
**Licenciatura em Engenharia Informática e Computação**

**Cadeira de Redes de Computadores**

**Turma 5 – Bancada 3**

- António Guilherme Sanches de Magalhães Koch e Silva (up20210860)
- Gonçalo Nuno Leitão Pinho da Costa (up202103336)

Porto, 23 de dezembro de 2023

# Introdução

Este relatório abrange uma análise do segundo trabalho prático realizado para a Unidade Curricular de Rede de Computadores que está dividida em duas partes distintas. Na primeira secção, examinamos uma aplicação de download que implementa um cliente FTP simples em linguagem C. O código é estruturado em três arquivos principais - utils.h, utils.c e download.c e tem como propósito de realizar o download de arquivos de um servidor FTP, fornecendo uma visão detalhada do processo, desde a análise de URLs até o encerramento adequado das conexões.

A segunda parte do relatório mergulha numa série de experiências relacionadas à configuração e análise de redes . Estas experiências cobrem desde a configuração de redes IP e bridges num switch até a implementação de um router. Foram explorados cenários que envolvem a comunicação entre dispositivos, a segmentação de redes, e a configuração de um router para facilitar a comunicação indireta entre diferentes redes.

Cada secção contribui para uma compreensão mais profunda dos conceitos essenciais para a configuração e desenvolvimento de aplicações de rede. A análise minuciosa dessas experiências oferece insights sobre a interação entre dispositivos numa rede, os protocolos fundamentais, e as estratégias de configuração de redes para otimizar a comunicação entre diversos sistemas.

## Parte 1: Aplicação de download

Para o desenvolvimento desta parte do projeto foi utilizado um código que permitia download de ficheiros através do protocolo FTP, este código segue as normas RFC959, funcionamento e arquitetura do protocolo FTP e a norma RFC1738, tratamento das partes constituintes dos endereços URL.

Para isso, o código aceita como argumento um link, no formato ftp://<user>:<password>@<host>/<url-path>, permitindo a transferência de qualquer ficheiro desde um servidor FTP, usando as normas referidas no RFC959, que fala sobre o FTP, mas também respeitando o RFC1738, que aborda a questão de informações provenientes de URL's.

O código consiste num cliente FTP simples implementado em linguagem C. Usando um URL fornecido como argumento na linha de comandos, permite o download de um arquivo de um servidor FTP. O código está dividido em três ficheiros: utils.h, utils.c e download.c.

O ficheiro utils.h contém definições de constantes, uma estrutura de dados chamada struct url para armazenar informações do URL, e protótipos de funções que serão implementadas no utils.c. Essas funções incluem o parsing do URL, a criação de um socket, a obtenção do tamanho do arquivo, o download do arquivo e a exibição de uma barra de progresso.

As funções especificadas no utils.h são executadas pelo utils.c. A função readFromSocket lê dados de um socket e imprime-os até que uma resposta completa seja recebida. A função create_socket cria e conecta um socket TCP ao endereço IP e porta fornecidos. As outras funções incluem o parsing do URL, a extração do número da porta de uma string, a obtenção do tamanho do arquivo, da resposta do servidor e do download real do arquivo.

O arquivo download.c contém a função main, que é o ponto de entrada do programa. Ela processa a linha de comandos, analisa o URL, obtém o endereço IP do host, mostra informações relevantes sobre a conexão FTP, cria e conecta o socket, faz a autenticação, entra

no modo passivo, cria um segundo socket para a transferência de dados e, finalmente, inicia o download do arquivo.

O código inclui manipulação básica de erros, exibindo mensagens de erro apropriadas e encerrando o programa em caso de falha. Além disso, fornece informações detalhadas sobre o processo de conexão e download.

De modo a validar o funcionamento do código, o mesmo foi testado em diferentes ambientes, modo anónimo, não anónimo, vários tipos de ficheiros, diferentes tamanhos de ficheiros, entre outros. Em casos em que o ficheiro não exista ou em erro, o programa termina. As respostas são impressas na consola para maior controlo por parte do utilizador, bem como uma barra de progresso da transferência do ficheiro.

# Parte 2: Configuração de rede e análise

## Experiência 1 – Configurar uma rede IP

Com esta experiência tínhamos como objetivo a comunicação entre dois computadores, ligados ao mesmo switch.

De modo a podermos realizar a experiencia, tivemos de inicialmente, ligar o E0 de cada um dos TUXs ao switch, para posteriormente configurarmos os seus respetivos endereços IP. Ao TUX34 atribuímos o IP 172.16.30.254, enquanto que o TUX 33 ficou com o endereço 172.16.30.1. Para podermos fazer esta atribuição dos IPS a cada um dos computadores tivemos de utilizar o comando ifconfig. Ao TUX 34 ficou atribuído o endereço MAC 00:21:5a:5a:74:3e. Relativamente ao TUX33, e já depois de feito o ping ao seu IP para descobrir o seu endereço MAC, vemos que este é 00:21:5a:5a:7d:b7. Como podemos ver pela figura 1 (Anexos), após ter sido feito o ping, o TUX34 guardou o endereço MAC do TUX33.

Na figura 2 está registada todas as conexões feitas entre estes dois computadores, onde são percetíveis as trocas de pacotes usando o protocolo ARP e o protocolo ICMP, pelo campo "Protocol" no Wireshark. O protocolo ARP (Address Resolution Protocol) tem como objetivo fazer a ligação entre um endereço IP e o respetivo endereço MAC, isto é, faz a conexão entre a Network Layer e a Link Layer, uma vez que na comunicação entre dois dispositivos, estes utilizam o endereço MAC para informar o destino e a fonte da informação. E isso é também percetível pela informação presente na linha 6, da figura 2, onde podemos ver que o endereço origem, envia um Broadcast, isto é, envia uma mensagem para todos os possíveis destinos, a perguntar quem tem o endereço 172.16.30.1. Na linha seguinte, o TUX33 informa o TUX34 de que ele é o destino e que tem o endereço Mac 00:21:5a:5a:74:3e. A partir deste momento, ambos os computadores sabem o endereço MAC para onde querem enviar a informação, desta forma não há necessidade de haver mais broadcasts.

Posteriormente, esta associação é guardada em cada um dos computadores, numa tabela MAC, onde é possível ver todos os endereços que o computador conheço e o seu endereço MAC. Desta forma, sempre que o computador pretender comunicar com aquele endereço IP da próxima vez não terá de fazer um Broadcast novamente, vai recorrer à sua tabela MAC e perceber que já sabe em que endereço MAC, aquele endereço IP se localiza. A tabela ARP do TUX34 está visível na figura 1, logo após o comando "arp -a". Se se eliminasse esta tabela e fosse feito novo ping ao mesmo TUX33, como o endereço deste não estaria guardado nesta tabela, seria feito um novo Broadcast para descobrir o seu endereço MAC.

Mais adiante, após a troca de informação sobre os seus endereços, os dois computadores já poderiam comunicar entre eles diretamente. Para isso, utilizaram o protocolo ICMP (Internet Control Message Protocol), com mensagens request e reply entre ambos, onde viam associados os endereços IP aos MAC.

Para esta experiência, utilizamos os comandos presentes e descritos no anexo 2.1. As figuras 1 e 2 são também relativas a esta experiência.

## Experiência 2 – Duas bridges num switch

Com esta experiência tínhamos como objetivo a criação de duas redes locais, uma com os TUXs 33 e 34, outra apenas com o TUX 32.

Para isso, tivemos de repetir a configuração relativa à experiência anterior, configurando o TUX32 com o IP 172.16.31.1, o TUX33 com o IP 172.16.30.1, já o IP 172.16.30.254

ficou para o TUX34. Na consola switch foram criadas as duas bridges , 30 e 31. Posteriormente, removemos as portas dos TUXs dadas por defeito (utilizando os comandos interface bridge port remove) e fizemos a ligação do TUX33 e 34 à bridge 30 e o TUX32 ficou conectado à bridge 31 (utilizando os comandos interface bridge port add, visíveis no anexo 2.2).

Relativamente aos domínios de Broadcast, durante a experiência, foi possível observar um por cada uma das bridges criadas. Assim, tentando estabelecer conexão entre o TUX33 e o TUX34, esta tinham sucesso e a partir da figura 4, conseguimos perceber que existe uma resposta ao pedido de endereço MAC do IP 172.16.30.254 e posteriormente existe troca de request e reply entre os dois computadores. No entanto, quando se tentava contactar com o TUX32, não existia qualquer tipo de resposta, havia apenas requests e mensagens "No response Found" visíveis na figura 3.

