2º Trabalho Laboratorial - Redes de Computadores - Turma 6 Grupo 1

André Gomes
up201806224@fe.up.pt

Filipe Recharte
up201806224@fe.up.pt

2 de janeiro de 2021

Resumo

Neste relatório são especificados os passos tomados para formar uma rede de computadores que possibilite, usando 3 computadores, 1 switch, 1 router e ligação á internet, fazer download de um ficheiro, com recurso a um programa criado por este grupo com uso de tecnologia FTP e TCP/IP, em qualquer um dos computadores.

Este relatório começa por especificar a aplicação de download criada, passando pelos seguintes módulos: args, na qual é feito o parse dos argumentos do programa, connection, que contém as funções de ligação a sockets e de troca de informação e por fim download, que contém o procedimento para fazer o download de um ficheiro, com recurso aos módulos anteriores.

Após isto é feita uma descrição e análise de cada uma das experiências: Experiência 1 consiste em criar uma rede com 2 computadores usando um switch, com o objetivo de conhecer endereços IP e MAC, pacotes ARP e comandos ping. Na experiência 2 é acrescentado um computador e são criadas 2 VLAN's dento do switch para mostrar que apenas computadores dentro da mesma VLAN é que conseguem comunicar entre si. Retira-se desta experiência conhecimento de como configurar uma VLAN e informação relativa a broadcast domains. Na terceira experiência um dos computadores é configurado como um router para estabelecer a comunicação entre as 2 VLAN's, com o objetivo de saber o que são e como se configuram routes nestes computadores. Na quarta experiência é configurado um router e é realizada a implementação da funcionalidade NAT, com o objetivo de preparar o acesso á internet nas próximas experiências. Na experiência 5 configura-se um servidor DNS em cada tux para ser feita a tradução de endereços normais para endereços IP e por fim, na experiência 6 é testado o programa de download desenvolvido anteriormente na rede de computadores feita até este momento para verificar que tanto o programa foi corretamente desenvolvido como a rede está corretamente implementada.

Conteúdo

1 Introdução

Neste relatório é explorado o processo de criação de uma aplicação de download de ficheiros usando o protocolo FTP e TCP/IP juntamente com o desenvolvimento de uma rede de computadores na qual será testada a aplicação.

Este projeto teve como objetivos ensinar o uso correto do protocolo FTP para realizar a transferência de um ficheiro, as funcionalidades de um switch e de um router e como fazer a configuração de uma simples rede de computadores.

2 Aplicação de Download

2.1 Arquitectura

A aplicação de Download está dividida em três partes: Download, Args e Connection.

2.1.1 Args

Neste módulo é feito o processamento dos argumentos da aplicação download. A partir da função parseArgs() é guardada a informação da URL dos argumentos do download para uma instância da struct args. No inicio da função são guardados os seguintes campos

- Utilizador
- Palavra passe
- Host
- Path

No final são usadas as funções getIP() e getFileName() para, respetivamente, a partir do URL, usando a função gethostbyname(), saber o endereço IP do Host e o nome do Host, e para extrair, do último elemento do path, o nome do ficheiro que se vai descarregar, para no final do programa, ao criar o ficheiro transferido, saber que nome lhe atribuir.

2.1.2 Connection

Neste módulo temos como primeira função init(), que é encarregada de, ao receber um endereço IP e uma porta, criar um socket e iniciar a ligação a esse socket.

A função sendCommand() recebe um file descriptor de um socket e uma string com o comando a ser enviado. Com recurso á função send() de sockets, o comando é enviado para o servidor.

As funções readResponse() e readResponsePassive() usam getline() para esvaziar o buffer que contém a resposta do servidor. Em readResponse() há uma pequena verificação do código de resposta para, caso haja algum erro, o programa termine em vez de ficar bloqueado. Em readResponsePassive() é feito o processamento da resposta com o envio do commando pasv. A função retira da mensagem de resposta um endereço IP e uma porta que retorna a partir dos argumentos.

A função saveFile() está encarregue de criar o ficheiro que irá transferir para a máquina e preencher o ficheiro com os dados que recebe do servidor. Enquanto é feita a leitura dos dados, estes são adicionados ao ficheiro sequencialmente, até o buffer de leitura estar vazio.

2.1.3 Download

O módulo de download apenas contém a função main() com os passos necessários para fazer o download de um ficheiro.

O programa começa por verificar se apenas foi passado um argumento a partir da linha de comandos, caso isto não se verifique, é apresentada uma mensagem de erro e o programa termina. Se o programa receber um URL, este é passado por argumento para a função parseArgs() que preenche a struct args com os componentes do URL.

Uma lista de detalhes sobre a ligação é imprimida na consola e o programa continua a sua execução, chamando o predicado init(), que cria um socket para conseguir comunicar com o servidor. O programa procede com uma invocação de readResponse() para ler a mensagem de ligação e esvaziar o buffer de leitura.

Nesta altura inicia-se a sequência de login [3], na qual o programa envia o comando "user ."com um nome de utilizador, lê a resposta com readResponse() e executa as mesmas duas funções com o comando "pass .". Caso não seja especificadas credenciais de login, o nome de utilizador usado é "anonymous"e a palavra passe usada é "1234". O programa executa o envio da palavra passe mesmo que não seja necessário, isto não afeta a troca de mensagens e simplifica o código.

o próximo comando a enviar é "pasv"para pedir ao servidor que transfira dados em modo passivo, o que faz com que seja necessário abrir outra ligação. A resposta a este comando é lida na função readResponsePassive() e contém o endereço IP e a porta á qual se realizará a nova ligação. Esta é feita via init() e devolve um novo file descriptor do socket de onde se retirará os dados.

Para finalizar é enviado o comando "retr [path]"com o caminho para o ficheiro e o programa chama a função saveFile() para guardar num ficheiro os dados lidos a partir do novo socket.

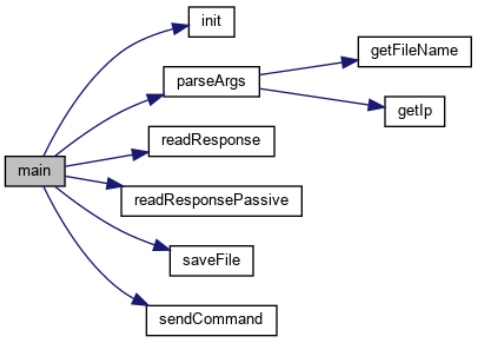


Figura 1: Gráfico de chamadas de função para o download

3 Configuração e Análise da Rede

3.1 Experiência nº 1

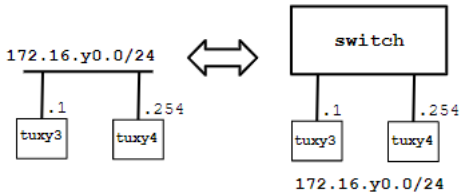


Figura 2: Arquitetura da Primeira Experiência

Esta experiência tem como objetivos perceber o que são pacotes ARP e para que servem, que tipo de pacotes é que o comando ping gera e o que são endereços MAC e endereços IP.

Os comandos usados para esta experiência podem ser encontrados no Anexo B.1.

3.1.1 Análise dos Logs

Pacotes ARP [1] são usados para, sabendo o endereço IP [2] de uma máquina, pedir o endereço MAC [3]. Os endereços MAC são os identificadores das placas de rede enquanto que os endereços IP servem como identificadores públicos que cada máquina necessita de usar numa rede para poder comunicar com outras máquinas. Uma máquina pode possuir vários endereços IP, mas apenas 1 endereço MAC.

O comando ping gera pacotes ICMP [4]. Estes pacotes são normalmente usados por routers ou hosts para mandar erros da camada 3 ou mensagens de controlo para outros routers ou hosts. Nesta experiência o comando ping serve para descobrir se existe conectividade entre computadores.

Ao analisar os logs desta experiência é possível verificar o formato dos pacotes ARP que são enviados quando se faz ping (Figura 9), juntamente com os pacotes ICMP de ping (Figura 10). O Wireshark atribui cores diferentes a tipos de pacotes diferentes, mas é possível ver o tipo de pacote nos detalhes dos mesmos (Figuras 11 e 12).

Para ver o tamanho de uma trama que é recebida, a partir do wireshark, verifica-se o valor em bytes passados "on wire", ou então, para tramas IPv4, estas possuem um campo "Total Length"com o valor em bytes do tamanho da trama.

[1] Adress Resolution Protocol
[2] Internet Protocol
[3] Medium Access Control
[4] Internet Control Message Protocol

3.2 Experiência nº 2

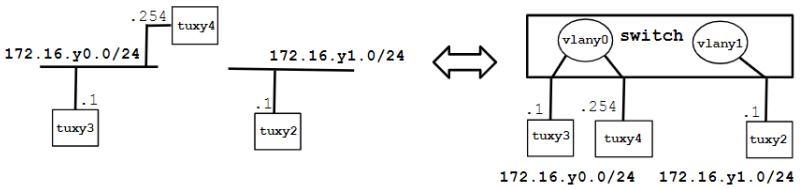


Figura 3: Arquitetura da Segunda Experiência

Esta experiência tem como objetivo a criação de duas VLAN's no switch e perceber a conectividade entre os tuxs, depois de os configurar em cada uma das sub-redes.

Os comandos usados para esta experiência podem ser encontrados no Anexo B.2.

3.2.1 Análise dos Logs

Para configurar as VLAN's criámos a VLAN 20 e 21 e associamos à primeira, os tux's 23 e 24 e à segunda o tux 22, com o objetivo de obter a arquitetura da figura 4. Para testar a conectividade entre os tux's foi feito ping do tux23 até o tux24 que ocorreu com sucesso como seria de esperar uma vez que se encontram na mesma sub-rede, como se pode comprovar na Figura 13.

Quanto à conexão entre o tux23 e o tux22, o ping não obteve resposta devido ao facto de não haver nenhuma rota entre as VLAN's, sendo impossível o tux23 chegar à interface de rede do tux22.

Também no tux23 foi feito ping em broadcast, ping -b 172.16.20.255, que não obteve resposta, como demonstra o registo da Figura 14. Aqui, seria expectável uma resposta do tux24 uma vez que se encontram na mesma sub-rede, mas tal não acontece porque echo-ignore-broadcast está ativado por default para evitar grandes amplificações de tráfego. No entanto, os logs realizados no tux24 (Figura 15) comprovam que este recebeu um pedido do tux23 por broadcast.

Repetimos o processo anteriormente descrito mas agora a partir do tux22. Mais uma vez não obtivemos resposta, mas a justificação é diferente. Neste caso não foi obtida nenhuma resposta pois não está configurado mais nenhum dispositivo na VLAN 21 a não ser o próprio tux22.

Assim, podemos concluir que exitem dois domínios de broadcast que correspondem às sub-redes VLAN 20 e VLAN 21 com os endereços 172.16.20.255 e 172.16.21.255, respetivamente.

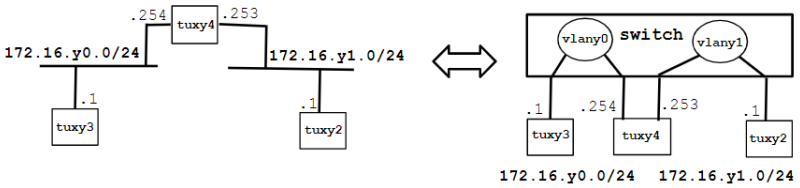


Figura 4: Arquitetura da Terceira Experiência

3.3 Experiência nº 3

Esta experiência tem como objectivo tornar o tuxy4 num router para possibilitar a comunicação entre o tuxy3 e o tuxy2, através das VLAN's 0 e 1. Para haver comunicação terá de se configurar os endereços IP's das portas ethernet dos tuxy's e as routes que serão usadas.

Os comandos usados para esta experiência podem ser encontrados no Anexo B.3.

3.3.1 Análise dos Logs

Para adicionar uma route a um dos tuxes é necessário ter três valores: IP da rede a aceder, máscara de bits desse IP e IP da porta a usar como gateway. Para haver ligação entre o tux23 e o tux22, adicionou-se uma route ao tux23 para aceder aos endereços 172.16.21.0/24 a partir do IP 72.16.20.254 (tux24 eth0) e uma route ao tux22 para aceder aos endereços 172.16.20.0/24 a partir do IP 172.16.21.253 (tux24 eth1). Estas routes podem ser vistas na forwarding table, através do comando: route -n.

Com estas routes definidas, é possivel fazer ping, a partir do tux23, a todas as interfaces dos outros tux's (Figura 16). Também se verifica na figura que a interface eth0 do tux24 enviou 2 pedidos ARP para determinar o endereço MAC da interface eth0 to tux23, enquanto que o tux23 mandou um pedido para saber o endereço MAC da interface eth0 do tux24.

Continuando nas figuras 17 e 18, é possível verificar que existe comunicação entre os tux's 23 e 22, visto que os pings emitidos obtêm uma resposta. Na primeira figura nota-se uma troca de mensagens ARP para o tux23 tomar conhecimento do endereço MAC da interface eth0 to tux24 e vice-versa, enquanto que na segunda figura existe uma troca de mensgens ARP para o tux22 tomar conhecimento do endereço MAC da interface eth1 do tux24 e vice-versa.

3.4 Experiência nº 4

Esta experiência tem como objetivo estabelecer uma ligação com a rede dos laboratórios e implementar rotas num router comercial, adicionar-lhe funcionalidade NAT, e perceber qual a sua função.

