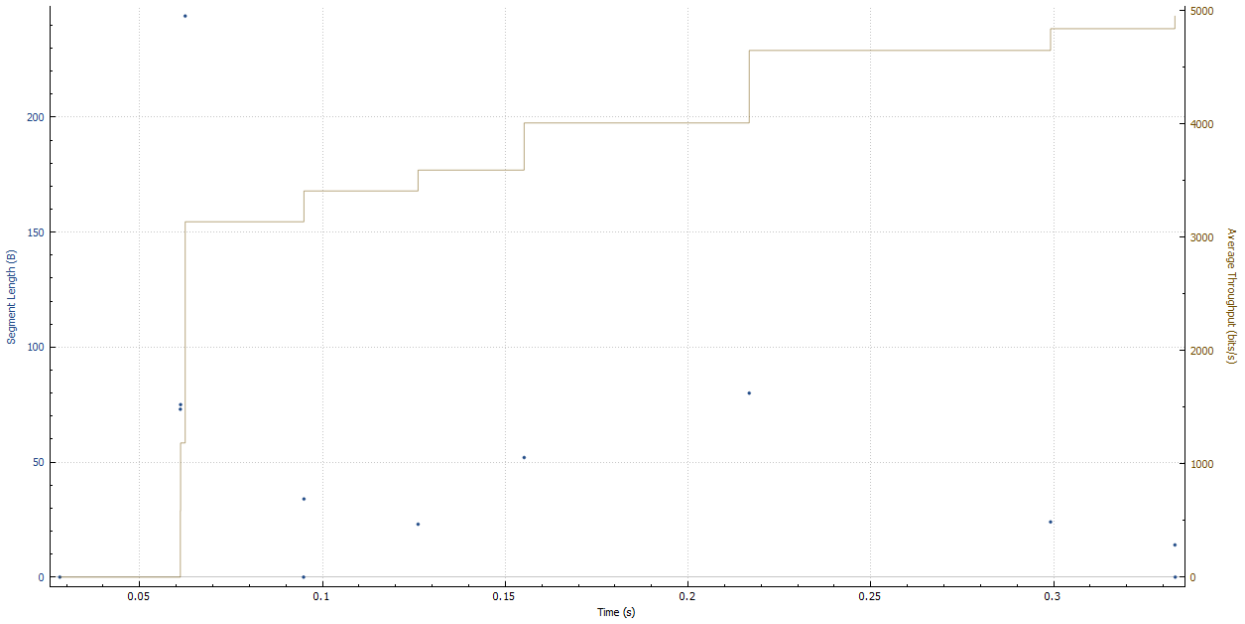


Figura 16 – Taxa de transmissão

```
> ./bin/download ftp://ftp.up.pt/pub/kodi/timestamp.txt
Host name  : mirrors.up.pt
IP Address : 193.137.29.15
Connected to 193.137.29.15:21
220-Welcome to the University of Porto's mirror archive (mirrors.up.pt)
220----------------------------------------------------------------
220-
220-All connections and transfers are logged. The max number of connections is 200.
220-
220-For more information please visit our website: http://mirrors.up.pt/
220-Questions and comments can be sent to mirrors@uporto.pt
220-
220-
220

Bytes written: 15
> user anonymous
< 331 Please specify the password.

Bytes written: 10
> pass pass
< 331 Please specify the password.
230 Login successful.

Logged in
Bytes written: 5
> pasv
< essful.
227 Entering Passive Mode (193,137,29,15,212,160).

Connected to 193.137.29.15:54432
Bytes written: 28
> retr pub/kodi/timestamp.txt
< 331 Plea♦150 Opening BINARY mode data connection for pub/kodi/timestamp.txt (11 bytes).

Downloading the file called timestamp.txt
The file has successfully downloaded
< ♦♦♦♦♦♦♦♦♦♦226 Transfer complete.

Closing connection
Bytes written: 5
> quit
File downloaded successfully
```