## Experiência 3 – Configurar um router em Linux

Esta experiência foi desenvolvida em sequência a experiência anterior, para isso foram utilizados todos os comandos da experiência anterior. Em acrescento, nesta experiência tínhamos como objetivo fazer com que o TUX34 fosse o ponto de comunicação entre o TUX32 e o 33. Desta forma, tivemos de adicionar uma nova bridge ao TUX34, de modo a este ficar ligado as duas ao mesmo tempo, e também tivemos de ativar o forwarding no TUX34.

O TUX34 teve de ficar atribuído a dois endereços IP, um em cada uma das bridges, sendo que na bridge 30 permaneceu com o endereço previamente atribuído, 172.16.30.254, já na bridge 31 atribuímos-lhe o endereço 172.16.31.253, usando o comando ifconfig.

Com esta configuração permitimos que o TUX34 funcionasse como router e ponto de ligação entre os outros dois computadores. Deste modo, permitimos a dois computadores que não estando na mesma rede, pudessem estabelecer conexão através de um intermediário. Através das imagens 11 e 12 conseguimos perceber que o redirecionamento acontecia no TUX34, uma vez que as mensagens tinham endereço fonte e destino os endereços IP dos TUXs 32 e 33, mas a mensagem estava a ser passado pelo TUX34, uma vez que este era o ponto de ligação entre os outros dois TUXs.

Através da figura 13 e 14 conseguimos perceber algo interessante, que está relacionado com o facto de que quando a observação da comunicação é feita na ponte entre os dois computadores, o endereço IP de destino e fonte são os endereços IP dos TUXs finais, no entanto, quando observamos os endereços MAC de destino e fonte, percebemos que em ambos os casos, estes não são os finais, mas sim o endereço MAC do TUX34. Este depois tratava do seu reencaminhamento para o destino final, onde colocava o seu MAC como endereço fonte e o destino respetivo. Portanto, apesar de os endereços IP serem os endereços finais do destino e fonte, os endereços MAC tinham de sofrer alterações durante a mensagem de modo a chegarem ao destino. Isto acontece, uma vez que ambos os TUXs sabem que têm de passar por um intermediário de modo a chegar ao destino. Isso é visível nas imagens 8 e 9, quando corremos o comando "route -n" vemos que para chegar ao endereço final usamos como gateway o endereço IP do TUX34.

## Experiência 4 – Configurar um router comercial usando NAT

Na quarta experiência, começamos com a infraestrutura que foi estabelecida na terceira experiência; essas LANs incluíam duas redes locais, uma com TUX33 e outra com TUX 32. O TUX 34 conecta essas duas subredes e serve como um router. O objetivo era configurar e adicionar um router com uma funcionalidade de Tradução de Endereço de Rede (NAT) à bridge 31 para permitir o acesso à internet.

A primeira coisa a fazer é conectar uma das portas do router à régua de conexões e a outra ao switch. Em seguida, como nas experiências anteriores, adicionamos a interface à bridge31. Após isso, substituímos o cabo que ligava o TUX 32 à consola do switch e o conectamos à consola do router comercial para que pudesse ser configurado. Usamos o comando "ip address add" na console do router para configurar os endereços IP correspondentes a cada interface. Por fim, ambos os TUXs foram conectados ao router por rotas padrão. Finalmente, criamos uma rota que conectasse o router a cada LAN usando o TUX 34.

Quando o TUX 32 estava conectado diretamente ao TUX 34 e não havia redirecionamentos ICMP, os pacotes de dados eram enviados pelo router pela rota padrão para o endereço IP de destino (figuras 14, 15, 16). Na segunda situação, quando as rotas e redirecionamentos foram ativadas, o router não foi usado para transmitir os pacotes. Isso ocorreu porque a conexão mais direta entre as redes estava disponível e o TUX 34 servia como intermediário entre as duas LANs. Portanto, os pacotes ICMP otimizam as conexões da rede, redirecionando quando necessário (figura 17).

Podemos ver o "traceroute" na figura 18.

Um método de mapeamento chamado Network Address Translation (NAT) pode converter endereços de uma rede local para um endereço público único e vice-versa. Assim, o endereço público é usado como origem quando um pacote é enviado para uma rede externa. Quando um computador responde, envia a resposta para esse endereço público. O endereço local que enviou a solicitação é então traduzido de volta para ele. Como resultado do uso extensivo do IPv4, é possível reduzir o número de endereços públicos utilizados e evitar an escassez de IPs únicos.

O NAT é ativado e a conexão à internet é estabelecida (figura 19). No entanto, quando o router é desativado, ele não consegue traduzir o endereço de destino recebido em um endereço da rede interna (figura 20). Como resultado, o emissor não consegue receber respostas às solicitações.

## Experiência 5 – DNS

Esta experiência tira proveito da configuração feita na rede, no decorrer das outras experiências, criando, para além disso, uma configuração DNS, para ser possível aceder a sites da internet, dentro da rede por nós criada, através do seu nome de domínio.

Para realizar esta experiência foi necessário primeiramente configurar o ficheiro resolv.conf, que se encontra na pasta ‘etc’ em cada um dos TUX´s, para que nele apenas estivesse presente a linha: “nameserver 127.16.2.1”, uma vez que este é o endereço do router presente no laboratório com acesso a internet.

Pela figura 21, conseguimos perceber que, antes de haver uma comunicação propriamente dita entre o computador e o website requisitado, existe troca de pacotes DNS de

modo a estabelecer uma ligação entre o nome do domínio que foi digitado no linha de comandos e o endereço IP desse domínio, uma vez que na troca dos restantes pacotes, serão utilizados os endereços IP's para ser estabelecida a comunicação.

Nas linhas 2 e 3 da figura 21 vemos que dois pacotes DNS são enviados desde o TUX32, com destino ao router do laboratório com pedidos de informação relativos aos endereços MAC e IP do domínio Google.com. Na linhas 4 e 5 é possível observar os endereços de fonte e de destino invertidos, já que o router está a informar o TUX32 dos endereços do site com o qual queremos estabelecer comunicação. Nas linhas 2 e 3 vemos apenas querys, enquanto que nas linhas 4 e 5 vemos também response, no campo de informação.

## Experiência 6 – Ligações TCP

Na execução da aplicação desenvolvida para esta experiência, observamos a abertura de duas ligações TCP distintas. A aplicação, compilada e executada no Tux53, demonstrou o seu funcionamento ao realizar o download de um ficheiro. Durante esse processo, pôde-se identificar a sequência de eventos que caracteriza a troca de informações entre cliente e servidor.

A primeira ligação TCP é estabelecida para o envio de comandos de controlo ao servidor, utilizando o Transmission Control Protocol (TCP) para garantir uma comunicação orientada e confiável. A segunda ligação é dedicada à receção do ficheiro selecionado. Ambas as ligações TCP incorporam um mecanismo de controlo de erros denominado ARQ (Automatic Repeat Request), o qual utiliza mensagens ACK (acknowledge) para confirmar a receção correta de pacotes e timeouts para identificar atrasos na receção.

O mecanismo ARQ revela-se essencial para assegurar a integridade da transmissão de dados. O "sequence number field" e os timeouts desempenham papéis cruciais nesse processo, permitindo a identificação do início da mensagem no pacote de dados e determinando o tempo de espera antes de considerar um pacote como perdido.

No que diz respeito ao controlo de fluxo, a aplicação implementa um mecanismo que permite ao recetor controlar o fluxo de receção de pacotes. A comunicação entre emissor e recetor é sensível às capacidades dos buffers de ambos, evitando a perda de pacotes na rede. Cada pacote enviado inclui uma "window size" em bytes, indicando ao emissor a quantidade de bytes que ainda pode enviar sem congestionar a rede.

O controlo de congestionamento é realizado pelos emissores de dados e adota o método Selective Repeat. Quando ocorre perda de pacotes, a rede é considerada congestionada. Para determinar o limite da rede, o emissor transfere vários pacotes de uma vez, utilizando estratégias como Additive Increase ou Slow Start. A perda de um pacote é detetada por timeouts ou três ACK consecutivos.

Na análise do comportamento da aplicação, é possível observar apenas pequenas variações na velocidade de transferência de pacotes. Essa estabilidade pode ser atribuída à ocorrência de congestionamento na rede, decorrente da competição por largura de banda entre os diversos usuários (figuras 22 e 23).

O controlo de congestionamento, uma característica inerente ao TCP (Transmission Control Protocol), desempenha um papel crucial nessas situações. Quando a rede atinge um estado de congestão devido à alta demanda por largura de banda, os mecanismos de controlo

de congestionamento do TCP entram em ação. Esses mecanismos ajustam dinamicamente a taxa de transmissão e a janela de congestionamento para otimizar a eficiência da rede e evitar a perda excessiva de pacotes.