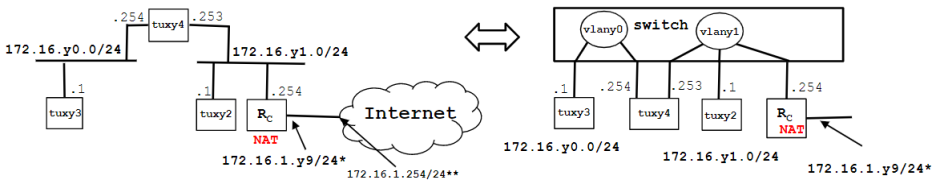


Figura 5: Arquitetura da Quarta Experiência

Os comandos usados para esta experiência podem ser encontrados no Anexo B.4.

3.4.1 Análise dos Logs

O NAT (Network Address Translation), tal como o nome indica, é um mecanismo implementado em routers que substitui os endereços IP locais nos pacotes por um endereço IP público de forma a se conseguir estabelecer uma ligação para fora da rede. Sendo assim, o router que implementa o NAT torna-se responsável por encaminhar todos os pacotes para o endereço correto, dentro ou fora da rede local.

Nesta experiência, começámos por configurar a interface GE 0/0 do router [1] atribuída à VLAN 21. Para a interface GE 0/1 do router, atribuiu-se o IP 172.16.1.29 para que fosse feita a ligação com a rede dos laboratórios.

Definiu-se que o tux24 serviria de router para o tux23 e o router RC para o tux22 e tux24. Adicionalmente, foram adicionadas as devidas rotas estáticas no router RC, com as instruções descritas no passo 1 do Anexo B.4.

Após estas configurações foi possível realizar o ping do tux23 a todos os outros pontos da nossa rede como a Figura 19 mostra. A conexão do tux23 às interfaces dos tux's já era possível nas experiências anteriores, agora é possível também aceder às interfaces GE 0/0 e GE 0/1 do router RC, sendo isto possível uma vez que, como referido anteriormente, se adicionaram duas rotas no router RC, a default gateway com o IP 172.16.1.29 e o reencaminhamento de pacotes para a rede com IP 172.16.20.0/24 (VLAN 20 onde se encontra o tux23) através da interface eth1 do tux24 com IP 172.16.21.253.

Até ao momento a conexão do tux22 à interface eth0 do tux23 é efetuada através da rota implementada no tux22 e descrita na experiência 3. Para verificar esta implementação foi removida essa mesma rota do tux22, e executou-se o comando traceroute onde se comprova que, como não havia nenhuma rota definida até à VLAN 20, o router com o IP 172.16.21.254, definido como default gateway do tux22 ficou responsável por redirecionar os pacotes ICMP até ao destino como mostra a Figura 21 e comprovam os logs da Figura 22, voltando a adicionar a rota e fazendo um novo traceroute notamos que os pacotes deixam de passar pelo router e passam a seguir o "melhor caminho"via tux24, como mostra a Figura 20.

Por fim, tentamos desde o tux23 fazer ping do router do laboratório

com o IP 172.16.1.254, sem sucesso uma vez que o NAT ainda não tinha sido adicionado ao router. Após a adição do mesmo [2], com os comandos delineados no passo 6 do Anexo B.4, voltamos a tentar o passo anterior, com sucesso uma vez que agora, o NAT permite que os dispositivos conectados à rede local, 172.16.2x.0/24 (interface 0/0), comuniquem com a rede externa, 172.16.1.29 (interface 0/1), como comprova a Figura 23.

3.5 Experiência nº 5

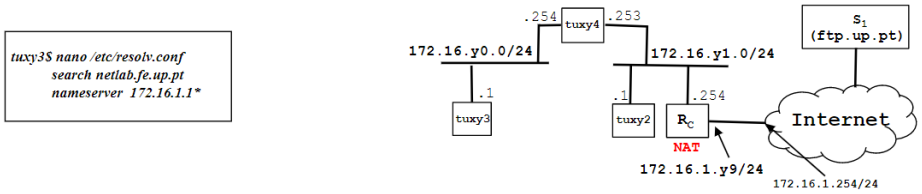


Figura 6: Arquitetura da Quinta Experiência

Esta experiência tem como objectivo saber como configurar um serviço DNS numa máquina e descobrir que pacotes são trocados pelo serviço de DNS, para além da informação contida nos mesmos.

Os comandos usados para esta experiência podem ser encontrados no Anexo B.5.

3.5.1 Análise dos Logs

Esta experiência contém apenas um log (Figura 24) da qual se vão retirar maior parte das conclusões.

Para configurar um serviço de DNS numa máquina só é necessário adicionar uma linha ao ficheiro "etc/resolv.conf"contendo o nome do servidor a usar e o endereço IP do mesmo. Após ter o DNS configurado, aparecerão pacotes relativos ao DNS nos logs.

Ao realizar um ping são enviados 2 pacotes DNS com dois pedidos: Adress Mapping Record (A), para pedir o endereço IPv4 do host e IP Version 6 Address Record (AAAA) para pedir o endereço IPv6. Ambos estes pedidos recebem respostas: Para "ftp.up.pt"o endereço IPv4 é 193.137.29.15 e o endereço IPv6 é 2001:690:2200:1200::15.

Após um ping ser enviado com sucesso e obter resposta, é enviado outro pedido DNS, agora do tipo Reverse-lookup Pointer Record (PTR) para, a partir do endereço IP obtido anteriormente, encontrar todos os host names, que no caso anterior, devolve "mirrors.up.pt". Na figura 24 foram feitos pings para "ftp.up.pt"e para "google.com".

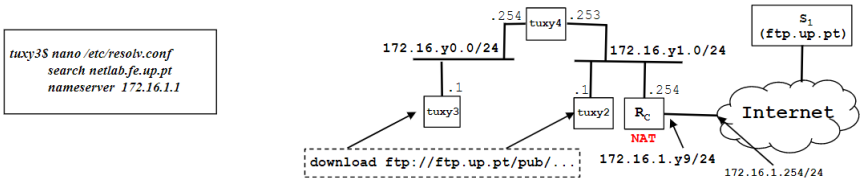


Figura 7: Arquitetura da Sexta Experiência

3.6 Experiência nº 6

Os comandos usados para esta experiência podem ser encontrados no Anexo B.6.

Nesta experiência, a aplicação que elaborámos, foi compilada e executada. Durante a execução, são abertas duas conexões TCP. Uma, para a porta 21, utilizada para controlo, e outra, para uma porta a ser definida, utilizada para todo o transporte de dados. Para iniciar uma conexão TCP, em primeiro lugar, abre-se uma nova porta e, de seguida, é feita a conexão ao servidor usando essa mesma porta.

Foi feita a conexão no modo passivo, escolhido na criação da aplicação, pois permite identificar a porta de retorno de maneira direta e, assim, iniciar uma nova ligação para a transferência binária. Após a criação da nova ligação de dados é enviado o comando retr direcionado ao ficheiro a transferir. Recebe-se de seguida a FTP-DATA, que indica o tamanho em bytes das tramas. Ao fim de cada trama é enviada uma mensagem ACK do cliente para o servidor para confirmar a receção dos dados. Ao chegar ao fim do ficheiro o servidor manda uma mensagem de sucesso com o código 226 para confirmar o fim da transferência do ficheiro. Todos estes passos descritos são visíveis nos logs da Figura 27.

O ARQ TCP é um mecanismo de controlo de erros na transmissão de dados onde pacotes de controlo ACK's são enviados pelo recetor, indicando a correta receção de uma trama de dados ou pacote. Os campos de maior relevância são o "sequence number field" e o "ACK number field". O primeiro identifica o byte que representa o início da mensagem no pacote de dados, e o segundo identifica o número de sequência do próximo byte que o emissor do ACK espera receber em seguida.

Relativamente ao controlo de congestão, foi possível verificar que, durante a transmissão de dados, o recetor envia em cada mensagem ACK o tamanho da janela TCP que informa quantos bytes podem ser recebidos. Esta janela pode ficar cheia caso o cliente não esteja a ler dados de forma rápida o suficiente, sendo que quando isto acontece o recetor envia uma mensagem ACK com o bit window full ativo, fazendo com que o emissor pare de enviar dados temporariamente como se verifica nos logs da Figura 26. Caso o emissor não respeite estas indicações, os bytes excedentes são descartados

levando a retransmissões desnecessárias. Quando o receptor conclui a leitura de todos os bytes recebidos no pacote TCP este envia um TCP window update, indicando que está pronto para receber mais dados, esta mensagem é visível nos logs da Figura 25.

Teoricamente, ao aumentar o número de conexões TCP a largura de banda disponível para cada uma das conexões diminui, reduzindo consequentemente o throughput. Na prática, o gráfico que obtivemos relativo ao throughput (Figura 8) não foi de encontro ao esperado uma vez que não é notório no gráfico uma queda quando se iniciou o segundo download e consequentemente uma reposição do throughput quando o mesmo termina. Este resultado pode dever-se ao facto de o throughput se encontrar bastante abaixo da capacidade máxima de ligação fastethernet, fazendo com que não seja visível esta variação. Outra possibilidade é o facto de o tamanho dos pacotes lidos na nossa aplicação ser de apenas 1 byte, o que pode ter limitado a velocidade do download, estando o bottle neck no processo de leitura do ficheiro e não na largura de banda disponível na conexão TCP. Outros fatores podem ter influenciado este gráfico uma vez que se nota uma quebra perto do segundo 100, que nada tem a ver com os downloads efetuados por nós.

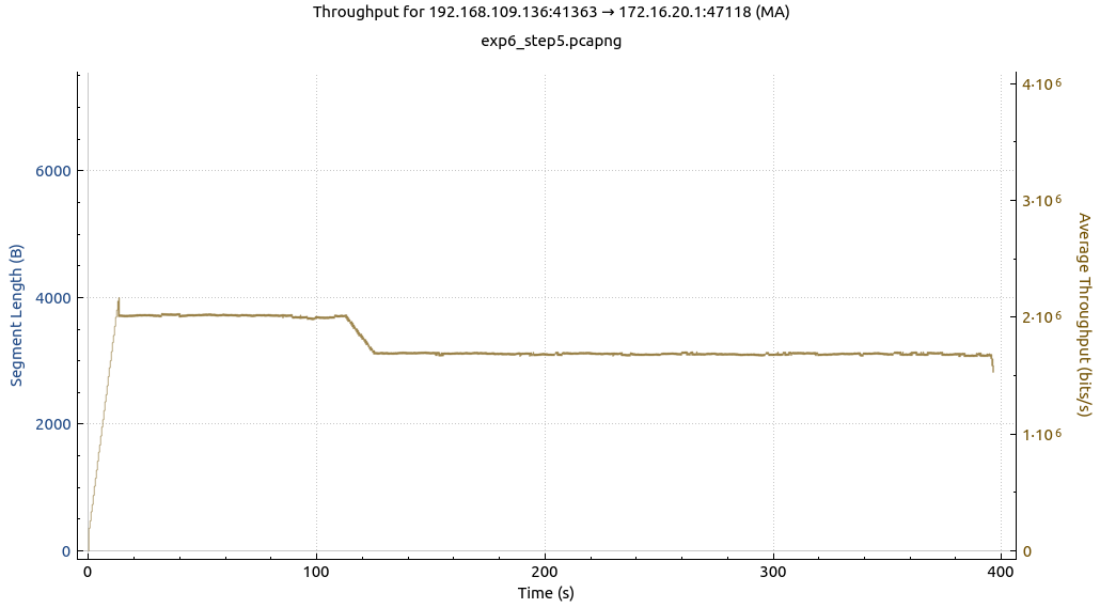


Figura 8: Gráfico de throughput do download do ficheiro crab.mp4 na experiência 6

4 Conclusão

Os objectivos deste projecto foram alcançados. A partir desta sequência de experiências e criação de aplicação de download foi possível obter conhecimento sobre o protocolo FTP e programação sobre sockets, configuração de uma rede de computadores composta por computadores, switches e routers, endereços IP endereços MAC, routes, ARP, NAT e mais especificidades não mencionadas.

Foi possível verificar os benefícios aliados à interligação de redes e como isto permite comunicações entre dispositivos que não estão ligados diretamente um ao outro. A configuração gradual da rede permitiu-nos, de forma fácil, perceber os mecanismos inerentes a esta e o seu impacto nas comunicações atuais. Tivemos oportunidade de trabalhar as nossas competências ao nível do desenho de redes e configuração dos dispositivos que as implementam. Considerámos este processo de aprendizagem muito enriquecedor para a nossa formação como engenheiros.

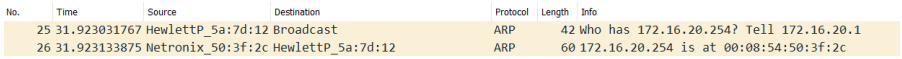
Referências

[1] Cisco Systems Inc. *Cisco 3900 Series, 2900 Series, and 1900 Series Integrated Services Routers Generation 2 Software Configuration Guide.* URL: `https://www.cisco.com/c/en/us/td/docs/routers/access/1900/software/configuration/guide/Software_Configuration.html`. (accessed: 22.12.2020).

[2] Cisco Systems Inc. *Configuring Network Address Translation: Getting Started.* URL: `https://www.cisco.com/c/en/us/support/docs/ip/network-address-translation-nat/13772-12.html`. (accessed: 22.12.2020).

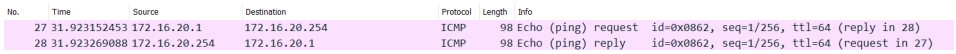
[3] J. Reynolds J. Postel. *FILE TRANSFER PROTOCOL (FTP).* URL: `https://tools.ietf.org/html/rfc959`. (accessed: 22.12.2020).