Figura 17 – Exemplo de um download bem sucedido usando a nossa aplicação

Código

main.c

```c
#include "../include/main.h"

#include <arpa/inet.h>
#include <netdb.h>
```

```c
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "Usage: %s <address to get IP address>\n", argv[0]);
        return -1;
    }

    char ip[20], answer[500];
    int port, control_socket, data_socket;

    urlArgs url_args;
    if (url_parse(argv[1], &url_args) != 0) {
        fprintf(stderr, "Error parsing URL\n");
        return -1;
    }

    control_socket = create_socket(url_args.ip, 21);
    if (control_socket < 0 || read_answer(control_socket, answer) != 220) {
        fprintf(stderr, "Failed to create socket to '%s' on port %d\n",
                url_args.ip, 21);
        return -1;
    }

    if (connection_server(control_socket, url_args.user, url_args.password) <
        0) {
        fprintf(stderr, "Error connecting to server\n");
        return -1;
    }

    if (passive_mode(control_socket, ip, &port) != 0) {
        fprintf(stderr, "Error entering passive mode\n");
        return -1;
    }

    data_socket = create_socket(ip, port);
    if (request_file(control_socket, url_args.path) != 150) {
        fprintf(stderr, "Error requesting file\n");
        return -1;
    }
```

```
    if (download_file(control_socket, data_socket, url_args.filename) != 226) {
        fprintf(stderr, "Error downloading file\n");
        return -1;
    }

    if (terminate_connection(control_socket, data_socket) != 0) {
        fprintf(stderr, "Error closing connection\n");
        return -1;
    }

    printf("File downloaded successfully\n");
    return 0;
}

int url_parse(char *url, urlArgs *url_parse) {
    struct hostent *h;
    char *start = strtok(url, "/");
    char *middle = strtok(NULL, "/");
    char *final = strtok(NULL, "");

    if (start == NULL || middle == NULL || final == NULL) {
        printf("Invalid URL\n");
        return -1;
    }

    if (strchr(middle, '@') != NULL) {
        char *login = strtok(middle, "@");
        char *host = strtok(NULL, "@");
        char *user = strtok(login, ":");
        char *password = strtok(NULL, ":");
        strcpy(url_parse->user, user);
        if (password != NULL) {
            strcpy(url_parse->password, password);
        } else {
            strcpy(url_parse->password, "pass");
        }
        strcpy(url_parse->host, host);
    } else {
        strcpy(url_parse->user, "anonymous");
        strcpy(url_parse->password, "pass");
        strcpy(url_parse->host, middle);
    }
```

```
    char *filename = final, *p;
    for (p = final; *p; p++) {
        if (*p == '/' || *p == '\\' || *p == ':') {
            filename = p + 1;
        }
    }

    strcpy(url_parse->filename, filename);
    strcpy(url_parse->path, final);

    if (!strcmp(url_parse->host, "") || !strcmp(url_parse->path, "")) {
        printf("Invalid URL\n");
        return -1;
    }

    if ((h = gethostbyname(url_parse->host)) == NULL) {
        herror("gethostbyname()");
        return -1;
    }

    strcpy(url_parse->host_name, h->h_name);
    strcpy(url_parse->ip, inet_ntoa(*((struct in_addr *)h->h_addr)));

    printf("Host name  : %s\n", h->h_name);
    printf("IP Address : %s\n", inet_ntoa(*((struct in_addr *)h->h_addr)));

    return 0;
}

int create_socket(char *serverAddress, int serverPort) {
    int sockfd;
    struct sockaddr_in server_addr;
    // char buf[] = "Mensagem de teste na travessia da pilha TCP/IP\n";

    /*server address handling*/
    bzero((char *)&server_addr, sizeof(server_addr));
    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = inet_addr(serverAddress);
    server_addr.sin_port = htons(serverPort);

    /*open a TCP socket*/
    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
```

```
        fprintf(stderr, "socket()");
        return -1;
    }
    /*connect to the server*/
    if (connect(sockfd, (struct sockaddr *)&server_addr, sizeof(server_addr)) <
        0) {
        fprintf(stderr, "connect()");
        return -1;
    }
    printf("Connected to %s:%d\n", serverAddress, serverPort);
    return sockfd;
}

int terminate_connection(const int control_socket, const int data_socket) {
    printf("Closing connection\n");
    if (write_to_server(control_socket, "quit\n") < 0) {
        return -1;
    }
    return close(control_socket) || close(data_socket);
}

int write_to_server(int socket, char *message) {
    size_t bytes;
    bytes = write(socket, message, strlen(message));
    if (bytes > 0)
        printf("Bytes written: %ld\n", bytes);
    else {
        fprintf(stderr, "write()");
        return -1;
    }
    printf("> %s", message);
    return 0;
}

int read_answer(int socket, char *response) {
    FILE *fp = fdopen(socket, "r");
    if (fp == NULL) {
        fprintf(stderr, "fdopen()");
        return -1;
    }
    char *line;
    size_t len = 0;
    int code;
```

```
    while ((getline(&line, &len, fp)) > 0) {
        strncat(response, line, len - 1);
        if (line[3] == ' ') {
            sscanf(line, "%d", &code);
            break;
        }
    }

    free(line);
    printf("< %s\n", response);
    return code;
}

int connection_server(int socket, char *user, char *password) {
    char response[500];
    char user_message[5 + strlen(user) + 1];  //  'user '= 5; + string pass + \0
    sprintf(user_message, "user %s\n", user);
    if (write_to_server(socket, user_message) < 0) {
        return -1;
    }
    if (read_answer(socket, response) != 331) return -1;
    char pass_message[5 + strlen(password) +
                      1];  // 'pass '= 5; + string pass + \0
    sprintf(pass_message, "pass %s\n", password);
    if (write_to_server(socket, pass_message) < 0) {
        return -1;
    }

    if (read_answer(socket, response) != 230) return -1;
    printf("Logged in\n");
    return 0;
}

int passive_mode(int socket, char *ip, int *port) {
    char *message = "pasv\n";
    char response[500];
    int ip1, ip2, ip3, ip4, port1, port2;
    if (write_to_server(socket, message) < 0) {
        return -1;
    }

    if (read_answer(socket, response) != 227) {
        printf("Error entering passive mode\n");
```

```
        return -1;
    }

    strtok(response, "(");
    char *response2 = strtok(NULL, ")");

    sscanf(response2, "%d,%d,%d,%d,%d,%d", &ip1, &ip2, &ip3, &ip4, &port1,
           &port2);
    *port = port1 * 256 + port2;

    sprintf(ip, "%d.%d.%d.%d", ip1, ip2, ip3, ip4);
    return 0;
}

int request_file(int connectionSocket, char *filename) {
    char fileCommand[5 + strlen(filename) + 1], answer[500];
    sprintf(fileCommand, "retr %s\n", filename);
    if (write_to_server(connectionSocket, fileCommand) < 0) {
        return -1;
    }
    return read_answer(connectionSocket, answer);
}

int download_file(int control_socket, int data_socket,
                  const char *targetfilename) {
    FILE *fileDescriptor = fopen(targetfilename, "wb");
    if (fileDescriptor == NULL) {
        fprintf(stderr, "Error opening or creating file '%s'\n",
                targetfilename);
        return -1;
    }

    char buffer[1024];
    ssize_t bytesRead;
    printf("Downloading the file called %s\n", targetfilename);
    while ((bytesRead = read(data_socket, buffer, sizeof(buffer) - 1)) > 0) {
        buffer[bytesRead] = '\0';
        if (fwrite(buffer, 1, bytesRead, fileDescriptor) != bytesRead) {
            fprintf(stderr, "Error writing data to file\n");
            fclose(fileDescriptor);
            return -1;
        }
    }
```

```
    if (bytesRead < 0) {
        fprintf(stderr, "Error reading from data socket\n");
        fclose(fileDescriptor);
        return -1;
    }

    printf("The file has successfully downloaded\n");

    if (fclose(fileDescriptor) < 0) {
        fprintf(stderr, "Error closing file\n");
        return -1;
    }
    char answer[500];
    return read_answer(control_socket, answer);
}
```