Nesse contexto, a maior constância na velocidade de transferência observada reflete a capacidade do TCP em equilibrar a demanda concorrente por recursos de rede compartilhados. O protocolo trabalha para evitar sobrecarregar a rede, adaptando-se continuamente às condições de congestionamento.

É importante ressaltar que, se houvesse menos usuários a competir pela largura de banda, seria esperado que a velocidade de transferência aumentasse. Num cenário menos congestionado, o TCP teria mais recursos disponíveis para distribuir entre os usuários, resultando em taxas de transmissão individuais mais altas.

Além disso, em ambientes menos congestionados, as variações na quantidade de pacotes transferidos poderiam ser mais pronunciadas. Com menos concorrência por recursos, a aplicação teria mais flexibilidade para ajustar dinamicamente a taxa de transmissão, podendo experimentar flutuações mais marcantes na velocidade de transferência de pacotes.

Em resumo, a maior constância na velocidade de transferência observada em um ambiente com muitos usuários reflete a eficácia do controlo de congestionamento do TCP, enquanto a presença de menos usuários poderia resultar em maior velocidade e variações mais expressivas na transferência de pacotes. Essa dinâmica evidencia a sensibilidade do desempenho da aplicação às condições específicas do ambiente de rede.

Numa fase subsequente, ao iniciar um segundo download no Tux52 durante o download no Tux53, observamos uma redução na velocidade de transferência no Tux53 (figuras 24 e 25). Este comportamento é atribuído à ação simultânea do controlo de congestionamento por ambos os computadores na rede. Este contexto sugere que o código da aplicação foi desenvolvido de forma a incorporar eficientemente os conceitos de controlo de erros, fluxo e congestionamento para garantir uma transmissão estável e confiável.

# Referencias


1. https://www.rfc-editor.org/rfc/rfc959
2. https://www.rfc-editor.org/rfc/rfc1738
3. https://beej.us/guide/bgnet/pdf/bgnet_a4_c_2.pdf
4. https://github.com/Educorreia932/FEUP-RCOM/tree/master/Projects/Project%202

# ANEXOS

## Anexos 1 – Imagens

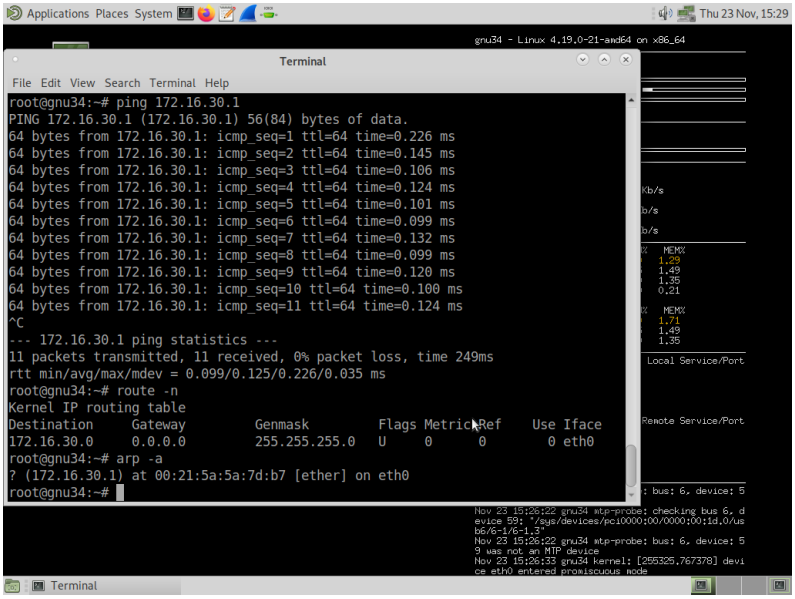


*Figura 1- TUX 34 após ping TUX 33 tabela ARP e tabela MAC*

| | | | | | |
|---|---|---|---|---|---|
| 6 | 8.218734869 | HewlettPacka_5a:74:… | Broadcast | ARP | 42 Who has 172.16.30.1? Tell 172.16.30.254 |
| 7 | 8.218859256 | HewlettPacka_5a:7d:… | HewlettPacka_5a:74:… | ARP | 60 172.16.30.1 is at 00:21:5a:5a:7d:b7 |
| 8 | 8.218876228 | 172.16.30.254 | 172.16.30.1 | ICMP | 98 Echo (ping) request id=0x7676, seq=1/256, ttl=64 (reply in 9) |
| 9 | 8.218965415 | 172.16.30.1 | 172.16.30.254 | ICMP | 98 Echo (ping) reply id=0x7676, seq=1/256, ttl=64 (request in 8) |
| 10 | 9.246060760 | 172.16.30.254 | 172.16.30.1 | ICMP | 98 Echo (ping) request id=0x7676, seq=2/512, ttl=64 (reply in 11) |
| 11 | 9.246160703 | 172.16.30.1 | 172.16.30.254 | ICMP | 98 Echo (ping) reply id=0x7676, seq=2/512, ttl=64 (request in 10) |
| 12 | 10.010693749 | Routerboardc_1c:a3:… | Spanning-tree-(for-… | STP | 60 RST. Root = 32768/0/c4:ad:34:1c:a3:2a Cost = 0 Port = 0x8015 |
| 13 | 10.270062276 | 172.16.30.254 | 172.16.30.1 | ICMP | 98 Echo (ping) request id=0x7676, seq=3/768, ttl=64 (reply in 14) |
| 14 | 10.270202168 | 172.16.30.1 | 172.16.30.254 | ICMP | 98 Echo (ping) reply id=0x7676, seq=3/768, ttl=64 (request in 13) |
| 15 | 11.294057297 | 172.16.30.254 | 172.16.30.1 | ICMP | 98 Echo (ping) request id=0x7676, seq=4/1024, ttl=64 (reply in 16) |
| 16 | 11.294187481 | 172.16.30.1 | 172.16.30.254 | ICMP | 98 Echo (ping) reply id=0x7676, seq=4/1024, ttl=64 (request in 15) |
| 17 | 12.012836410 | Routerboardc_1c:a3:… | Spanning-tree-(for-… | STP | 60 RST. Root = 32768/0/c4:ad:34:1c:a3:2a Cost = 0 Port = 0x8015 |
| 18 | 12.318060698 | 172.16.30.254 | 172.16.30.1 | ICMP | 98 Echo (ping) request id=0x7676, seq=5/1280, ttl=64 (reply in 19) |
| 19 | 12.318164482 | 172.16.30.1 | 172.16.30.254 | ICMP | 98 Echo (ping) reply id=0x7676, seq=5/1280, ttl=64 (request in 18) |
| 20 | 13.342066195 | 172.16.30.254 | 172.16.30.1 | ICMP | 98 Echo (ping) request id=0x7676, seq=6/1536, ttl=64 (reply in 21) |
| 21 | 13.342171516 | 172.16.30.1 | 172.16.30.254 | ICMP | 98 Echo (ping) reply id=0x7676, seq=6/1536, ttl=64 (request in 20) |

*Figura 2- Experiência 1*

| | | | | | |
|---|---|---|---|---|---|
| 37 | 47.162941183 | 172.16.30.1 | 172.16.31.1 | ICMP | 98 Echo (ping) request id=0x20bb, seq=8/2048, ttl=64 (no response found!) |
| 38 | 48.051858900 | Routerboardc_1c:a3:… | Spanning-tree-(for-… | STP | 60 RST. Root = 32768/0/c4:ad:34:1c:a3:2a Cost = 0 Port = 0x8002 |
| 39 | 48.186939486 | 172.16.30.1 | 172.16.31.1 | ICMP | 98 Echo (ping) request id=0x20bb, seq=9/2304, ttl=64 (no response found!) |
| 40 | 49.210952106 | 172.16.30.1 | 172.16.31.1 | ICMP | 98 Echo (ping) request id=0x20bb, seq=10/2560, ttl=64 (no response found!) |
| 41 | 50.054051149 | Routerboardc_1c:a3:… | Spanning-tree-(for-… | STP | 60 RST. Root = 32768/0/c4:ad:34:1c:a3:2a Cost = 0 Port = 0x8002 |
| 42 | 50.234943775 | 172.16.30.1 | 172.16.31.1 | ICMP | 98 Echo (ping) request id=0x20bb, seq=11/2816, ttl=64 (no response found!) |
| 43 | 51.258937957 | 172.16.30.1 | 172.16.31.1 | ICMP | 98 Echo (ping) request id=0x20bb, seq=12/3072, ttl=64 (no response found!) |