A Figuras



No.	Time	Source	Destination	Protocol	Length	Info
25	31.923031767	HewlettP_5a:7d:12	Broadcast	ARP	42	Who has 172.16.20.254? Tell 172.16.20.1
26	31.923133875	Netronix_50:3f:2c	HewlettP_5a:7d:12	ARP	60	172.16.20.254 is at 00:08:54:50:3f:2c

Figura 9: Pacotes ARP



No.	Time	Source	Destination	Protocol	Length	Info
27	31.923152453	172.16.20.1	172.16.20.254	ICMP	98	Echo (ping) request id=0x0862, seq=1/256, ttl=64 (reply in 28)
28	31.923269088	172.16.20.254	172.16.20.1	ICMP	98	Echo (ping) reply id=0x0862, seq=1/256, ttl=64 (request in 27)

Figura 10: Comando Ping em pacotes ICMP

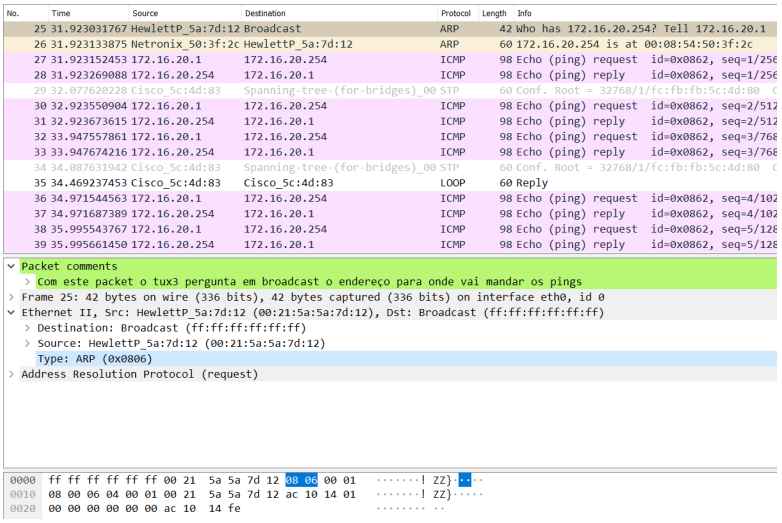


Figura 11: Determinação de pacotes do tipo ARP via wireshark

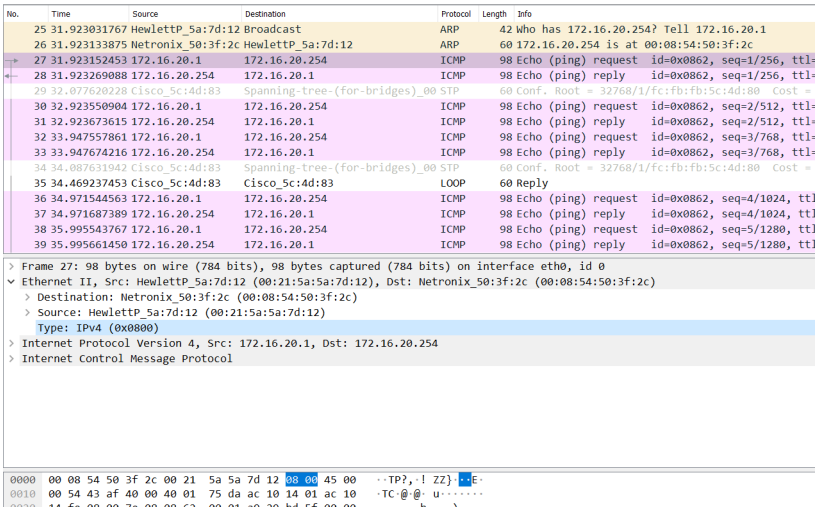


Figura 12: Determinação de pacotes do tipo ICMP via wireshark

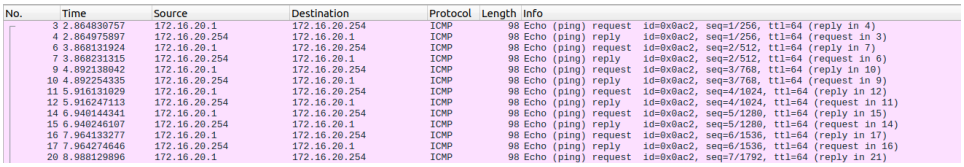


Figura 13: Excerto de logs de ping do tux24 a partir do tux23 na experiência 2

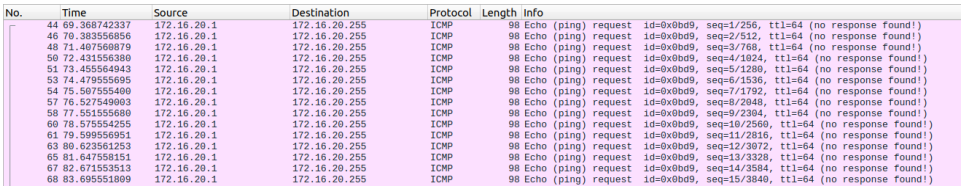


Figura 14: Excerto de logs de ping em broadcast no tux23 na experiência 2

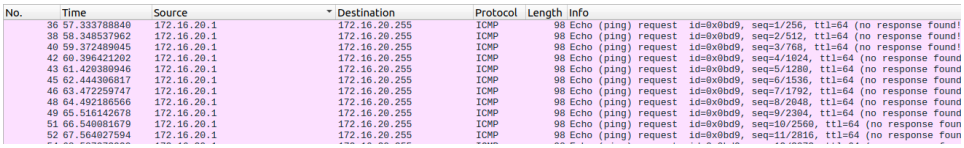
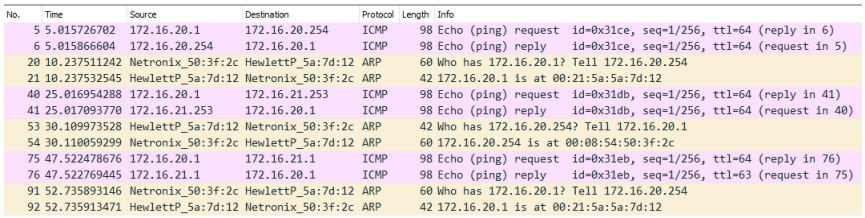
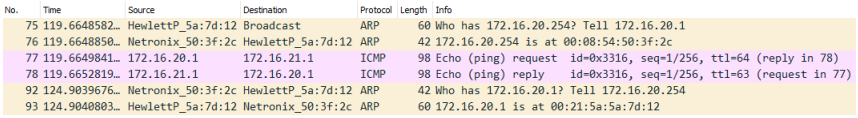


Figura 15: Excerto de logs no tux24 durante ping em broadcast no tux23 na experiência 2



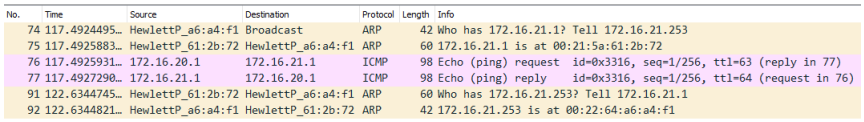
No.	Time	Source	Destination	Protocol	Length	Info
5	5.015726702	172.16.20.1	172.16.20.254	ICMP	98	Echo (ping) request id=0x31ce, seq=1/256, ttl=64 (reply in 6)
6	5.015866604	172.16.20.254	172.16.20.1	ICMP	98	Echo (ping) reply id=0x31ce, seq=1/256, ttl=64 (request in 5)
20	10.237511242	Netronix_50:3f:2c	HewlettP_5a:7d:12	ARP	60	Who has 172.16.20.1? Tell 172.16.20.254
21	10.237532545	HewlettP_5a:7d:12	Netronix_50:3f:2c	ARP	42	172.16.20.1 is at 00:21:5a:5a:7d:12
40	25.016954288	172.16.20.1	172.16.21.253	ICMP	98	Echo (ping) request id=0x31db, seq=1/256, ttl=64 (reply in 41)
41	25.017093770	172.16.21.253	172.16.20.1	ICMP	98	Echo (ping) reply id=0x31db, seq=1/256, ttl=64 (request in 40)
53	30.109973528	HewlettP_5a:7d:12	Netronix_50:3f:2c	ARP	42	Who has 172.16.20.254? Tell 172.16.20.1
54	30.110059299	Netronix_50:3f:2c	HewlettP_5a:7d:12	ARP	60	172.16.20.254 is at 00:08:54:50:3f:2c
75	47.522478676	172.16.20.1	172.16.21.1	ICMP	98	Echo (ping) request id=0x31eb, seq=1/256, ttl=64 (reply in 76)
76	47.522769445	172.16.21.1	172.16.20.1	ICMP	98	Echo (ping) reply id=0x31eb, seq=1/256, ttl=63 (request in 75)
91	52.735893146	Netronix_50:3f:2c	HewlettP_5a:7d:12	ARP	60	Who has 172.16.20.1? Tell 172.16.20.254
92	52.735913471	HewlettP_5a:7d:12	Netronix_50:3f:2c	ARP	42	172.16.20.1 is at 00:21:5a:5a:7d:12

Figura 16: Excerto de logs captados no tux23 na experiência 3



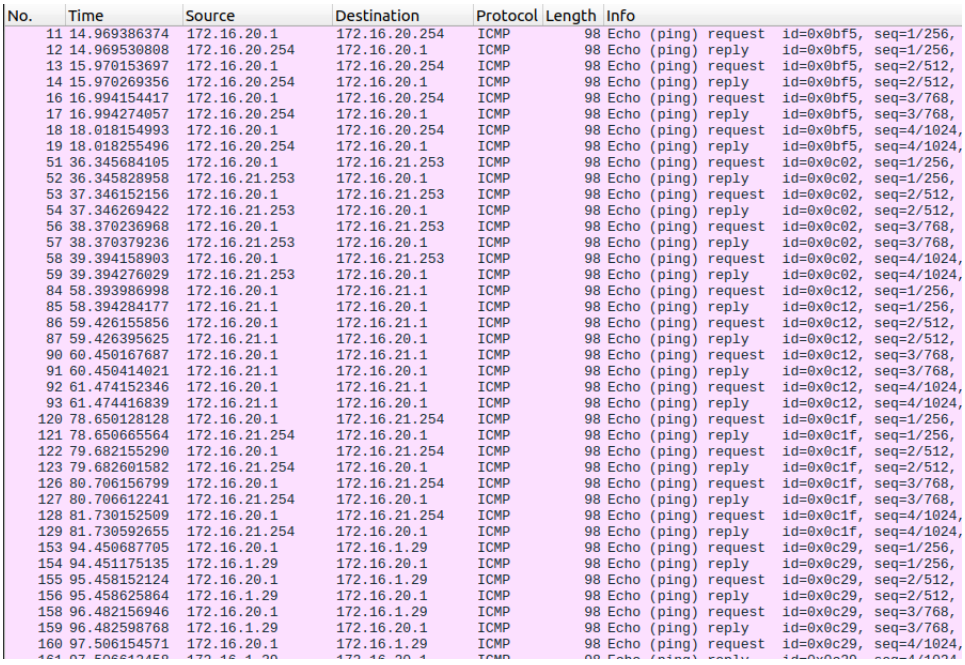
No.	Time	Source	Destination	Protocol	Length	Info
75	119.6648582…	HewlettP_5a:7d:12	Broadcast	ARP	60	Who has 172.16.20.254? Tell 172.16.20.1
76	119.6648850…	Netronix_50:3f:2c	HewlettP_5a:7d:12	ARP	42	172.16.20.254 is at 00:08:54:50:3f:2c
77	119.6649841…	172.16.20.1	172.16.21.1	ICMP	98	Echo (ping) request id=0x3316, seq=1/256, ttl=64 (reply in 78)
78	119.6652819…	172.16.21.1	172.16.20.1	ICMP	98	Echo (ping) reply id=0x3316, seq=1/256, ttl=63 (request in 77)
92	124.9039676…	Netronix_50:3f:2c	HewlettP_5a:7d:12	ARP	42	Who has 172.16.20.1? Tell 172.16.20.254
93	124.9040803…	HewlettP_5a:7d:12	Netronix_50:3f:2c	ARP	60	172.16.20.1 is at 00:21:5a:5a:7d:12

Figura 17: Excerto de logs captados no tux24 eth0 na experiência 3



No.	Time	Source	Destination	Protocol	Length	Info
74	117.4924495…	HewlettP_a6:a4:f1	Broadcast	ARP	42	Who has 172.16.21.1? Tell 172.16.21.253
75	117.4925883…	HewlettP_61:2b:72	HewlettP_a6:a4:f1	ARP	60	172.16.21.1 is at 00:21:5a:61:2b:72
76	117.4925931…	172.16.20.1	172.16.21.1	ICMP	98	Echo (ping) request id=0x3316, seq=1/256, ttl=63 (reply in 77)
77	117.4927290…	172.16.21.1	172.16.20.1	ICMP	98	Echo (ping) reply id=0x3316, seq=1/256, ttl=64 (request in 76)
91	122.6344745…	HewlettP_61:2b:72	HewlettP_a6:a4:f1	ARP	60	Who has 172.16.21.253? Tell 172.16.21.1
92	122.6344821…	HewlettP_a6:a4:f1	HewlettP_61:2b:72	ARP	42	172.16.21.253 is at 00:22:64:a6:a4:f1