main.h

```
#ifndef MAIN_H
#define MAIN_H

#include <arpa/inet.h>
#include <netdb.h>
#include <unistd.h>

#define h_addr h_addr_list[0]
#define SERVER_PORT 21
#define SERVER_ADDR "ftp.up.pt"
typedef struct {
    char *h_name;  // Official name of the host.
    char **
        h_aliases;  // A NULL-terminated array of alternate names  for the host.
    int h_addrtype;      // The type of address being returned; usually AF_INET.
    int h_length;        // The length of the address in bytes.
    char **h_addr_list;  // A zero-terminated array of network
                         //  addresses for the host.
                         // Host addresses are in Network Byte Order.
} hostent;

typedef struct {
    char host[256];
```

```
    char user[64];
    char password[64];
    char path[256];
    char filename[128];
    char host_name[256];
    char ip[16];
} urlArgs;

// URL parsing
int url_parse(char *url, urlArgs *parsed_url);

// Socket creation and termination
int create_socket(char *ip, int port);
int terminate_connection(const int control_socket, const int data_socket);

// FTP commands
int write_to_server(int socket, char *message);
int read_answer(int socket, char *response);
int connection_server(int socket, char *user, char *password);
int passive_mode(int socket, char *ip, int *port);
int request_file(int connection_socket, char *filename);
int download_file(int control_socket, int data_socket,
                  const char *target_filename);

#endif  // MAIN_H
```