*Figura 3- Experiência 2 Ping TUX33 ao TUX 32*

| | | | | | |
|---|---|---|---|---|---|
| 22 | 12.847521252 | HewlettPacka_5a:7d:… | HewlettPacka_5a:74:… | ARP | 42 Who has 172.16.30.254? Tell 172.16.30.1 |
| 23 | 12.847654998 | HewlettPacka_5a:74:… | HewlettPacka_5a:7d:… | ARP | 60 172.16.30.254 is at 00:21:5a:5a:74:3e |
| 24 | 13.743557613 | 172.16.30.1 | 172.16.30.254 | ICMP | 98 Echo (ping) request id=0x2141, seq=7/1792, ttl=64 (reply in 25) |
| 25 | 13.743698692 | 172.16.30.254 | 172.16.30.1 | ICMP | 98 Echo (ping) reply id=0x2141, seq=7/1792, ttl=64 (request in 24) |
| 26 | 14.015334841 | Routerboardc_1c:a3:… | Spanning-tree-(for-… | STP | 60 RST. Root = 32768/0/c4:ad:34:1c:a3:2a Cost = 0 Port = 0x8002 |
| 27 | 14.767559129 | 172.16.30.1 | 172.16.30.254 | ICMP | 98 Echo (ping) request id=0x2141, seq=8/2048, ttl=64 (reply in 28) |
| 28 | 14.767688335 | 172.16.30.254 | 172.16.30.1 | ICMP | 98 Echo (ping) reply id=0x2141, seq=8/2048, ttl=64 (request in 27) |

*Figura 4- Experiência 2 Ping TUX 33 ao TUX 34*

| | | | | | |
|---|---|---|---|---|---|
| 10 | 16.035380617 | 172.16.30.1 | 172.16.30.255 | ICMP | 98 Echo (ping) request id=0x21ae, seq=1/256, ttl=64 (no response found!) |
| 11 | 16.035575963 | 172.16.30.254 | 172.16.30.1 | ICMP | 98 Echo (ping) reply id=0x21ae, seq=1/256, ttl=64 |
| 12 | 17.046559204 | 172.16.30.1 | 172.16.30.255 | ICMP | 98 Echo (ping) request id=0x21ae, seq=2/512, ttl=64 (no response found!) |
| 13 | 17.046707756 | 172.16.30.254 | 172.16.30.1 | ICMP | 98 Echo (ping) reply id=0x21ae, seq=2/512, ttl=64 |
| 14 | 18.019737986 | Routerboardc_1c:a3:… | Spanning-tree-(for-… | STP | 60 RST. Root = 32768/0/c4:ad:34:1c:a3:2a Cost = 0 Port = 0x8002 |
| 15 | 18.070556459 | 172.16.30.1 | 172.16.30.255 | ICMP | 98 Echo (ping) request id=0x21ae, seq=3/768, ttl=64 (no response found!) |
| 16 | 18.070698027 | 172.16.30.254 | 172.16.30.1 | ICMP | 98 Echo (ping) reply id=0x21ae, seq=3/768, ttl=64 |

*Figura 5- Experiência 2 Ping TUX 33 a todos*

| | | | | | |
|---|---|---|---|---|---|
| 3 | 2.544086926 | 172.16.31.1 | 172.16.31.255 | ICMP | 98 Echo (ping) request id=0x14cb, seq=1/256, ttl=64 (no response found!) |
| 4 | 3.574338050 | 172.16.31.1 | 172.16.31.255 | ICMP | 98 Echo (ping) request id=0x14cb, seq=2/512, ttl=64 (no response found!) |
| 5 | 4.004389665 | Routerboardc_1c:a3:… | Spanning-tree-(for-… | STP | 60 RST. Root = 32768/0/c4:ad:34:1c:a3:30 Cost = 0 Port = 0x8002 |
| 6 | 4.598331324 | 172.16.31.1 | 172.16.31.255 | ICMP | 98 Echo (ping) request id=0x14cb, seq=3/768, ttl=64 (no response found!) |
| 7 | 5.622325158 | 172.16.31.1 | 172.16.31.255 | ICMP | 98 Echo (ping) request id=0x14cb, seq=4/1024, ttl=64 (no response found!) |
| 8 | 6.006580097 | Routerboardc_1c:a3:… | Spanning-tree-(for-… | STP | 60 RST. Root = 32768/0/c4:ad:34:1c:a3:30 Cost = 0 Port = 0x8002 |
| 9 | 6.646322902 | 172.16.31.1 | 172.16.31.255 | ICMP | 98 Echo (ping) request id=0x14cb, seq=5/1280, ttl=64 (no response found!) |

*Figura 6- Experiência 2 Ping TUX 32 a todos*

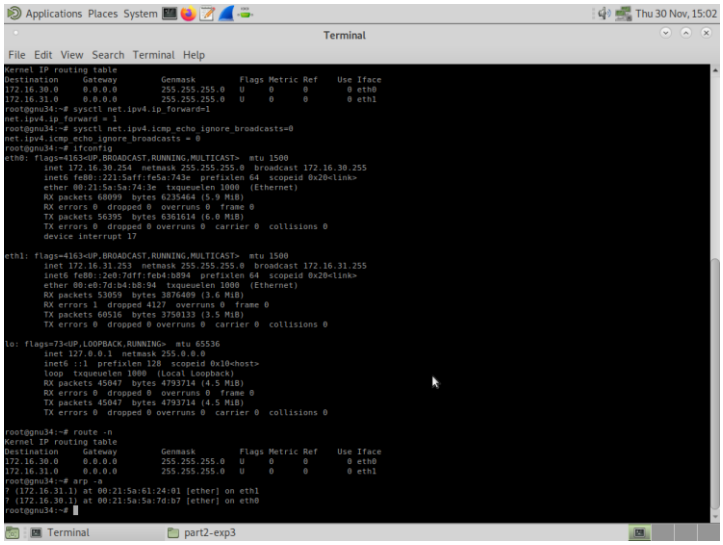


*Figura 7- Experiência 3 ifconfig no TUX 34*

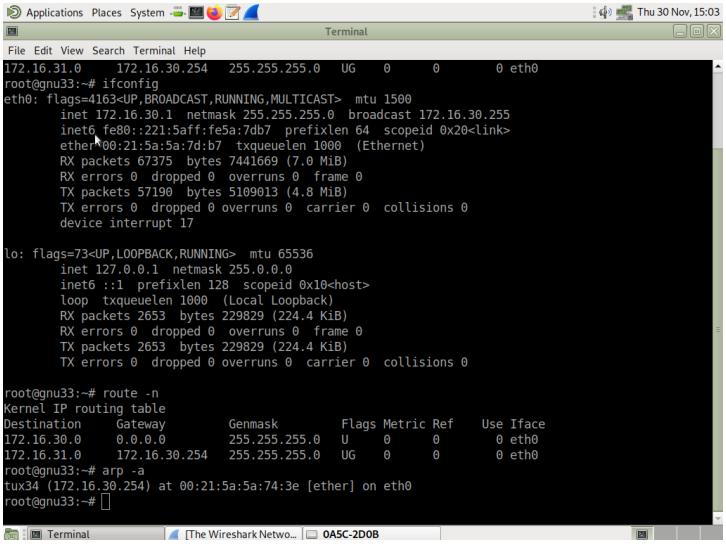


*Figura 8- Experiência 3 ifconfig no TUX 33*

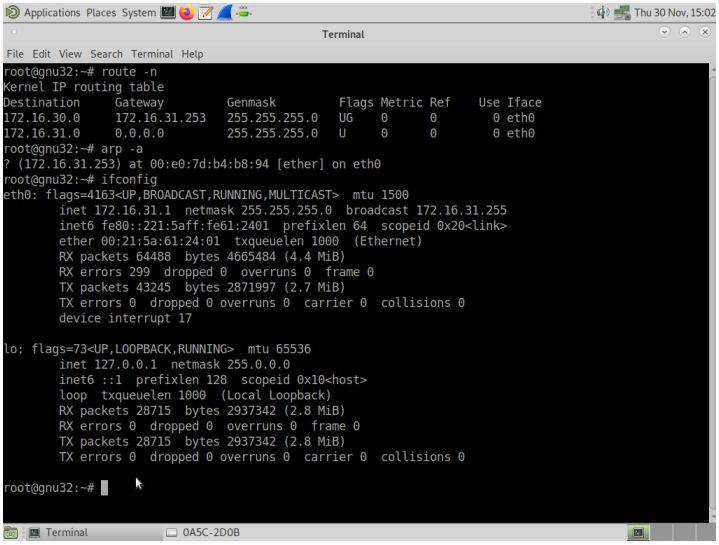


*Figura 9- Experiência 3 ifconfig no TUX 32*

```
 42 19.185407796  HewlettPacka_61:24:… Netronix_b4:b8:94    ARP      42 Who has 172.16.31.253? Tell 172.16.31.1
 43 19.185495167  Netronix_b4:b8:94    HewlettPacka_61:24:… ARP      60 172.16.31.253 is at 00:e0:7d:b4:b8:94
 44 20.022122995  Routerboardc_1c:a3:… Spanning-tree-(for-… STP      60 RST. Root = 32768/0/c4:ad:34:1c:a3:30  Cost = 0  Port = 0x8002
 45 20.145456975  172.16.31.1          172.16.30.1          ICMP     98 Echo (ping) request  id=0x0b50, seq=15/3840, ttl=64 (reply in 46)
 46 20.145699604  172.16.30.1          172.16.31.1          ICMP     98 Echo (ping) reply    id=0x0b50, seq=15/3840, ttl=63 (request in 45)
 47 21.169455767  172.16.31.1          172.16.30.1          ICMP     98 Echo (ping) request  id=0x0b50, seq=16/4096, ttl=64 (reply in 48)
 48 21.169709780  172.16.30.1          172.16.31.1          ICMP     98 Echo (ping) reply    id=0x0b50, seq=16/4096, ttl=63 (request in 47)
```