Figura 18: Excerto de logs captados no tux24 eth1 na experiência 3



No.	Time	Source	Destination	Protocol	Length	Info
11	14.969386374	172.16.20.1	172.16.20.254	ICMP	98	Echo (ping) request id=0x0bf5, seq=1/256,
12	14.969530808	172.16.20.254	172.16.20.1	ICMP	98	Echo (ping) reply id=0x0bf5, seq=1/256,
13	15.970153697	172.16.20.1	172.16.20.254	ICMP	98	Echo (ping) request id=0x0bf5, seq=2/512,
14	15.970269356	172.16.20.254	172.16.20.1	ICMP	98	Echo (ping) reply id=0x0bf5, seq=2/512,
16	16.994154417	172.16.20.1	172.16.20.254	ICMP	98	Echo (ping) request id=0x0bf5, seq=3/768,
17	16.994274057	172.16.20.254	172.16.20.1	ICMP	98	Echo (ping) reply id=0x0bf5, seq=3/768,
18	18.018154993	172.16.20.1	172.16.20.254	ICMP	98	Echo (ping) request id=0x0bf5, seq=4/1024,
19	18.018255496	172.16.20.254	172.16.20.1	ICMP	98	Echo (ping) reply id=0x0bf5, seq=4/1024,
51	36.345684105	172.16.20.1	172.16.21.253	ICMP	98	Echo (ping) request id=0x0c02, seq=1/256,
52	36.345828958	172.16.21.253	172.16.20.1	ICMP	98	Echo (ping) reply id=0x0c02, seq=1/256,
53	37.346152156	172.16.20.1	172.16.21.253	ICMP	98	Echo (ping) request id=0x0c02, seq=2/512,
54	37.346269422	172.16.21.253	172.16.20.1	ICMP	98	Echo (ping) reply id=0x0c02, seq=2/512,
56	38.370236968	172.16.20.1	172.16.21.253	ICMP	98	Echo (ping) request id=0x0c02, seq=3/768,
57	38.370379236	172.16.21.253	172.16.20.1	ICMP	98	Echo (ping) reply id=0x0c02, seq=3/768,
58	39.394158903	172.16.20.1	172.16.21.253	ICMP	98	Echo (ping) request id=0x0c02, seq=4/1024,
59	39.394276029	172.16.21.253	172.16.20.1	ICMP	98	Echo (ping) reply id=0x0c02, seq=4/1024,
84	58.393986998	172.16.20.1	172.16.21.1	ICMP	98	Echo (ping) request id=0x0c12, seq=1/256,
85	58.394284177	172.16.21.1	172.16.20.1	ICMP	98	Echo (ping) reply id=0x0c12, seq=1/256,
86	59.426155856	172.16.20.1	172.16.21.1	ICMP	98	Echo (ping) request id=0x0c12, seq=2/512,
87	59.426395625	172.16.21.1	172.16.20.1	ICMP	98	Echo (ping) reply id=0x0c12, seq=2/512,
90	60.450167687	172.16.20.1	172.16.21.1	ICMP	98	Echo (ping) request id=0x0c12, seq=3/768,
91	60.450414021	172.16.21.1	172.16.20.1	ICMP	98	Echo (ping) reply id=0x0c12, seq=3/768,
92	61.474152346	172.16.20.1	172.16.21.1	ICMP	98	Echo (ping) request id=0x0c12, seq=4/1024,
93	61.474416839	172.16.21.1	172.16.20.1	ICMP	98	Echo (ping) reply id=0x0c12, seq=4/1024,
120	78.650128128	172.16.20.1	172.16.21.254	ICMP	98	Echo (ping) request id=0x0c1f, seq=1/256,
121	78.650665564	172.16.21.254	172.16.20.1	ICMP	98	Echo (ping) reply id=0x0c1f, seq=1/256,
122	79.682155290	172.16.20.1	172.16.21.254	ICMP	98	Echo (ping) request id=0x0c1f, seq=2/512,
123	79.682601582	172.16.21.254	172.16.20.1	ICMP	98	Echo (ping) reply id=0x0c1f, seq=2/512,
126	80.706156799	172.16.20.1	172.16.21.254	ICMP	98	Echo (ping) request id=0x0c1f, seq=3/768,
127	80.706612241	172.16.21.254	172.16.20.1	ICMP	98	Echo (ping) reply id=0x0c1f, seq=3/768,
128	81.730152509	172.16.20.1	172.16.21.254	ICMP	98	Echo (ping) request id=0x0c1f, seq=4/1024,
129	81.730592655	172.16.21.254	172.16.20.1	ICMP	98	Echo (ping) reply id=0x0c1f, seq=4/1024,
153	94.450687705	172.16.20.1	172.16.1.29	ICMP	98	Echo (ping) request id=0x0c29, seq=1/256,
154	94.451175135	172.16.1.29	172.16.20.1	ICMP	98	Echo (ping) reply id=0x0c29, seq=1/256,
155	95.458152124	172.16.20.1	172.16.1.29	ICMP	98	Echo (ping) request id=0x0c29, seq=2/512,
156	95.458625864	172.16.1.29	172.16.20.1	ICMP	98	Echo (ping) reply id=0x0c29, seq=2/512,
158	96.482156946	172.16.20.1	172.16.1.29	ICMP	98	Echo (ping) request id=0x0c29, seq=3/768,
159	96.482598768	172.16.1.29	172.16.20.1	ICMP	98	Echo (ping) reply id=0x0c29, seq=3/768,
160	97.506154571	172.16.20.1	172.16.1.29	ICMP	98	Echo (ping) request id=0x0c29, seq=4/1024,

Figura 19: Excerto de logs no teste de conectividade do tux23 a todos os outros IP's na experiência 4

```
root@tux22:~# traceroute 172.16.20.1
traceroute to 172.16.20.1 (172.16.20.1), 30 hops max, 60 byte packets
 1  172.16.21.253 (172.16.21.253)  0.181 ms  0.173 ms  0.157 ms
 2  172.16.20.1 (172.16.20.1)  0.271 ms  0.314 ms  0.298 ms
root@tux22:~#
```

Figura 20: traceroute após adicionar rota no tux22 até á interface eth0 do tux23 na experiência 4

```
root@tux22:~# traceroute 172.16.20.1
traceroute to 172.16.20.1 (172.16.20.1), 30 hops max, 60 byte packets
 1  172.16.21.254 (172.16.21.254)  0.482 ms  0.473 ms  0.525 ms
 2  172.16.21.253 (172.16.21.253)  0.612 ms  0.352 ms  0.350 ms
 3  172.16.20.1 (172.16.20.1)  0.433 ms  0.420 ms  0.401 ms
root@tux22:~#
```

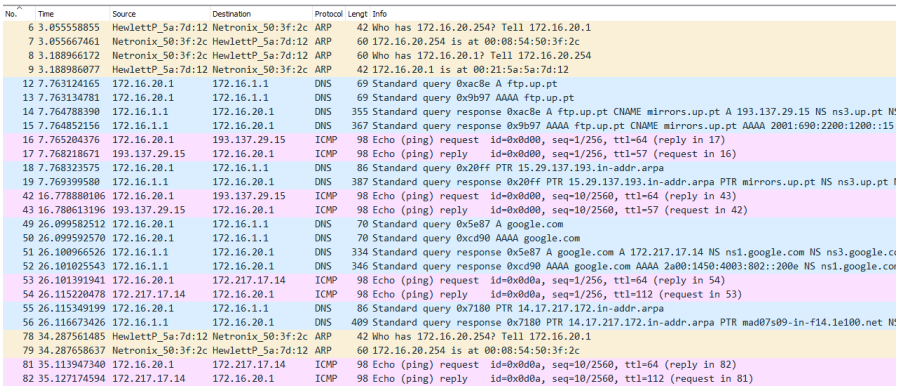
Figura 21: traceroute após remover rota no tux22 até á interface eth0 do tux23 na experiência 4

No.	Time	Source	Destination	Protocol	Length	Info
6	4.937818001	172.16.21.1	172.16.20.1	ICMP	98	Echo (ping) request id=0x0e24, seq=1/256, ttl=64 (repl
7	4.938167416	172.16.21.254	172.16.21.1	ICMP	70	Redirect (Redirect for host)
8	4.938390210	172.16.20.1	172.16.21.1	ICMP	98	Echo (ping) reply id=0x0e24, seq=1/256, ttl=63 (requ
9	5.967300381	172.16.21.1	172.16.20.1	ICMP	98	Echo (ping) request id=0x0e24, seq=2/512, ttl=64 (repl
10	5.967617181	172.16.21.254	172.16.21.1	ICMP	70	Redirect (Redirect for host)
11	5.967825448	172.16.20.1	172.16.21.1	ICMP	98	Echo (ping) reply id=0x0e24, seq=2/512, ttl=63 (requ
13	6.991301129	172.16.21.1	172.16.20.1	ICMP	98	Echo (ping) request id=0x0e24, seq=3/768, ttl=64 (repl
14	6.991598164	172.16.21.254	172.16.21.1	ICMP	70	Redirect (Redirect for host)
15	6.991823122	172.16.20.1	172.16.21.1	ICMP	98	Echo (ping) reply id=0x0e24, seq=3/768, ttl=63 (requ
16	8.015301527	172.16.21.1	172.16.20.1	ICMP	98	Echo (ping) request id=0x0e24, seq=4/1024, ttl=64 (rep
17	8.015599121	172.16.21.254	172.16.21.1	ICMP	70	Redirect (Redirect for host)
18	8.015795165	172.16.20.1	172.16.21.1	ICMP	98	Echo (ping) reply id=0x0e24, seq=4/1024, ttl=63 (rec
23	9.039308840	172.16.21.1	172.16.20.1	ICMP	98	Echo (ping) request id=0x0e24, seq=5/1280, ttl=64 (rep
24	9.039627316	172.16.21.254	172.16.21.1	ICMP	70	Redirect (Redirect for host)
25	9.039825945	172.16.20.1	172.16.21.1	ICMP	98	Echo (ping) reply id=0x0e24, seq=5/1280, ttl=63 (rec
29	10.063305536	172.16.21.1	172.16.20.1	ICMP	98	Echo (ping) request id=0x0e24, seq=6/1536, ttl=64 (rep
30	10.063622546	172.16.21.254	172.16.21.1	ICMP	70	Redirect (Redirect for host)
31	10.063815378	172.16.20.1	172.16.21.1	ICMP	98	Echo (ping) reply id=0x0e24, seq=6/1536, ttl=63 (rec
32	11.091301415	172.16.21.1	172.16.20.1	ICMP	98	Echo (ping) request id=0x0e24, seq=7/1792, ttl=64 (rep
33	11.091627923	172.16.21.254	172.16.21.1	ICMP	70	Redirect (Redirect for host)
34	11.091837517	172.16.20.1	172.16.21.1	ICMP	98	Echo (ping) reply id=0x0e24, seq=7/1792, ttl=63 (rec
36	12.115295318	172.16.21.1	172.16.20.1	ICMP	98	Echo (ping) request id=0x0e24, seq=8/2048, ttl=64 (rep

Figura 22: Excerto de logs após remover rota no tux22 ao fazer ping do tux23 na experiência 4

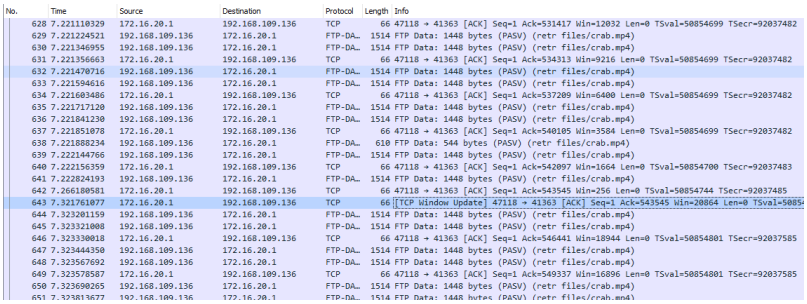
No.	Time	Source	Destination	Protocol	Length	Info
12	14.958350946	172.16.20.1	172.16.1.254	ICMP	98	Echo (ping) request id=0x0ae2, seq=1/256, ttl=64 (rep
13	14.959149245	172.16.1.254	172.16.20.1	ICMP	98	Echo (ping) reply id=0x0ae2, seq=1/256, ttl=62 (rec
14	15.971661977	172.16.20.1	172.16.1.254	ICMP	98	Echo (ping) request id=0x0ae2, seq=2/512, ttl=64 (rep
15	15.972248305	172.16.1.254	172.16.20.1	ICMP	98	Echo (ping) reply id=0x0ae2, seq=2/512, ttl=62 (rec
17	16.995654982	172.16.20.1	172.16.1.254	ICMP	98	Echo (ping) request id=0x0ae2, seq=3/768, ttl=64 (rep
18	16.996232788	172.16.1.254	172.16.20.1	ICMP	98	Echo (ping) reply id=0x0ae2, seq=3/768, ttl=62 (rec
19	18.019648744	172.16.20.1	172.16.1.254	ICMP	98	Echo (ping) request id=0x0ae2, seq=4/1024, ttl=64 (r
20	18.020251205	172.16.1.254	172.16.20.1	ICMP	98	Echo (ping) reply id=0x0ae2, seq=4/1024, ttl=62 (r
22	19.043652834	172.16.20.1	172.16.1.254	ICMP	98	Echo (ping) request id=0x0ae2, seq=5/1280, ttl=64 (r
23	19.044245726	172.16.1.254	172.16.20.1	ICMP	98	Echo (ping) reply id=0x0ae2, seq=5/1280, ttl=62 (r
25	20.067653421	172.16.20.1	172.16.1.254	ICMP	98	Echo (ping) request id=0x0ae2, seq=6/1536, ttl=64 (r
26	20.068244008	172.16.1.254	172.16.20.1	ICMP	98	Echo (ping) reply id=0x0ae2, seq=6/1536, ttl=62 (r
31	21.091651135	172.16.20.1	172.16.1.254	ICMP	98	Echo (ping) request id=0x0ae2, seq=7/1792, ttl=64 (r
32	21.092239487	172.16.1.254	172.16.20.1	ICMP	98	Echo (ping) reply id=0x0ae2, seq=7/1792, ttl=62 (r
34	22.115657569	172.16.20.1	172.16.1.254	ICMP	98	Echo (ping) request id=0x0ae2, seq=8/2048, ttl=64 (r
35	22.116235026	172.16.1.254	172.16.20.1	ICMP	98	Echo (ping) reply id=0x0ae2, seq=8/2048, ttl=62 (r
36	23.139654215	172.16.20.1	172.16.1.254	ICMP	98	Echo (ping) request id=0x0ae2, seq=9/2304, ttl=64 (r
37	23.140309337	172.16.1.254	172.16.20.1	ICMP	98	Echo (ping) reply id=0x0ae2, seq=9/2304, ttl=62 (r
39	24.163656369	172.16.20.1	172.16.1.254	ICMP	98	Echo (ping) request id=0x0ae2, seq=10/2560, ttl=64 (
40	24.164227262	172.16.1.254	172.16.20.1	ICMP	98	Echo (ping) reply id=0x0ae2, seq=10/2560, ttl=62 (