*Figura 10- Experiência 3 TUX 32 ping TUX 33*

```
18 34.020801598  HewlettPacka_61:24:… Broadcast            ARP      60 Who has 172.16.31.253? Tell 172.16.31.1
19 34.020824157  Netronix_b4:b8:94    HewlettPacka_61:24:… ARP      42 172.16.31.253 is at 00:e0:7d:b4:b8:94
20 34.020938208  172.16.31.1          172.16.30.1          ICMP     98 Echo (ping) request  id=0x0f0d, seq=1/256, ttl=64 (reply in 21)
21 34.021253681  172.16.30.1          172.16.31.1          ICMP     98 Echo (ping) reply    id=0x0f0d, seq=1/256, ttl=63 (request in 20)
22 34.037306560  Routerboardc_1c:a3:… Spanning-tree-(for-… STP      60 RST. Root = 32768/0/c4:ad:34:1c:a3:30  Cost = 0  Port = 0x8001
23 35.024163897  172.16.31.1          172.16.30.1          ICMP     98 Echo (ping) request  id=0x0f0d, seq=2/512, ttl=64 (reply in 24)
24 35.024355612  172.16.30.1          172.16.31.1          ICMP     98 Echo (ping) reply    id=0x0f0d, seq=2/512, ttl=63 (request in 23)
```

*Figura 11- Experiência 3 TUX 32 ping TUX 33 – captura no TUX 34 eth1*

```
21 40.027572362  HewlettPacka_5a:74:… Broadcast            ARP      42 Who has 172.16.30.1? Tell 172.16.30.254
22 40.027724266  HewlettPacka_5a:7d:… HewlettPacka_5a:74:… ARP      60 172.16.30.1 is at 00:21:5a:5a:7d:b7
23 40.027741587  172.16.31.1          172.16.30.1          ICMP     98 Echo (ping) request  id=0x0f0d, seq=1/256, ttl=63 (reply in 24)
24 40.027862063  172.16.30.1          172.16.31.1          ICMP     98 Echo (ping) reply    id=0x0f0d, seq=1/256, ttl=64 (request in 23)
25 40.043884492  Routerboardc_1c:a3:… Spanning-tree-(for-… STP      60 RST. Root = 32768/0/c4:ad:34:1c:a3:2a  Cost = 0  Port = 0x8001
26 41.030806013  172.16.31.1          172.16.30.1          ICMP     98 Echo (ping) request  id=0x0f0d, seq=2/512, ttl=63 (reply in 27)
27 41.030955962  172.16.30.1          172.16.31.1          ICMP     98 Echo (ping) reply    id=0x0f0d, seq=2/512, ttl=64 (request in 26)
```

*Figura 12- Experiência 3 TUX 32 ping TUX 33 – captura no TUX 34 eth0*

```
    20 34.020938208  172.16.31.1          172.16.30.1          ICMP     98 Echo (ping) request  id=0x0f0d, seq=1/256, ttl=64 (reply in 21)
    21 34.021253681  172.16.30.1          172.16.31.1          ICMP     98 Echo (ping) reply    id=0x0f0d, seq=1/256, ttl=63 (request in 20)
    22 34.037306560  Routerboardc_1c:a3:… Spanning-tree-(for-… STP      60 RST. Root = 32768/0/c4:ad:34:1c:a3:30  Cost = 0  Port = 0x8001
    23 35.024163897  172.16.31.1          172.16.30.1          ICMP     98 Echo (ping) request  id=0x0f0d, seq=2/512, ttl=64 (reply in 24)
    24 35.024355612  172.16.30.1          172.16.31.1          ICMP     98 Echo (ping) reply    id=0x0f0d, seq=2/512, ttl=63 (request in 23)
    25 36.039513545  Routerboardc_1c:a3:… Spanning-tree-(for-… STP      60 RST. Root = 32768/0/c4:ad:34:1c:a3:30  Cost = 0  Port = 0x8001
    26 36.048112054  172.16.31.1          172.16.30.1          ICMP     98 Echo (ping) request  id=0x0f0d, seq=3/768, ttl=64 (reply in 27)
    27 36.048266823  172.16.30.1          172.16.31.1          ICMP     98 Echo (ping) reply    id=0x0f0d, seq=3/768, ttl=63 (request in 26)
    28 37.072141088  172.16.31.1          172.16.30.1          ICMP     98 Echo (ping) request  id=0x0f0d, seq=4/1024, ttl=64 (reply in 29)

> Frame 20: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface eth1, id 0
v Ethernet II, Src: HewlettPacka_61:24:01 (00:21:5a:61:24:01), Dst: Netronix_b4:b8:94 (00:e0:7d:b4:b8:94)
  > Destination: Netronix_b4:b8:94 (00:e0:7d:b4:b8:94)
  > Source: HewlettPacka_61:24:01 (00:21:5a:61:24:01)
    Type: IPv4 (0x0800)
```

*Figura 13- Experiência 3 TUX 32 ping TUX 33 – captura no TUX 34 eth1 endereços MAC*

```
    24 40.027862063  172.16.30.1          172.16.31.1          ICMP     98 Echo (ping) reply    id=0x0f0d, seq=1/256, ttl=64 (request in 23)
    25 40.043884492  Routerboardc_1c:a3:… Spanning-tree-(for-… STP      60 RST. Root = 32768/0/c4:ad:34:1c:a3:2a  Cost = 0  Port = 0x8001
    26 41.030806013  172.16.31.1          172.16.30.1          ICMP     98 Echo (ping) request  id=0x0f0d, seq=2/512, ttl=63 (reply in 27)
    27 41.030955962  172.16.30.1          172.16.31.1          ICMP     98 Echo (ping) reply    id=0x0f0d, seq=2/512, ttl=64 (request in 26)
    28 42.046085470  Routerboardc_1c:a3:… Spanning-tree-(for-… STP      60 RST. Root = 32768/0/c4:ad:34:1c:a3:2a  Cost = 0  Port = 0x8001
    29 42.054747884  172.16.31.1          172.16.30.1          ICMP     98 Echo (ping) request  id=0x0f0d, seq=3/768, ttl=63 (reply in 30)
    30 42.054871713  172.16.30.1          172.16.31.1          ICMP     98 Echo (ping) reply    id=0x0f0d, seq=3/768, ttl=64 (request in 29)
    31 43.078781737  172.16.31.1          172.16.30.1          ICMP     98 Echo (ping) request  id=0x0f0d, seq=4/1024, ttl=63 (reply in 32)
    32 43.078905775  172.16.30.1          172.16.31.1          ICMP     98 Echo (ping) reply    id=0x0f0d, seq=4/1024, ttl=64 (request in 31)
    33 44.048275693  Routerboardc_1c:a3:… Spanning-tree-(for-… STP      60 RST. Root = 32768/0/c4:ad:34:1c:a3:2a  Cost = 0  Port = 0x8001
    34 44.102756294  172.16.31.1          172.16.30.1          ICMP     98 Echo (ping) request  id=0x0f0d, seq=5/1280, ttl=63 (reply in 35)

> Frame 24: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface eth0, id 0
v Ethernet II, Src: HewlettPacka_5a:7d:b7 (00:21:5a:5a:7d:b7), Dst: HewlettPacka_5a:74:3e (00:21:5a:5a:74:3e)
  > Destination: HewlettPacka_5a:74:3e (00:21:5a:5a:74:3e)
  > Source: HewlettPacka_5a:7d:b7 (00:21:5a:5a:7d:b7)
    Type: IPv4 (0x0800)
```

*Figura 14- Experiência 3 TUX 32 ping TUX 33 – captura no TUX 34 eth0 endereços MAC*

```
12 8.437335091   172.16.30.1          172.16.30.254        ICMP     98 Echo (ping) request  id=0x223e, seq=4/1024, ttl=64 (reply in 13)
13 8.437470932   172.16.30.254        172.16.30.1          ICMP     98 Echo (ping) reply    id=0x223e, seq=4/1024, ttl=64 (request in 12)
14 9.461328313   172.16.30.1          172.16.30.254        ICMP     98 Echo (ping) request  id=0x223e, seq=5/1280, ttl=64 (reply in 15)
15 9.461461500   172.16.30.254        172.16.30.1          ICMP     98 Echo (ping) reply    id=0x223e, seq=5/1280, ttl=64 (request in 14)
```

*Figura 15- Experiência 4 TUX 33 ping TUX 34 – captura no TUX 33*

```
12 8.189260083   172.16.30.1          172.16.31.1          ICMP     98 Echo (ping) request  id=0x2295, seq=4/1024, ttl=64 (reply in 13)
13 8.189510467   172.16.31.1          172.16.30.1          ICMP     98 Echo (ping) reply    id=0x2295, seq=4/1024, ttl=63 (request in 12)
14 9.213262114   172.16.30.1          172.16.31.1          ICMP     98 Echo (ping) request  id=0x2295, seq=5/1280, ttl=64 (reply in 15)
15 9.213565926   172.16.31.1          172.16.30.1          ICMP     98 Echo (ping) reply    id=0x2295, seq=5/1280, ttl=63 (request in 14)
```

*Figura 16- Experiência 4 TUX 33 ping TUX 32 – captura no TUX 33*

```
 9 10.128843228  172.16.30.1          172.16.2.254         ICMP     98 Echo (ping) request  id=0x23cd, seq=2/512, ttl=64 (reply in 10)
10 10.129280576  172.16.2.254         172.16.30.1          ICMP     98 Echo (ping) reply    id=0x23cd, seq=2/512, ttl=62 (request in 9)
11 11.156835074  172.16.30.1          172.16.2.254         ICMP     98 Echo (ping) request  id=0x23cd, seq=3/768, ttl=64 (reply in 12)
12 11.157261247  172.16.2.254         172.16.30.1          ICMP     98 Echo (ping) reply    id=0x23cd, seq=3/768, ttl=62 (request in 11)
```

*Figura 16- Experiência 4 TUX 33 ping Router do laboratório – captura no TUX 33*

```
 5 2.776272077   172.16.31.1          172.16.30.1          ICMP     98 Echo (ping) request  id=0x1fd7, seq=2/512, ttl=64 (reply in 7)
 6 2.776433481   172.16.31.254        172.16.31.1          ICMP    126 Redirect             (Redirect for host)
 7 2.776620727   172.16.30.1          172.16.31.1          ICMP     98 Echo (ping) reply    id=0x1fd7, seq=2/512, ttl=63 (request in 5)
 8 3.542883395   172.16.31.1          193.136.152.71       NTP      90 NTP Version 4, client
 9 3.552030200   193.136.152.71       172.16.31.1          NTP      90 NTP Version 4, server
10 3.800277592   172.16.31.1          172.16.30.1          ICMP     98 Echo (ping) request  id=0x1fd7, seq=3/768, ttl=64 (reply in 12)
11 3.800413853   172.16.31.254        172.16.31.1          ICMP    126 Redirect             (Redirect for host)
```

*Figura 17- Experiência 4 TUX 32 ping TUX 33 após remoção da rota entre TUX 32 e TUX 34– captura no TUX 32*

```
root@gnu32:~# traceroute -n 172.16.30.1
traceroute to 172.16.30.1 (172.16.30.1), 30 hops max, 60 byte packets
 1  172.16.31.254  6.628 ms  6.627 ms  6.641 ms
 2  172.16.31.253  6.775 ms  6.761 ms  6.746 ms
 3  172.16.30.1  7.028 ms  7.017 ms  7.001 ms
root@gnu32:~# route add -net 172.16.30.0/24 gw 172.16.31.253
root@gnu32:~# traceroute -n 172.16.30.1
traceroute to 172.16.30.1 (172.16.30.1), 30 hops max, 60 byte packets
 1  172.16.31.253  0.193 ms  0.169 ms  0.151 ms
 2  172.16.30.1  0.355 ms  0.339 ms  0.323 ms
root@gnu32:~#
```

*Figura 18- Experiência 4 traceroute no TUX 32 antes e após a reinserção da rota entre TUX 32 e TUX 34*

| | | | | | |
|---|---|---|---|---|---|
| 9 10.128843228 | 172.16.30.1 | 172.16.2.254 | ICMP | 98 Echo (ping) request | id=0x23cd, seq=2/512, ttl=64 (reply in 10) |
| 10 10.129280576 | 172.16.2.254 | 172.16.30.1 | ICMP | 98 Echo (ping) reply | id=0x23cd, seq=2/512, ttl=62 (request in 9) |
| 11 11.156835074 | 172.16.30.1 | 172.16.2.254 | ICMP | 98 Echo (ping) request | id=0x23cd, seq=3/768, ttl=64 (reply in 12) |
| 12 11.157261247 | 172.16.2.254 | 172.16.30.1 | ICMP | 98 Echo (ping) reply | id=0x23cd, seq=3/768, ttl=62 (request in 11) |

*Figura 19- Experiência 4 TUX 33 ping router do laboratório antes de desligar a funcionalidade NAT*

| | | | | | |
|---|---|---|---|---|---|
| 10 11.580031625 | 172.16.30.1 | 172.16.2.254 | ICMP | 98 Echo (ping) request | id=0x246d, seq=2/512, ttl=64 (no response found!) |
| 11 12.604026929 | 172.16.30.1 | 172.16.2.254 | ICMP | 98 Echo (ping) request | id=0x246d, seq=3/768, ttl=64 (no response found!) |
| 12 13.155123170 | Routerboardc_1c:a3:… | Spanning-tree-(for-… | STP | 60 RST. Root = 32768/0/c4:ad:34:1c:a3:2a | Cost = 0 Port = 0x8001 |
| 13 13.628032080 | 172.16.30.1 | 172.16.2.254 | ICMP | 98 Echo (ping) request | id=0x246d, seq=4/1024, ttl=64 (no response found!) |
| 14 14.652042680 | 172.16.30.1 | 172.16.2.254 | ICMP | 98 Echo (ping) request | id=0x246d, seq=5/1280, ttl=64 (no response found!) |

*Figura 20- Experiência 4 TUX 33 ping router do laboratório após desligar a funcionalidade NAT no router da bancada*

| | | | | |
|---|---|---|---|---|
| 2 1.528438688 | 172.16.31.1 | 172.16.2.1 | DNS | 70 Standard query 0x9a52 A google.com |
| 3 1.528449792 | 172.16.31.1 | 172.16.2.1 | DNS | 70 Standard query 0x995b AAAA google.com |
| 4 1.529167414 | 172.16.2.1 | 172.16.31.1 | DNS | 86 Standard query response 0x9a52 A google.com A 142.250.184.174 |
| 5 1.529183198 | 172.16.2.1 | 172.16.31.1 | DNS | 98 Standard query response 0x995b AAAA google.com AAAA 2a00:1450:4003:80c::200e |
| 6 1.529471853 | 172.16.31.1 | 142.250.184.174 | ICMP | 98 Echo (ping) request id=0x23db, seq=1/256, ttl=64 (reply in 7) |
| 7 1.544315133 | 142.250.184.174 | 172.16.31.1 | ICMP | 98 Echo (ping) reply id=0x23db, seq=1/256, ttl=108 (request in 6) |
| 8 1.544412073 | 172.16.31.1 | 172.16.2.1 | DNS | 88 Standard query 0xfff1 PTR 174.184.250.142.in-addr.arpa |
| 9 1.545071656 | 172.16.2.1 | 172.16.31.1 | DNS | 127 Standard query response 0xfff1 PTR 174.184.250.142.in-addr.arpa PTR mad07s23-in-f14.1e100.net |