Figura 23: Excerto de logs captados no tux23 ao fazer ping do router do laboratório na experiência 4



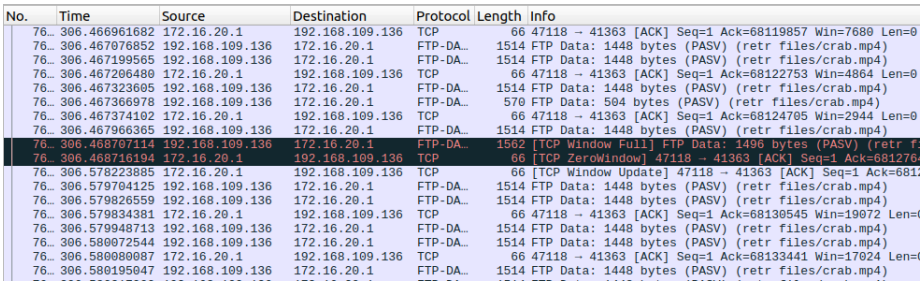
No.	Time	Source	Destination	Protocol	Lengt	Info
6	3.055558855	HewlettP_5a:7d:12	Netronix_50:3f:2c	ARP	42	Who has 172.16.20.254? Tell 172.16.20.1
7	3.055667461	Netronix_50:3f:2c	HewlettP_5a:7d:12	ARP	60	172.16.20.254 is at 00:08:54:50:3f:2c
8	3.188966172	Netronix_50:3f:2c	HewlettP_5a:7d:12	ARP	60	Who has 172.16.20.1? Tell 172.16.20.254
9	3.188986077	HewlettP_5a:7d:12	Netronix_50:3f:2c	ARP	42	172.16.20.1 is at 00:21:5a:5a:7d:12
12	7.763124165	172.16.20.1	172.16.1.1	DNS	69	Standard query 0xac8e A ftp.up.pt
13	7.763134781	172.16.20.1	172.16.1.1	DNS	69	Standard query 0x9b97 AAAA ftp.up.pt
14	7.764788390	172.16.1.1	172.16.20.1	DNS	355	Standard query response 0xac8e A ftp.up.pt CNAME mirrors.up.pt A 193.137.29.15 NS ns3.up.pt NS
15	7.764852156	172.16.1.1	172.16.20.1	DNS	367	Standard query response 0x9b97 AAAA ftp.up.pt CNAME mirrors.up.pt AAAA 2001:690:2200:1200::15
16	7.765204376	172.16.20.1	193.137.29.15	ICMP	98	Echo (ping) request id=0x0d00, seq=1/256, ttl=64 (reply in 17)
17	7.768218671	193.137.29.15	172.16.20.1	ICMP	98	Echo (ping) reply id=0x0d00, seq=1/256, ttl=57 (request in 16)
18	7.768323575	172.16.20.1	172.16.1.1	DNS	86	Standard query 0x20ff PTR 15.29.137.193.in-addr.arpa
19	7.769399580	172.16.1.1	172.16.20.1	DNS	387	Standard query response 0x20ff PTR 15.29.137.193.in-addr.arpa PTR mirrors.up.pt NS ns3.up.pt
42	16.778880106	172.16.20.1	193.137.29.15	ICMP	98	Echo (ping) request id=0x0d00, seq=10/2560, ttl=64 (reply in 43)
43	16.780613196	193.137.29.15	172.16.20.1	ICMP	98	Echo (ping) reply id=0x0d00, seq=10/2560, ttl=57 (request in 42)
49	26.099582512	172.16.20.1	172.16.1.1	DNS	70	Standard query 0x5e87 A google.com
50	26.099592570	172.16.20.1	172.16.1.1	DNS	70	Standard query 0xcd90 AAAA google.com
51	26.100966526	172.16.1.1	172.16.20.1	DNS	334	Standard query response 0x5e87 A google.com A 172.217.17.14 NS ns1.google.com NS ns3.google.co
52	26.101025543	172.16.1.1	172.16.20.1	DNS	346	Standard query response 0xcd90 AAAA google.com AAAA 2a00:1450:4003:802::200e NS ns1.google.com
53	26.101391941	172.16.20.1	172.217.17.14	ICMP	98	Echo (ping) request id=0x0d0a, seq=1/256, ttl=64 (reply in 54)
54	26.115220478	172.217.17.14	172.16.20.1	ICMP	98	Echo (ping) reply id=0x0d0a, seq=1/256, ttl=112 (request in 53)
55	26.115349199	172.16.20.1	172.16.1.1	DNS	86	Standard query 0x7180 PTR 14.17.217.172.in-addr.arpa
56	26.116673426	172.16.1.1	172.16.20.1	DNS	409	Standard query response 0x7180 PTR 14.17.217.172.in-addr.arpa PTR mad07s09-in-f14.1e100.net NS
78	34.287561485	HewlettP_5a:7d:12	Netronix_50:3f:2c	ARP	42	Who has 172.16.20.254? Tell 172.16.20.1
79	34.287658637	Netronix_50:3f:2c	HewlettP_5a:7d:12	ARP	60	172.16.20.254 is at 00:08:54:50:3f:2c
81	35.113947340	172.16.20.1	172.217.17.14	ICMP	98	Echo (ping) request id=0x0d0a, seq=10/2560, ttl=64 (reply in 82)
82	35.127174594	172.217.17.14	172.16.20.1	ICMP	98	Echo (ping) reply id=0x0d0a, seq=10/2560, ttl=112 (request in 81)

Figura 24: Excerto de logs captados no tux23 eth0 na experiência 5



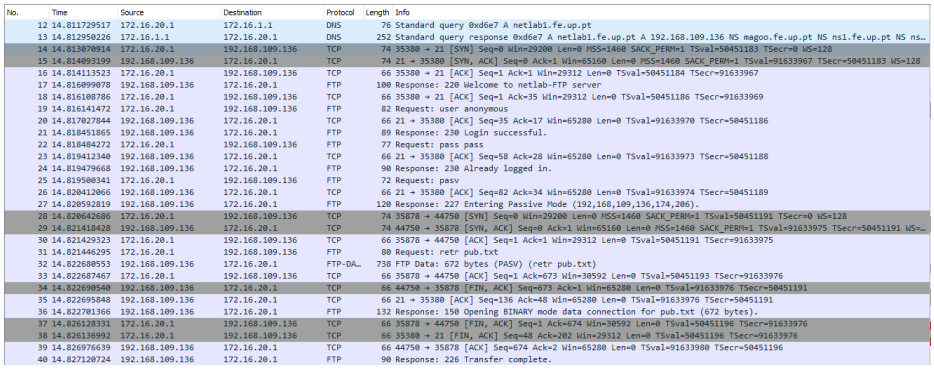
No.	Time	Source	Destination	Protocol	Length	Info
628	7.221110329	172.16.20.1	192.168.109.136	TCP	66	47118 → 41363 [ACK] Seq=1 Ack=531417 Win=12032 Len=0 TSval=50854699 TSecr=92037482
629	7.221224521	192.168.109.136	172.16.20.1	FTP-DA…	1514	FTP Data: 1448 bytes (PASV) (retr files/crab.mp4)
630	7.221346955	192.168.109.136	172.16.20.1	FTP-DA…	1514	FTP Data: 1448 bytes (PASV) (retr files/crab.mp4)
631	7.221356663	172.16.20.1	192.168.109.136	TCP	66	47118 → 41363 [ACK] Seq=1 Ack=534313 Win=9216 Len=0 TSval=50854699 TSecr=92037482
632	7.221470716	192.168.109.136	172.16.20.1	FTP-DA…	1514	FTP Data: 1448 bytes (PASV) (retr files/crab.mp4)
633	7.221594616	192.168.109.136	172.16.20.1	FTP-DA…	1514	FTP Data: 1448 bytes (PASV) (retr files/crab.mp4)
634	7.221603486	172.16.20.1	192.168.109.136	TCP	66	47118 → 41363 [ACK] Seq=1 Ack=537209 Win=6400 Len=0 TSval=50854699 TSecr=92037482
635	7.221717120	192.168.109.136	172.16.20.1	FTP-DA…	1514	FTP Data: 1448 bytes (PASV) (retr files/crab.mp4)
636	7.221841230	192.168.109.136	172.16.20.1	FTP-DA…	1514	FTP Data: 1448 bytes (PASV) (retr files/crab.mp4)
637	7.221851078	172.16.20.1	192.168.109.136	TCP	66	47118 → 41363 [ACK] Seq=1 Ack=540105 Win=3584 Len=0 TSval=50854699 TSecr=92037482
638	7.221888234	192.168.109.136	172.16.20.1	FTP-DA…	610	FTP Data: 544 bytes (PASV) (retr files/crab.mp4)
639	7.222144766	192.168.109.136	172.16.20.1	FTP-DA…	1514	FTP Data: 1448 bytes (PASV) (retr files/crab.mp4)
640	7.222156359	172.16.20.1	192.168.109.136	TCP	66	47118 → 41363 [ACK] Seq=1 Ack=542097 Win=1664 Len=0 TSval=50854700 TSecr=92037483
641	7.222824193	192.168.109.136	172.16.20.1	FTP-DA…	1514	FTP Data: 1448 bytes (PASV) (retr files/crab.mp4)
642	7.266180581	172.16.20.1	192.168.109.136	TCP	66	47118 → 41363 [ACK] Seq=1 Ack=543545 Win=256 Len=0 TSval=50854744 TSecr=92037485
643	7.321761077	172.16.20.1	192.168.109.136	TCP	66	[TCP Window Update] 47118 → 41363 [ACK] Seq=1 Ack=543545 Win=20864 Len=0 TSval=50854
644	7.323201159	192.168.109.136	172.16.20.1	FTP-DA…	1514	FTP Data: 1448 bytes (PASV) (retr files/crab.mp4)
645	7.323321008	192.168.109.136	172.16.20.1	FTP-DA…	1514	FTP Data: 1448 bytes (PASV) (retr files/crab.mp4)
646	7.323330018	172.16.20.1	192.168.109.136	TCP	66	47118 → 41363 [ACK] Seq=1 Ack=546441 Win=18944 Len=0 TSval=50854801 TSecr=92037585
647	7.323444350	192.168.109.136	172.16.20.1	FTP-DA…	1514	FTP Data: 1448 bytes (PASV) (retr files/crab.mp4)
648	7.323567692	192.168.109.136	172.16.20.1	FTP-DA…	1514	FTP Data: 1448 bytes (PASV) (retr files/crab.mp4)
649	7.323578587	172.16.20.1	192.168.109.136	TCP	66	47118 → 41363 [ACK] Seq=1 Ack=549337 Win=16896 Len=0 TSval=50854801 TSecr=92037585
650	7.323690265	192.168.109.136	172.16.20.1	FTP-DA…	1514	FTP Data: 1448 bytes (PASV) (retr files/crab.mp4)

Figura 25: Excerto de logs captados durante o download de um ficheiro na experiência 6



No.	Time	Source	Destination	Protocol	Length	Info
76…	306.466961682	172.16.20.1	192.168.109.136	TCP	66	47118 → 41363 [ACK] Seq=1 Ack=68119857 Win=7680 Len=0
76…	306.467076852	192.168.109.136	172.16.20.1	FTP-DA…	1514	FTP Data: 1448 bytes (PASV) (retr files/crab.mp4)
76…	306.467199565	192.168.109.136	172.16.20.1	FTP-DA…	1514	FTP Data: 1448 bytes (PASV) (retr files/crab.mp4)
76…	306.467206480	172.16.20.1	192.168.109.136	TCP	66	47118 → 41363 [ACK] Seq=1 Ack=68122753 Win=4864 Len=0
76…	306.467323605	192.168.109.136	172.16.20.1	FTP-DA…	1514	FTP Data: 1448 bytes (PASV) (retr files/crab.mp4)
76…	306.467366978	192.168.109.136	172.16.20.1	FTP-DA…	570	FTP Data: 504 bytes (PASV) (retr files/crab.mp4)
76…	306.467374102	172.16.20.1	192.168.109.136	TCP	66	47118 → 41363 [ACK] Seq=1 Ack=68124705 Win=2944 Len=0
76…	306.467966365	192.168.109.136	172.16.20.1	FTP-DA…	1514	FTP Data: 1448 bytes (PASV) (retr files/crab.mp4)
76…	306.468707114	192.168.109.136	172.16.20.1	FTP-DA…	1562	[TCP Window Full] FTP Data: 1496 bytes (PASV) (retr f
76…	306.468716194	172.16.20.1	192.168.109.136	TCP	66	[TCP ZeroWindow] 47118 → 41363 [ACK] Seq=1 Ack=681276
76…	306.578223885	172.16.20.1	192.168.109.136	TCP	66	[TCP Window Update] 47118 → 41363 [ACK] Seq=1 Ack=681
76…	306.579704125	192.168.109.136	172.16.20.1	FTP-DA…	1514	FTP Data: 1448 bytes (PASV) (retr files/crab.mp4)
76…	306.579826559	192.168.109.136	172.16.20.1	FTP-DA…	1514	FTP Data: 1448 bytes (PASV) (retr files/crab.mp4)
76…	306.579834381	172.16.20.1	192.168.109.136	TCP	66	47118 → 41363 [ACK] Seq=1 Ack=68130545 Win=19072 Len=
76…	306.579948713	192.168.109.136	172.16.20.1	FTP-DA…	1514	FTP Data: 1448 bytes (PASV) (retr files/crab.mp4)
76…	306.580072544	192.168.109.136	172.16.20.1	FTP-DA…	1514	FTP Data: 1448 bytes (PASV) (retr files/crab.mp4)
76…	306.580080087	172.16.20.1	192.168.109.136	TCP	66	47118 → 41363 [ACK] Seq=1 Ack=68133441 Win=17024 Len=
76…	306.580195047	192.168.109.136	172.16.20.1	FTP-DA…	1514	FTP Data: 1448 bytes (PASV) (retr files/crab.mp4)

Figura 26: Excerto de logs captados durante o download de um ficheiro na experiência 6



No.	Time	Source	Destination	Protocol	Length	Info
12	14.811729517	172.16.20.1	172.16.1.1	DNS	76	Standard query 0xd6e7 A netlab1.fe.up.pt
13	14.812950226	172.16.1.1	172.16.20.1	DNS	252	Standard query response 0xd6e7 A netlab1.fe.up.pt A 192.168.109.136 NS magoo.fe.up.pt NS ns1.fe.up.pt NS ns…
14	14.813070914	172.16.20.1	192.168.109.136	TCP	74	35380 → 21 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=50451183 TSecr=0 WS=128
15	14.814093199	192.168.109.136	172.16.20.1	TCP	74	21 → 35380 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 TSval=91633967 TSecr=50451183 WS=128
16	14.814113523	172.16.20.1	192.168.109.136	TCP	66	35380 → 21 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=50451184 TSecr=91633967
17	14.816099078	192.168.109.136	172.16.20.1	FTP	100	Response: 220 Welcome to netlab-FTP server
18	14.816108786	172.16.20.1	192.168.109.136	TCP	66	35380 → 21 [ACK] Seq=1 Ack=35 Win=29312 Len=0 TSval=50451186 TSecr=91633969
19	14.816141472	172.16.20.1	192.168.109.136	FTP	82	Request: user anonymous
20	14.817027844	192.168.109.136	172.16.20.1	TCP	66	21 → 35380 [ACK] Seq=35 Ack=17 Win=65280 Len=0 TSval=91633970 TSecr=50451186
21	14.818451865	192.168.109.136	172.16.20.1	FTP	89	Response: 230 Login successful.
22	14.818484272	172.16.20.1	192.168.109.136	FTP	77	Request: pass pass
23	14.819412340	192.168.109.136	172.16.20.1	TCP	66	21 → 35380 [ACK] Seq=58 Ack=28 Win=65280 Len=0 TSval=91633973 TSecr=50451188
24	14.819479668	192.168.109.136	172.16.20.1	FTP	90	Response: 230 Already logged in.
25	14.819500341	172.16.20.1	192.168.109.136	FTP	72	Request: pasv
26	14.820412066	192.168.109.136	172.16.20.1	TCP	66	21 → 35380 [ACK] Seq=82 Ack=34 Win=65280 Len=0 TSval=91633974 TSecr=50451189
27	14.820592819	192.168.109.136	172.16.20.1	FTP	120	Response: 227 Entering Passive Mode (192,168,109,136,174,206).
28	14.820642686	172.16.20.1	192.168.109.136	TCP	74	35878 → 44750 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=50451191 TSecr=0 WS=128
29	14.821419428	192.168.109.136	172.16.20.1	TCP	74	44750 → 35878 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 TSval=91633975 TSecr=50451191 WS=…
30	14.821429323	172.16.20.1	192.168.109.136	TCP	66	35878 → 44750 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=50451191 TSecr=91633975
31	14.821446295	172.16.20.1	192.168.109.136	FTP	80	Request: retr pub.txt
32	14.822680553	192.168.109.136	172.16.20.1	FTP-DA…	738	FTP Data: 672 bytes (PASV) (retr pub.txt)
33	14.822687467	172.16.20.1	192.168.109.136	TCP	66	35878 → 44750 [ACK] Seq=1 Ack=673 Win=30592 Len=0 TSval=50451193 TSecr=91633976
34	14.822690540	192.168.109.136	172.16.20.1	TCP	66	44750 → 35878 [FIN, ACK] Seq=673 Ack=1 Win=65280 Len=0 TSval=91633976 TSecr=50451191
35	14.822695848	192.168.109.136	172.16.20.1	TCP	66	21 → 35380 [ACK] Seq=136 Ack=48 Win=65280 Len=0 TSval=91633976 TSecr=50451191
36	14.822701366	192.168.109.136	172.16.20.1	FTP	132	Response: 150 Opening BINARY mode data connection for pub.txt (672 bytes).
37	14.826128331	172.16.20.1	192.168.109.136	TCP	66	35878 → 44750 [FIN, ACK] Seq=1 Ack=674 Win=30592 Len=0 TSval=50451196 TSecr=91633976
38	14.826136992	172.16.20.1	192.168.109.136	TCP	66	35380 → 21 [FIN, ACK] Seq=48 Ack=202 Win=29312 Len=0 TSval=50451196 TSecr=91633976
39	14.826976639	192.168.109.136	172.16.20.1	TCP	66	44750 → 35878 [ACK] Seq=674 Ack=2 Win=65280 Len=0 TSval=91633980 TSecr=50451196
40	14.827120724	192.168.109.136	172.16.20.1	FTP	90	Response: 226 Transfer complete.

Figura 27: Excerto de logs captados durante o download do ficheiro 'pub.txt' na experiência 6

B Passos

B.1 Experiência 1

Passos 2 e 3

Ligar Cabos

```
TUX23E0  -> Switch
TUX24E0  -> Switch

tux3:
> ifconfig eth0 up
> ifconfig eth0 172.16.20.1/24
> ifconfig eth0

tux4:
> ifconfig eth0 up
> ifconfig eth0 172.16.20.254/24
> ifconfig eth0
```

IP	MAC	tux/ether
172.16.20.1	00:21:5a:5a:7d:12	tux23 eth0
172.16.20.254	00:08:54:50:3f:2c	tux24 eth0

Tabela 1: Endreços IP, Endereços MAC e interfaces correspondentes

Passo 4

```
tux3:
> ping 172.16.20.254

tux4:
> ping 172.16.20.1
```

Passo 5

```
tux3:
> route -n
> arp -a
```

Passo 6

```
tux3:
> arp -d 172.16.20.254
> arp -a [retorna nada]
```

B.2 Experiência 2

Passo 1

Começar por fazer as ligações dos cabo

```
TUX23E0  -> Switch Porta 1 (Verificar que luz acende no Switch
    para saber porta)
TUX24E0  -> Switch Porta 2
TUX22E0  -> Switch Porta 3
```

Vista frontal das ligações no switch, lado esquerdo

TUX23E0 (1)	TUX22E0 (3)	empty (5)
TUX24E0 (2)	empty (4)	empty (6)

Tabela 2: Vista frontal das ligações no switch, lado esquerdo.

```
tux23:
> ifconfig eth0 up
> ifconfig eth0 172.16.20.1/24
> ifconfig eth0

tux24:
> ifconfig eth0 up
> ifconfig eth0 172.16.20.254/24
> ifconfig eth0

tux22:
> ifconfig eth0 up
> ifconfig eth0 172.16.21.1/24
> ifconfig eth0
```

IP	MAC	tux/ether
172.16.21.1	00:21:5a:61:2b:72	tux22 eth0
172.16.20.1	00:21:5a:5a:7d:12	tux23 eth0
172.16.20.254	00:08:54:50:3f:2c	tux24 eth0

Tabela 3: Endreços IP, Endereços MAC e interfaces correspondentes.

Passo 2

Ligar Cabo

```
TUX23S0  -> T3
T4 -> Switch Console
```

Para criar as vlans é necessário modificar o switch, para isso escolhe-se um tux e liga-se a porta série desse tux á porta série da consola do switch. Abre-se o terminal GtKterm, prime-se Enter para verificar que há ligação, depois pode-se seguir os passos abaixo.

Estes passos só precisam de ser realizados uma vez, não é necessário fazer em todos os tux's e não é preciso alterar cabos.

Dar login no switch:

```
>enable
>password: ******
```

Criar VLAN (vlan20):

```
>configure terminal
>vlan 20
>end
>show vlan id 20
```

Adicionar ports: Os valores das ports são os números que acendem ao ligar os cabos ao switch

Liga-se as ports 1 e 2 á VLAN20

Adicionar porta 1 à vlan 20:

```
>configure terminal
>interface fastethernet 0/1
>switchport mode access
>switchport access vlan 20
>end
>show running-config interface fastethernet 0/1
>show interfaces fastethernet 0/1 switchport
```

Adicionar porta 2 à vlan 20:

```
>configure terminal
>interface fastethernet 0/2
>switchport mode access
>switchport access vlan 20
>end
```

Passo 3

Criar outra VLAN (vlan21):

```
>configure terminal
>vlan 21
>end
>show vlan id 21
```

Adicionar a porta do tux22, porta 3:

```
>configure terminal
>interface fastethernet 0/3
>switchport mode access
>switchport access vlan 21
>end
```

Passo 4

Antes de fazer a captura no wireshark, verifica-se que os pings que se vão fazer abaixo funcionam. assim que estiver tudo bem, prepara-se os comandos nas consolas e pode-se começar a captura no WireShark.

Passo 5

No tux23, pingar o tux4:

```
ping 172.16.20.254
```

Pingar o tux22:

```
ping 172.16.21.1
```

Passo 6

Parar a captura e guardar o ficheiro.

Passo 7

Antes de fazer estas capturas prepara-se o ping de baixo, depois segue-se estes passos:

1. Em tux23 começar captura em eth0
2. Trocar para tux24
3. Em tux24 começar captura em eth0
4. Trocar para tux22
5. Em tux22 começar captura em eth0

Passo 8

No tux23, pingar em broadcast:

```
ping -b 172.16.20.255
```

Passo 9

1. Parar a captura em tux23
2. Parar a captura em tux24
3. Parar a captura em tux22
4. Guardar o ficheiro de tux23
5. Guardar o ficheiro de tux24
6. Guardar o ficheiro de tux22

Passo 10

Antes de fazer estas capturas prepara-se o ping de baixo, depois segue-se estes passos:

1. Em tux23 começar captura em eth0
2. Trocar para tux24
3. Em tux24 começar captura em eth0
4. Trocar para tux22
5. Em tux22 começar captura em eth0

No tux22, pingar em broadcast:

```
ping -b 172.16.21.255
```

1. Parar a captura em tux23
2. Parar a captura em tux24
3. Parar a captura em tux22
4. Guardar o ficheiro de tux23
5. Guardar o ficheiro de tux24
6. Guardar o ficheiro de tux22

B.3 Experiência 3

Passo 1

Começar por fazer as ligações dos cabos:

```
TUX23E0  -> Switch Porta 1
TUX22E0  -> Switch Porta 2
TUX24E0  -> Switch Porta 3
TUX24E1  -> Switch Porta 4
```

TUX23E0 (1)	TUX24E0 (3)	empty (5)
TUX22E0 (2)	TUX24E1 (4)	empty (6)

Tabela 4: Vista frontal das ligações no switch, lado esquerdo.

Configurar IP's:
tux23:

```
> ifconfig eth0 up
> ifconfig eth0 172.16.20.1/24
> ifconfig eth0
```

tux24:

```
> ifconfig eth0 up
> ifconfig eth0 172.16.20.254/24
> ifconfig eth0

> ifconfig eth1 up
> ifconfig eth1 172.16.21.253/24
> ifconfig eth1
```

tux22:

```
> ifconfig eth0 up
> ifconfig eth0 172.16.21.1/24
> ifconfig eth0
```

IP	MAC	tux/ether
172.16.21.1	00:21:5a:61:2b:72	tux22 eth0
172.16.20.1	00:21:5a:5a:7d:12	tux23 eth0
172.16.20.254	00:08:54:50:3f:2c	tux24 eth0
172.16.21.253	00:22:64:a6:a4:f1	tux24 eth1

Tabela 5: Endreços IP, Endereços MAC e interfaces correspondentes.

Configurar VLAN's:

```
TUX23S0  -> T3
T4 -> Switch Console
```

Liga-se um Cabo 'S0', TUX23S0 por exemplo, á porta 23 da prateleira de cima ('T3') e um cabo da porta 'T4' a 'switch console'.

Esta secção é para ser feita apenas uma vez, num tux á escolha, a partir do terminal GtkTerm, sem necessidade de alterar cabos.

VLAN 0:

```
- tux23 eth0 -> port 1
- tux24 eth0 -> port 3
```

VLAN 1:

```
- tux22 eth0 -> port 2
- tux24 eth1 -> port 4
```

Intruções Detalhadas:

Dar login no switch:

```
>enable
>password: ******
```

Criar VLAN (vlan20):

```
>configure terminal
>vlan 20
>end
>show vlan id 20
```

Adicionar porta 1 a vlan 20:

```
>configure terminal
>interface fastethernet 0/1
>switchport mode access
>switchport access vlan 20
>end
```

Adicionar porta 3 a vlan 20:

```
>configure terminal
>interface fastethernet 0/3
>switchport mode access
>switchport access vlan 20
>end
```

Criar VLAN (vlan21):

```
>configure terminal
>vlan 21
>end
>show vlan id 21
```

Adicionar porta 2 a vlan 21:

```
>configure terminal
>interface fastethernet 0/2
>switchport mode access
>switchport access vlan 21
>end
```

Adicionar porta 4 a vlan 21:

```
>configure terminal
>interface fastethernet 0/4
>switchport mode access
>switchport access vlan 21
>end
```

No final verificar se está tudo correto com:

```
>show vlan
```

Também se pode verificar com:

```
>show running-config interface fastethernet 0/1
>show interfaces fastethernet 0/1 switchport
```

testando com números de portas diferentes.