*Figura 21- Experiência 5 TUX 32 ping google.com*

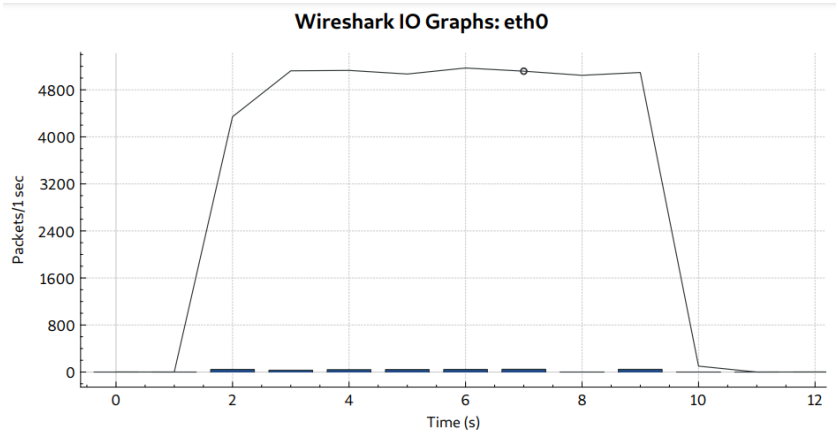


*Figura 22- Experiência 6 pacotes por segundo durante uma transferência*

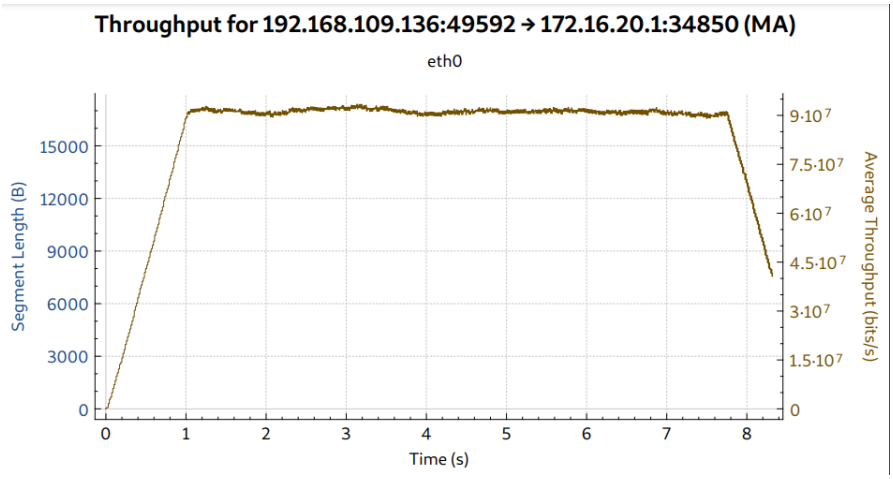


*Figura 23- Experiência 6 bites por segundo durante uma transferência*

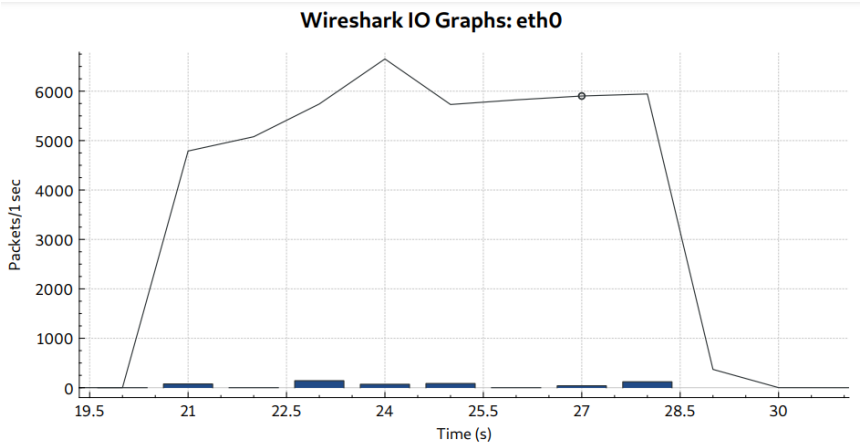


*Figura 24- Experiência 6 pacotes por segundo durante duas transferências em simultâneo*

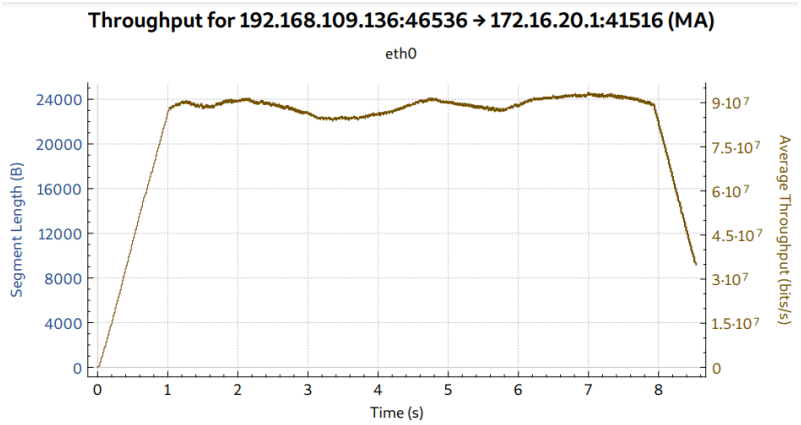


*Figura 25- Experiência 6 bits por segundo durante duas transferências em simultâneo*

# Anexos 2 – Comandos

## 2.1 Experiência 1

TUX33:

Ifconfig eth0 up

Ifconfig eth0 172.16.30.1/24

TUX34:

Ifconfig eth0 up

Ifconfig eth0 172.16.30.254/24

Ping 172.16.30.1 [Conexão verificada]

Arp -d 172.16.30.1 [eliminar TUX33 da tabela MAC]

Arp -a [verificação da tabela MAC]

Ping 172.16.30.1

## 2.2 Experiência 2

Consola switch:

/system reset-configuration

TUX 32:

Ifconfig eth0 up

Ifconfig eth0 172.16.31.1/24

Consola switch:

/interface bridge add name=bridge30

/interface bridge add name=bridge31

/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=bridge30 interface=ether1

/interface bridge port add bridge=bridge30 interface=ether2

/interface bridge port add bridge=bridge31 interface=ether3

/interface bridge port print [verificar que as conexões foram bem feitas]

TUX33:

Ping 172.16.31.1 [ping ao TUX32]

Ping 172.16.30.254 [ping ao TUX34]

Ping -b 172.16.30.255 [Broadcast]

TUX32:

Ping -b 172.16.31.255

## 2.3 Experiência 3

TUX34:

Ifconfig eth1 up

Ifconfig eth1 172.16.31.253/24

Consola switch:

/interface bridge port remove [find interface=ether4]

/interface bridge port add bridge=bridge31 interface=ether4

TUX34:

sysctl net.ipv4.ip_forward=1

sysctl net.ipv4.icmp_echo_ignore_broadcasts=0

TUX 32:

Route add -net 172.16.30.0/24 gw 172.16.31.253

TUX33:

Route add -net 172.16.31.0/24 gw 172.16.30.254

TUX32:

Route -n

ping 172.16.30.254

ping 172.16.31.253

ping 172.16.30.1

arp -d 172.16.30.254

arp -d 172.16.31.253

arp -d 172.16.30.1

TUX 33:

arp -d 172.16.30.254

arp -d 172.16.31.253

arp -d 172.16.31.1

TUX34:

arp -d 172.16.30.1

arp -d 172.16.31.1

TUX32:

Ping 172.16.30.1

## 2.4 Experiência 4:

Consola switch:

Interface bridge port remove [find interface=ether5]

Interface bridge port add bridge=bridge31 interface=ether5

Consola Router:

System reset-configuration

ip address add address=172.16.2.59/24 interface=ether1

ip address add address=172.16.31.254/24 interface=ether2

ip route add dst-address=172.16.30.0/24 gateway=172.16.31.523

ip route add dst-address=0.0.0.0/0 gateway=172.16.2.524

TUX32:

Route add default gw 172.16.31.254

TUX33:

Route add default gw 172.16.30.254

TUX34:

Route add default gw 172.16.31.254

TUX33:

Ping 172.16.30.254

Ping 172.16.31.1

Ping 172.16.31.254

TUX32:

Sysctl net.ipv4.conf.eth0.accept_redirects=0

Sysctl net.ipv4.conf.all.accept_redirects=0

Route del -net 172.16.30.0 gw 172.16.31.523 netmask 255.255.255.0

Ping 172.16.30.1

Traceroute -n 172.16.30.1

Route add -net 172.16.30.0/24 gw 172.16.31.253

Traceroute -n 172.16.30.1

Sysctl net.ipv4.conf.eth0.accept_redirects=1

Sysctl net.ipv4.conf.all.accept_redirects=1

TUX33:

Ping 172.16.1.254

Consola Router:

Ip firewall nat disable 0

TUX33:

Ping 172.16.1.254

Consola Router:

Ip firewall nat enable 0

## 2.5 Experiência 5:

TUX32, 33, 34:

Ping goole.com

## Anexos 3 – Código

### 3.1 – Utils.h

```c
#include <arpa/inet.h>
#include <errno.h>
#include <fcntl.h>
#include <netdb.h>
#include <netinet/in.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>

#define MAX_LEN 256
#define h_addr h_addr_list[0] // The first address in h_addr_list.
#define SERVER_PORT 21

/**
 * Struct used to store fields passed in arguments
*/
struct url {
    char user[50];
    char password[50];
    char filepath[100];
    char host[50];
};

/**
 * Parses arguments of a "ftp://[<user>:<password>@]<host>/<file-path>" string
*/
int parse_url(char* arguments, struct url* url);

int get_port(char* str);

int create_socket(char* ip, int port);

int get_file_size(char* response);

int download_file(int file_size, int data_socket_fd, char* filepath);

void progress_bar(float percentage);

char readFromSocket(FILE* fp);

```

### 3.2 – Utils.c

```c
#include "utils.h"

#define STYLE_BOLD "\033[1m"
#define STYLE_NO_BOLD "\033[22m"

char readFromSocket(FILE* fp) {
    char buf[MAX_LEN];
    do {
        fgets(buf, MAX_LEN - 1, fp);
        printf("%s", buf);
    } while (buf[3] == '-');

    return buf[0];
}
```

```c

int create_socket(char* ip, int port) {
    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 an TCP socket

    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        perror("socket()");
        exit(0);
    }

    /* Connect to the server */
    if (connect(sockfd, (struct sockaddr*) &server_addr, sizeof(server_addr)) < 0) {
        perror("connect()");
        exit(0);
    }

    return sockfd;
}

int get_port(char* str) {
    int port;
    str += 27; // Skip useless info

    char* token = strtok(str, ",");

    for (int i = 0; i < 4; i++)
        token = strtok(NULL, ",");

    port = atoi(token) * 256;
    token = strtok(NULL, ",");
    port += atoi(token);

    printf("\nPort : %d\n\n", port);
    return port;
}

int parse_url(char* arguments, struct url* url) {
    strcpy(url->user, "anonymous"); // Assume anon
    strcpy(url->password, "");      // Assume anon

    // Get protocol name

    if (strncmp(arguments, "ftp://", 6)) {
        printf("ERROR: Couldn't parse the protocol name\n");
        return -1;
    }

    arguments += 6; // Skip ftp://
    char* token;

    // Get user
    if (strstr(arguments, "@") == NULL) {
        puts("No username found. Assuming anonymous user.");
        strcpy(url->user, "anonymous"); // Assume anon

        // Get host
        token = strtok(arguments, "/");
    } else {
        if (strstr(arguments, ":") == NULL) { // No password provided
            token = strtok(arguments, "@");
```

```
            strcpy(url->user, token);
            strcpy(url->password, "");
        }

        else {
            token = strtok(arguments, ":");
            strcpy(url->user, token);
            token = strtok(NULL, "@");
            strcpy(url->password, token);
        }

        // Get host
        token = strtok(NULL, "/");
    }

    if (token == NULL) {
        printf("ERROR: Couldn't parse the host\n");
        return -1;
    }

    strcpy(url->host, token);

    // Get file path

    token = strtok(NULL, "");

    if (token == NULL) {
        printf("ERROR: Couldn't parse the filepath.\n");
        return -1;
    }

    strcpy(url->filepath, token);

    return 0;
}

int get_file_size(char* response) {
    char* s = strrchr(response, '(') + 1;
    int file_size = atoi(strtok(s, " "));

    return file_size;
}

int download_file(int file_size, int data_socket_fd, char* filepath) {
    char buf[MAX_LEN];
    int bytes;
    float total_bytes = 0, percentage = 0;

    char* filename = strrchr(filepath, '/');

    if (filename == NULL)
        filename = filepath;
    else
        filename += 1;

    FILE* file = fopen(filename, "w");
    if (file == NULL) {
        printf("ERROR: Failed to open file.\n");
        exit(1);
    }

    printf("\nStarting to download %s\n\n", filename);
    progress_bar(percentage);
    while ((bytes = read(data_socket_fd, buf, sizeof(buf))) > 0) {
        bytes = fwrite(buf, 1, bytes, file);
        total_bytes += bytes;
        percentage = (float) total_bytes / file_size * 100;
        progress_bar(percentage);
    }
    printf("\nFinished downloading %s\n\n", filename);
```

```
153.
154.     fclose(file);
155.
156.     return 0;
157. }
158.
159. void progress_bar(float percentage) {
160.     printf("[");
161.
162.     for (int i = 0; i < 50; i++) {
163.         if (percentage / 2 < i)
164.             printf(" ");
165.
166.         else
167.             printf("x");
168.     }
169.
170.     printf("] %.2f%% complete\n", percentage);
171. }
172.
173.
174.
```

## 3.3 – Download.c

```
 1. #include "utils.h"
 2.
 3. int main(int argc, char** argv) {
 4.     if (argc != 2) {
 5.         printf("Usage download ftp://[<user>:<password>@]<host>/<url-path>\n");
 6.         exit(1);
 7.     }
 8.
 9.     // Parse & store arguments
10.     struct url url;
11.     if (parse_url(argv[1], &url) < 0) {
12.         puts("Aborting\n");
13.         exit(1);
14.     }
15.
16.     struct hostent* h;
17.     if ((h = gethostbyname(url.host)) == NULL) {
18.         herror("gethostbyname");
19.         exit(1);
20.     }
21.     // Get IP
22.     char* address = inet_ntoa(*((struct in_addr*) h->h_addr));
23.
24.     // Print info
25.     printf("User:       %s\n", url.user);
26.     puts("Password:   *****");
27.     printf("Host:       %s\n", url.host);
28.     printf("Filepath:        %s\n", url.filepath);
29.     printf("Host name:  %s\n", h->h_name);
30.     printf("IP Address: %s\n", address);
31.     printf("Port:       %d\n\n", SERVER_PORT);
32.
33.     // Open socket
34.     int sockfd = create_socket(address, SERVER_PORT);
35.     FILE* fp = fdopen(sockfd, "r");
36.
37.     // Welcome Message
38.     if (readFromSocket(fp) != '2') {
39.         printf("ERROR: Error in connection.\n");
```

```
        exit(1);
    }

    /* Login
        user username\n
        pass password\n
    */

    // Send User
    char buf[MAX_LEN];
    sprintf(buf, "user %s\n", url.user);
    if (write(sockfd, buf, strlen(buf)) < 0) {
        printf("ERROR: Failed to send user.\n");
        exit(1);
    }

    // Read response
    char res = readFromSocket(fp);

    // Check if it server sent "331 Please specify the password".
    if (res == '3') {
        //Check if password is set
        if (!strcmp(url.password, "")) {
            if (strcmp(url.user, "anonymous")) {
                // Ask for user password
                printf("\nPlease input a password: ");
                char pass[MAX_LEN];
                fgets(pass, sizeof(pass), stdin);
                strcpy(url.password, pass);
            }
        }
        // Send password
        sprintf(buf, "pass %s\n", url.password);
        if (write(sockfd, buf, strlen(buf)) < 0) {
            printf("ERROR: Failed to send password.\n");
            exit(1);
        }
        // Read response
        if (readFromSocket(fp) != '2') {
            printf("ERROR: Login was not successful.\n");
            exit(1);
        }
    } else if (res != '2') {
        printf("ERROR: Failed to send user.\n");
        exit(1);
    }

    // Enter passive mode
    if (write(sockfd, "pasv\n", 5) < 0) {
        printf("ERROR: Failed to send pasv.\n");
        exit(1);
    }
    // Read response
    do {
        fgets(buf, MAX_LEN - 1, fp);
        printf("%s", buf);
    } while (buf[3] == '-');

    if (buf[0] != '2') {
        printf("ERROR: Failed to enter passive mode.\n");
        exit(1);
    }
    // Open socket
    int data_socket_fd = create_socket(address, get_port(buf));
    // Send retr command
    sprintf(buf, "retr %s\n", url.filepath);
    if (write(sockfd, buf, strlen(buf)) < 0) {
        printf("ERROR: Failed to send retr command.\n");
        exit(1);
    }
```

```c
    // Read response
    do {
        fgets(buf, MAX_LEN - 1, fp);
        printf("%s", buf);
    } while (buf[3] == '-');

    if (buf[0] != '2' && buf[0] != '1') {
        printf("ERROR: Failed to retrieve file.\n");
        exit(1);
    }

    download_file(get_file_size(buf), data_socket_fd, url.filepath);
    close(data_socket_fd);
    close(sockfd);

    return 0;
}

```

### 3.4 – Makefile

```
# Compiler and linker
CC = gcc

# Flags
CFLAGS = -Wall -g

main:
	@gcc ${CFLAGS} -o download download.c utils.c -lm

clean:
	@rm -f download

download: clean main
	@./download "ftp://ftp:frp@ftp.up.pt/pub/ubuntu/project/ubuntu-archive-keyring.gpg.sig"

```