Enable IP forwarding and Disable ICMP echo-ignore-broadcast:
Troca-se para o tux24 e faz-se os seguintes comando no terminal:

```
echo 1 > /proc/sys/net/ipv4/ip_forward
echo 0 > /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts
```

Passo 2

Valores de IP e MAC da tabela acima

Passo 3

No tux23 adiciona-se uma rota para dizer que para aceder á vlan com endereços 172.16.21.0/24 usa-se como **gateway** o IP 172.16.20.254. Este comando é para ser feito na consola:

```
route add -net 172.16.21.0/24 gw 172.16.20.254
```

Da mesma forma, no tux22 faz-se:

```
route add -net 172.16.20.0/24 gw 172.16.21.253
```

Testa-se pingar o tux22 a partir do tux23 e o oposto para ver se existe conectividade.

```
Em tux23:
ping 172.16.21.1

Em tux22:
ping 172.16.20.1
```

Passo 4

Fazer 'route -n' em cada 1 dos 3 tuxs para observar as routes.
tux22:

Destination	Gateway	Genmask	Flags	Metric	Ref	Use	Iface
172.16.20.0	172.16.21.253	255.255.255.0	UG	0	0	0	eth0
172.16.21.0	0.0.0.0	255.255.255.0	U	0	0	0	eth0

Tabela 6: Tabela de encaminhamento para o tux22.

tux23:

Destination	Gateway	Genmask	Flags	Metric	Ref	Use	Iface
172.16.20.0	0.0.0.0	255.255.255.0	U	0	0	0	eth0
172.16.21.0	172.16.21.254	255.255.255.0	UG	0	0	0	eth0

Tabela 7: Tabela de encaminhamento para o tux23.

Destination	Gateway	Genmask	Flags	Metric	Ref	Use	Iface
172.16.20.0	0.0.0.0	255.255.255.0	U	0	0	0	eth0
172.16.21.0	0.0.0.0	255.255.255.0	U	0	0	0	eth1

Tabela 8: Tabela de encaminhamento para o tux24.

tux24:

Passo 5

Trocar para o tux23. Ligar o WireShark e começar a capturar pacotes na interface eth0

Passo 6

A partir do tux23:

1. pingar a eth0 do tux24 - ping 172.16.20.254
2. pingar a eth1 do tux24 - ping 172.16.21.253
3. pingar a eth0 do tux22 - ping 172.16.21.1

Para cada um verificar a conectividade

Passo 7

Parar a captura no tux23 e guardar logs.

Passo 8

1. Trocar para o tux24.
2. Ligar duas instâncias de WireShark. Uma para o eth0 e outra para o eth1.
3. Começar a capturar na eth0 e começar a capturar na eth1.

Passo 9

No tux24, apagar a tabela ARP e verificar que está vazia

```
> arp -a (verificar quais os IPs que se podem apagar)
> arp -d 172.16.20.254 (um dos IPs, tem de ser feito com todos
   os IPs de cima)
> arp -a (tem de retornar nada)
```

Fazer o mesmo no tux22 e tux23

Passo 10

No tux23 começar a pingar o tux22 (ping 172.16.21.1).

Passo 11

Trocar para tux24, parar as capturas e guardar os ficheiros.

B.4 Experiência 4

Passo 1

Ligar Cabos:

Mesma configuração da experiência 3, com a adição do router a ligar ao switch e a ligar á lab network.

```
TUX23E0     -> Switch Porta 1
TUX22E0     -> Switch Porta 2
TUX24E0     -> Switch Porta 3
TUX24E1     -> Switch Porta 4
ROUTERGE0   -> Switch Porta 5       (router-switch)
ROUTERGE1   -> Prateleira Porta 1   (router-lab network)
```

Ligações	no	switch
TUX23E0 (1)	TUX24E0 (3)	GE0 (5)
TUX22E0 (2)	TUX24E1 (4)	empty (6)

Tabela 9: Vista frontal das ligações no switch, lado esquerdo.

Configurar IP's: Os IP's do router têm de ser feitos a partir do GtkTerm.

> Configure commercial router RC and connect it (no NAT) to the lab network (172.16.1.0/24)

```
>interface gigabitethernet 0/0
>ip address 172.16.21.254    255.255.255.0
>no shutdown
>exit
>show interface gigabitethernet 0/0

>interface gigabitethernet 0/1
>ip address 172.16.1.29    255.255.255.0
>no shutdown
>exit
>show interface gigabitethernet 0/1
```

Estas routes são necessárias para o passo 2, mas podem ser adicionadas aqui.

```
>ip route 0.0.0.0 0.0.0.0 172.16.1.254
>ip route 172.16.20.0 255.255.255.0 172.16.21.253
```

Após ter o router configurado é necessário configurar tudo para trás como na experiência 3.

Configurar IP's:

tux23:

```
> ifconfig eth0 up
> ifconfig eth0 172.16.20.1/24
> ifconfig eth0
```

tux24:

```
> ifconfig eth0 up
> ifconfig eth0 172.16.20.254/24
> ifconfig eth0

> ifconfig eth1 up
> ifconfig eth1 172.16.21.253/24
> ifconfig eth1
```

tux22:

```
> ifconfig eth0 up
> ifconfig eth0 172.16.21.1/24
> ifconfig eth0
```

IP	MAC	tux/ether
172.16.20.1	00:21:5a:5a:7d:12	tux23 eth0
172.16.20.254	00:08:54:50:3f:2c	tux24 eth0
172.16.21.253	00:22:64:a6:a4:f1	tux24 eth1
172.16.21.1	00:21:5a:61:2b:72	tux22 eth0
172.16.21.254	68:ef:bd:e3:d7:78	RC eth0

Tabela 10: Endreços IP, Endereços MAC e interfaces correspondentes.

Configurar VLAN's

```
TUX23S0  -> T3
T4 -> Switch Console
```

Liga-se um Cabo 'S0', pode ser o TUX23S0, á porta 23 da prateleira de cima ('T3') e um cabo da porta 'T4' a 'switch console'.

Esta secção é para ser feita apenas uma vez, num tux á escolha, a partir do terminal GtkTerm, sem necessidade de alterar cabos.

VLAN 0:

- tux23 eth0 -> port 1
- tux24 eth0 -> port 3

VLAN 1:

- tux22 eth0 -> port 2
- tux24 eth1 -> port 4
- router eth0 -> port 5

Instruções Detalhadas:
Dar login no switch:

```
>enable
>password: ******
```

Criar VLAN (vlan20):

```
>configure terminal
>vlan 20
>end
>show vlan id 20
```

Adicionar porta 1 à vlan 20:

```
>configure terminal
>interface fastethernet 0/1
>switchport mode access
>switchport access vlan 20
>end
```

Adicionar porta 3 à vlan 20:

```
>configure terminal
>interface fastethernet 0/3
>switchport mode access
>switchport access vlan 20
>end
```

Criar VLAN (vlan21):

```
>configure terminal
>vlan 21
>end
>show vlan id 21
```

Adicionar porta 2 à vlan 21:

```
>configure terminal
>interface fastethernet 0/2
>switchport mode access
>switchport access vlan 21
>end
```

Adicionar porta 4 à vlan 21:

```
>configure terminal
>interface fastethernet 0/4
>switchport mode access
>switchport access vlan 21
>end
```

Adicionar porta 5 à vlan 21:

```
>configure terminal
>interface fastethernet 0/5
>switchport mode access
>switchport access vlan 21
>end
```

No final verificar se está tudo correto com:

```
>show vlan
```

Também se pode verificar com:

```
>show running-config interface fastethernet 0/1
>show interfaces fastethernet 0/1 switchport
```

ao testar com números de portas diferentes

Enable IP forwarding and Disable ICMP echo-ignore-broadcast

Troca-se para o tux24 e faz-se os seguintes comando no terminal

```
echo 1 > /proc/sys/net/ipv4/ip_forward
echo 0 > /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts
```

Adicionar Routes

No tux23 adiciona-se uma rota para dizer que para aceder á vlan com endereços 172.16.21.0/24 usa-se como gateway o IP 172.16.20.254. Este comando é para ser feito na consola:

```
route add -net 172.16.21.0/24 gw 172.16.20.254
```

Da mesma forma, no tux22 faz-se:

```
route add -net 172.16.20.0/24 gw 172.16.21.253
```

Testa-se pingar o tux22 a partir do tux23 e o oposto para ver se existe conectividade.

```
Em tux23:
ping 172.16.21.1

Em tux22:
ping 172.16.20.1
```

Se as rotas de cima estiverem corretas, é necessário adicionar mais umas routes para o passo 2:

tux22:

```
# do tux2 aceder a internet a partir do router
route add -net 172.16.1.0/24 gw 172.16.21.254
# definir RC como default router
route add default gw 172.16.21.254
```

tux23:

```
# definir tux24 como default router
route add default gw 172.16.20.254
```

tux24:

```
# do tux4 aceder a internet a partir do router
route add -net 172.16.1.0/24 gw 172.16.21.254
# definir RC como default router
route add default gw 172.16.21.254
```

Passo 2

Usar nos tuxs para verificar as tabelas resultantes:

```
> route -n
```

As tabelas devem ficar na seguinte forma:

Destination	Gateway	Genmask	Flags	Metric	Ref	Use	Iface
172.16.20.0	172.16.21.253	255.255.255.0	UG	0	0	0	eth0
172.16.21.0	0.0.0.0	255.255.255.0	U	0	0	0	eth0
172.16.1.0	172.16.21.254	255.255.255.0	UG	0	0	0	eth0

Tabela 11: Tabela de encaminhamento para o tux22.

Destination	Gateway	Genmask	Flags	Metric	Ref	Use	Iface
172.16.20.0	0.0.0.0	255.255.255.0	U	0	0	0	eth0
172.16.21.0	172.16.21.254	255.255.255.0	UG	0	0	0	eth0

Tabela 12: Tabela de encaminhamento para o tux23.

Destination	Gateway	Genmask	Flags	Metric	Ref	Use	Iface
172.16.20.0	0.0.0.0	255.255.255.0	U	0	0	0	eth0
172.16.21.0	0.0.0.0	255.255.255.0	U	0	0	0	eth1
172.16.1.0	172.16.21.254	255.255.255.0	UG	0	0	0	eth1

Tabela 13: Tabela de encaminhamento para o tux24.

Router:

Destination	Gateway	Genmask	Flags	Metric	Ref	Use	Iface
172.16.20.0	172.16.21.253	255.255.255.0	U	0	0	0	eth0
0.0.0.0	172.16.1.254	0.0.0.0	U	0	0	0	eth1

Tabela 14: Tabela de encaminhamento para o router.

Passo 3

No tux23, começar o wireshark e a capturar pacotes, pingar tux22, eth0 de tux24, eth1 de tux24 e Rc:

```
ping 172.16.20.254  #eth0 tux24
ping 172.16.21.253  #eth1 tux24
ping 172.16.21.1    #eth0 tux22
ping 172.16.21.254  #eth0 Router
ping 172.16.1.29    #eth1 Router
```

No final guarda-se o ficheiro.

Passo 4

Trocar para tux22 e seguir os próximos pontos:

> - Do: echo 0 > /proc/sys/net/ipv4/conf/eth0/acceptredirects and echo 0 > /proc/sys/net/ipv4/conf/all/acceptredirects

No terminal do tux22:

```
echo 0 > /proc/sys/net/ipv4/conf/eth0/accept_redirects
echo 0 > /proc/sys/net/ipv4/conf/all/accept_redirects
```

> - remove the route to 172.16.y0.0/24 via tuxy4

No tux22 usar comandos de baixo para apagar a entrada que usa como gateway o IP 172.16.21.253

```
# Para ver tabela
> route -n
# deve resultar em tabela de baixo
```

Destination	Gateway	Genmask	Flags	Metric	Ref	Use	Iface
172.16.20.0	172.16.21.253	255.255.255.0	UG	0	0	0	eth0
172.16.21.0	0.0.0.0	255.255.255.0	U	0	0	0	eth0
172.16.1.0	172.16.21.254	255.255.255.0	UG	0	0	0	eth0

Tabela 15: Tabela de encaminhamento atualizada para o tux22.

Aqui é preciso apagar a primeira entrada:

```
# Para apagar entrada
> route del -net 172.16.20.0/24 gw 172.16.21.253
```

> - In tuxy2, ping tuxy3

No tux22 fazer um ping:

```
ping 172.16.20.1    #eth0 tux23
```

> - Using capture at tuxy2, try to understand the path followed by ICMP ECHO and ECHO-REPLY packets (look at MAC addresses)

No tux22, liga-se o wireshark e começa-se a capturar no eth0:

```
ping 172.16.20.1    #eth0 tux23
```

deixar uns segundos e guardar os logs.

> - In tuxy2, do traceroute tuxy3

```
traceroute 172.16.20.1    #eth0 tux23
```

> - In tuxy2, add again the route to 172.16.y0.0/24 via tuxy4 and do traceroute tuxy3

```
route add -net 172.16.20.0/24 gw 172.16.21.253
# seguido de
traceroute 172.16.20.1    #eth0 tux23
```

> - Activate the acceptance of ICMP redirect at tuxy2 when there is no route to 172.16.y0.0/24 via tuxy4 and try to understand what happens

```
# Apagar a entrada 172.16.20.0/24 outra vez no tux22
route -n
route del -net 172.16.20.0/24 gw 172.16.21.253

# ativar isto de novo
echo 1 > /proc/sys/net/ipv4/conf/eth0/accept_redirects
echo 1 > /proc/sys/net/ipv4/conf/all/accept_redirects
```

Começa-se uma captura e faz-se os mesmos comandos que se fez na outra captura:

```
ping 172.16.20.1            #eth0
traceroute 172.16.20.1      #eth0 tux23
traceroute -I 172.16.20.1   #eth0 tux23
```

Termina-se a captura e guarda-se os logs.

Passo 5

Trocar para tux23 e fazer:

```
ping 172.16.1.254
```

É suposto não funcionar porque é necessário implementar NAT.

Passo 6

```
# Defines Ethernet 0 with an IP address and as a NAT inside
    interface.
conf t
interface gigabitethernet 0/0 *
ip address 172.16.21.254 255.255.255.0
no shutdown
ip nat inside
exit

# Defines Ethernet 1 with an IP address and as a NAT outside
    interface.
# 172.16.2.29 para i320
interface gigabitethernet 0/1 *
ip address 172.16.1.29 255.255.255.0
no shutdown
ip nat outside
exit

# Defines a NAT pool named ovrld with a range of a single IP
    address, 172.16.1.29.
ip nat pool ovrld 172.16.1.29 172.16.1.29 prefix 24
# Indicates that any packets received on the inside interface
    that are permitted by access-list 1 has the source address
    translated to an address out of the NAT pool named ovrld.
    Translations are overloaded, which allows multiple inside
    devices to be translated to the same valid IP address.
ip nat inside source list 1 pool ovrld overload

# Access-list 1 permits packets with source addresses ranging
    from 172.16.20.0 through 172.16.20.7 and 172.16.21.0
    through 172.16.21.7.
access-list 1 permit 172.16.20.0 0.0.0.7
access-list 1 permit 172.16.21.0 0.0.0.7

ip route 0.0.0.0 0.0.0.0 172.16.1.254
ip route 172.16.20.0 255.255.255.0 172.16.21.253
end

# In room I320 use interface fastethernet
```

Passo 7

Trocar para tux23 e fazer:

```
ping 172.16.1.254
```

B.5 Experiência 5

Passo 0

Primeiro é necessário configurar tudo como na experiência 4 para ser possivel testar esta experiência

Passo 1 Começando no tux22:

```
echo $'search netlab.fe.up.pt\nnameserver 172.16.1.1' > /etc/
    resolv.conf
```

e fazer o mesmo no tux3 e tux4.

Passo 2

Verificar se a configuração ficou bem feita usando ping com nomes em vez de IP's

```
ping ftp.up.pt
```

Passo 3

Ligar wireshark e fazer:

```
ping ftp.up.pt
ping google.com
```

B.6 Experiência 6

Passo 1

Compilar a aplicação de download e configurar a rede até à experiência anterior.

Passo 2

Começar a capturar pacotes no tux23 e fazer

```
download ftp://netlab1.fe.up.pt/pub.txt;
```

Passo 3

Parar a captura e guardar os logs.

Passo 4

Analisar os Logs.

Passo 5

No tux23 o download terá de descarregar um ficheiro maior, podemos fazer:

```
download ftp://rcom:rcom@netlab1.fe.up.pt/files/crab.mp4;
```

já que este ficheiro tem 86.0 MB.

Inicia-se então a captura no WireShark, inicia-se o download no tux23, troca-se para o tux22 e faz-se

```
download ftp://rcom:rcom@netlab1.fe.up.pt/pipe.txt
```

Não esquecer de compilar o download no tux22 antes de fazer este passo. No final guardar os logs.

C Código Fonte - Download

```h
#ifndef DOWNLOAD_HEADER
#define DOWNLOAD_HEADER

#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

#include "args.h"
#include "connection.h"

#endif // DOWNLOAD_HEADER
```

Listing 1: download.h

```c
#include "download.h"

/**
 * @brief Main function called to execute the download
 *
 * @param argc number of arguments
 * @param argv value of the arguments
 * @return int return value
 */
int main(int argc, char** argv){
  if (argc != 2) {
    fprintf(stderr, "usage: download ftp://[<user>:<password>@]<host>/<url-path>\n");
    exit(1);
  }

  args arguments;
  int socketfd, socketfd_rec;
  char urlcpy[256];
  char command[256];

  strcpy(urlcpy, argv[1]);

  if (parseArgs(urlcpy, &arguments) != 0){
    printf("usage: %s ftp://[<user>:<password>@]<host>/<url-path>\n",argv[0]);
    return -1;
  }

  printf("\nhost: %s\npath: %s\nuser: %s\npassword: %s\nfile name: %s\nhost name: %s\nip address: %s\n\n",
  arguments.host, arguments.path, arguments.user, arguments.password, arguments.file_name, arguments.host_name, arguments.ip);

  if (init(arguments.ip, 21, &socketfd) != 0){
    printf("Error: init()\n");
    return -1;
```

```
  }

  socketFile = fdopen(socketfd, "r");
  readResponse();

  // login
  sprintf(command, "user %s\r\n", arguments.user);
  sendCommand(socketfd, command);
  if (readResponse() != 0) return 1;
  sprintf(command, "pass %s\r\n", arguments.password);
  sendCommand(socketfd, command);
  if (readResponse() != 0) return 1;

  // get ip and port
  char ip[32]; int port;
  sprintf(command, "pasv\r\n");
  sendCommand(socketfd, command);
  readResponsePassive(&ip, &port);
  printf("ip: %s\nport: %d\n", ip, port);

  if (init(ip, port, &socketfd_rec) != 0){
    printf("Error: init()\n");
    return -1;
  }

  sprintf(command, "retr %s\r\n", arguments.path);
  sendCommand(socketfd, command);
  if (readResponse() != 0) return 1;

  saveFile(arguments.file_name, socketfd_rec);
  return 0;
}
```

Listing 2: download.c

```
#ifndef CONNECT_HEADER
#define CONNECT_HEADER

#include <sys/socket.h>
#include <netinet/in.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>


#include "args.h"
FILE * socketFile;

/**
 * @brief Opens a socket and connects to the server
 *
 * @param ip Server IP Adress
```

```
 * @param port Server Port
 * @param socketfd File Descriptor returned by socket()
 * @return int Returns 0 if there are no errors or a positive number otherwise
 */
int init(char *ip, int port, int *socketfd);

/**
 * @brief Sends a message to the server via socket
 *
 * @param socketfd File Descriptor of the socket
 * @param command String with the command to send
 * @return int Returns 0 if there are no errors or a positive number otherwise
 */
int sendCommand(int socketfd, char * command);

/**
 * @brief Prints to the standart output the reply received after sending a command to empty the reply buffer
 *
 * @return int Returns 0
 */
int readResponse();

/**
 * @brief Reads the reply to the "pasv" command ans parses the IP Adress and Port Values
 *
 * @param ip IP Adress
 * @param port Server Port
 * @return int Returns 0
 */
int readResponsePassive(char (*ip)[], int *port);

/**
 * @brief While reading the reply from the served, dumps the information received into a file
 *
 * @param filename Name of the File to be created
 * @param socketfd Socket File Descriptor from where to read the content
 * @return int Returns 0 if there are no errors or a positive number otherwise
 */
int saveFile(char* filename, int socketfd);
#endif // CONNECT_HEADER
```

Listing 3: connection.h

```
#include "connection.h"

int init(char *ip, int port, int *socketfd){
  struct  sockaddr_in server_addr;
```

```
5
6   /*server address handling*/
7   bzero((char*)&server_addr,sizeof(server_addr));
8   server_addr.sin_family = AF_INET;
9   server_addr.sin_addr.s_addr = inet_addr(ip);  /*32 bit
      Internet address network byte ordered*/
10  server_addr.sin_port = htons(port);   /*server TCP port must
      be network byte ordered */
11
12  /*opens a TCP socket*/
13  if ((*socketfd = socket(AF_INET,SOCK_STREAM,0)) < 0) {
14    perror("socket()");
15    return 1;
16  }
17
18  /*connects to the server*/
19  if(connect(*socketfd, (struct sockaddr *)&server_addr, sizeof
      (server_addr)) < 0){
20    perror("connect()");
21    return 1;
22  }
23
24  return 0;
25 }
26
27
28 int sendCommand(int socketfd, char * command){
29   printf(" about to send command: \n> %s", command);
30   int sent = send(socketfd, command, strlen(command), 0);
31   if (sent == 0){
32     printf("sendCommand: Connection closed");
33     return 1;
34   }
35   if (sent == -1){
36     printf("sendCommand: error");
37     return 2;
38   }
39   printf("> command sent\n");
40   return 0;
41 }
42
43 int readResponse(){
44   char * buf;
45   size_t bytesRead = 0;
46
47   while (1){
48     getline(&buf, &bytesRead, socketFile);
49     printf("< %s", buf);
50     if (buf[3] == ' '){
51       long code = strtol(buf, &buf, 10);
52       if (code == 550 || code == 530)
53       {
54         printf("Command error\n");
55         return 1;
```

```
56        }
57        break;
58      }
59    }
60    return 0;
61  }
62
63  int readResponsePassive(char (*ip)[], int *port){
64    char * buf;
65    size_t bytesRead = 0;
66
67    while (1){
68      getline(&buf, &bytesRead, socketFile);
69      printf("< %s", buf);
70      if (buf[3] == ' '){
71        break;
72      }
73    }
74
75    strtok(buf, "(");
76    char* ip1 = strtok(NULL, ",");        // 193
77    char* ip2 = strtok(NULL, ",");        // 137
78    char* ip3 = strtok(NULL, ",");        // 29
79    char* ip4 = strtok(NULL, ",");        // 15
80
81    sprintf(*ip, "%s.%s.%s.%s", ip1, ip2, ip3, ip4);
82
83    char* p1 = strtok(NULL, ",");        // 199
84    char* p2 = strtok(NULL, ")");        // 78
85
86    *port = atoi(p1)*256 + atoi(p2);
87
88    return 0;
89  }
90
91
92  int saveFile(char* filename, int socketfd){
93    printf("> filename: %s\n", filename);
94    int filefd = open(filename, O_WRONLY | O_CREAT, 0777);
95    if (filefd < 0){
96      printf("Error: saveFile\n");
97      return 1;
98    }
99
100   int bytes_read;
101   char buf[1];
102   do {
103     bytes_read = read(socketfd, buf, 1);
104     if (bytes_read > 0) write(filefd, buf, bytes_read);
105   } while (bytes_read != 0);
106
107   close(filefd);
108   return 0;
```

```
109 }
```

Listing 4: connection.c

```
1  #ifndef ARGS_HEADER
2  #define ARGS_HEADER
3
4  #define h_addr h_addr_list[0] //The first address in
       h_addr_list.
5
6  #include <stdio.h>
7  #include <string.h>
8  #include <netdb.h>
9  #include <sys/socket.h>
10 #include <netinet/in.h>
11 #include <arpa/inet.h>
12 /**
13  * @brief Struct that keeps the data from the URL argument
14  */
15 typedef struct args
16 {
17   char user[128];        ///< User used for Login
18   char password[128];    ///< Password used for Login
19   char host[256];        ///< Host name in URL
20   char path[240];        ///< Path to the file
21   char file_name[128];   ///< Name of the File
22   char host_name[128];   ///< Host Name from gethostbyname()
23   char ip[128];          ///< IP Adress from gethostbyname()
24
25 } args;
26
27 /**
28  * @brief Parses the URL received via arguments to the struct
       args
29  *
30  * @param url string received through the arguments of download
31  * @param args struct args where the information will be saved
32  * @return int Returns 0 if there are no errors or a positive
       number otherwise
33  */
34 int parseArgs(char *url, args * args);
35
36 /**
37  * @brief Given a host name returns the ip address using
       gethostbyname()
38  *
39  * @param host String with the host's name
40  * @param args struct args to put the IP Adress in
41  * @return int Returns 0 if there are no errors or 1 otherwise
42  */
43 int getIp(char *host, args * args);
44
45 /**
46  * @brief Gets the last element of the path string,
```

```
    corresponding to the file name
 *
 * @param args arguments struct to get path and fill filename
 * @return int Returns 0 if there are no errors or 1 otherwise
 */
int getFileName(args * args);
#endif // ARGS_HEADER
```

Listing 5: args.h

```
#include "args.h"

int parseArgs(char *url, args *args){
  char* ftp = strtok(url, "/");        // ftp:
  char* urlrest = strtok(NULL, "/");   // [<user>:<password>@]<
    host>
  char* path = strtok(NULL, "");       // <url-path>

  if (strcmp(ftp, "ftp:") != 0){
    printf("Error: Not using ftp\n");
    return 1;
  }

  char* user = strtok(urlrest, ":");
  char* pass = strtok(NULL, "@");


  // no user:password given
  if (pass == NULL)
  {
    user = "anonymous";
    pass = "pass";
    strcpy(args->host, urlrest);
  } else
    strcpy(args->host, strtok(NULL, ""));


  strcpy(args->path, path);
  strcpy(args->user, user);
  strcpy(args->password, pass);


  if (getIp(args->host, args) != 0){
    printf("Error: getIp()\n");
    return 2;
  }

  if (getFileName(args) != 0){
    printf("Error: getFileName()\n");
    return 3;
  }

  return 0;
}
```

```
44
45 int getIp(char *host, args *args){
46   struct hostent *h;
47
48   printf("Running gethostbyname - if this blocks -> CTRL+C or
       wait\n");
49   if ((h=gethostbyname(host)) == NULL) {
50     herror("gethostbyname");
51     return 1;
52   }
53   printf("Returning from gethostbyname\n");
54
55   strcpy(args->host_name,h->h_name);
56   strcpy(args->ip, inet_ntoa( *( (struct in_addr *)h->h_addr )
       ));
57   return 0;
58 }
59
60 int getFileName(args * args){
61   char fullpath[256];
62   strcpy(fullpath, args->path);
63   char* token = strtok(fullpath, "/");
64   while( token != NULL ) {
65     strcpy(args->file_name, token);
66     token = strtok(NULL, "/");
67   }
68   return 0;
69 }
```

Listing 6: args.c