Rede de Computadores

2º Trabalho Laboratorial

2023/2024

Redes de Computadores

Turma 12 – Grupo 9

- Francisco Dias Pires Ferreira de Sousa (up202108715@fe.up.pt)

- Simão Queirós Rodrigues (up202005700@fe.up.pt)

Porto, 22 de Dezembro de 2023

Sumário

No contexto da Unidade Curricular de Redes de Computadores 2023/2024, este projeto visa criar uma aplicação de download usando o FTP e estudar a configuração de uma rede de computadores.

Introdução

O objetivo deste projeto foi desenvolver e testar um aplicativo para fazer o download de conteúdo usando o protocolo FTP, além de configurar uma rede de computadores. O relatório está organizado em sete partes principais:

DownloadApp: Aplicação criada de download FTP com a sua respetiva arquitetura.

Configuração e Análise de redes:

- Experiência 1: Configurar uma rede IP;
- Experiência 2: Implementar duas bridges num switch;
- Experiência 3: Configurar um router em Linux;
- Experiência 4: Configurar um router comercial com NAT;
- Experiência 5: DNS;
- Experiência 6: Ligações TCP.

Conclusões: Síntese das informações apresentadas nas seções anteriores, acompanhada de uma reflexão sobre os objetivos de aprendizagem alcançados durante a realização do projeto.

DownloadApp

Arquitetura da aplicação

No âmbito deste relatório, apresenta-se a descrição detalhada da aplicação desenvolvida, cuja principal funcionalidade é o download de ficheiros utilizando o protocolo FTP. Este desenvolvimento foi embasado na análise e aplicação das normas RFC959, que detalha a operação e estrutura do protocolo FTP, e a norma RFC1738, que aborda a composição dos endereços URL.

O processo inicia com a análise do URL fornecido como parâmetro. Esta análise é realizada através de expressões regulares, visando extrair uma estrutura de dados contendo informações cruciais para a conexão: o 'host', referente ao servidor com o qual se estabelece a comunicação; o 'resource', que indica o caminho para o ficheiro desejado; o 'file', que especifica o nome e a extensão do ficheiro a ser transferido; os campos 'user' e 'password', utilizados para autenticação no servidor e que podem assumir valores padrão caso não sejam especificados; e o 'ip', derivado do host através do sistema DNS.

Para assegurar a integridade e correção dos dados, implementaram-se várias camadas de validação. Estas incluem a verificação da correta formulação do URL, a existência do servidor e do ficheiro solicitado, fortalecendo a robustez do tratamento dos dados iniciais.

No que toca à comunicação com o servidor, todas as funções empregadas recebem como argumentos um socket, um buffer para receção de mensagens, e devolvem um código numérico de três dígitos que representa a resposta do servidor. A receção destas mensagens é efetuada por uma máquina de estados, concretizada na função readResponse. Se, em qualquer fase, o código recebido não pertencer às gamas 2XX ou 3XX, como pode ocorrer durante as fases de login, a operação é interrompida e uma mensagem de erro é exibida.

O procedimento inicia-se com a criação de um socket, estabelecendo conexão com o IP do host. Após a fase de handshake, procede-se à autenticação no servidor usando as credenciais fornecidas. Posteriormente, solicita-se ao servidor que entre em modo passivo através do código “passv”, o que permite obter o IP e a porta para uma segunda conexão FTP, destinada à receção do ficheiro. Esta fase do download é realizada pelas funções requestResource e getResource.

Por fim, após a transferência bem-sucedida do ficheiro, as conexões e os sockets são encerrados, utilizando-se comandos “quit” na execução da função closeConnection, concluindo assim o processo de download.

Configuração e Análise de redes:

Experiência 1: Configurar uma rede IP

Esta experiência teve como finalidade a configuração de endereços IP em dois computadores, designados por TuxY3 e TuxY4, conectados através de um switch. O foco principal consistiu na exploração do comportamento do protocolo ARP, observando as alterações na comunicação ao remover entradas de IPs das tabelas ARP correspondentes.

Inicialmente, estabeleceu-se a ligação física, conectando a porta E0 de cada computador Tux ao switch. Utilizou-se o comando ifconfig para atribuir um IP específico a cada máquina. Os endereços MAC e IP de ambos os computadores foram posteriormente verificados nas tabelas ARP, que foram atualizadas após se testar a conectividade entre os dois dispositivos com o comando ping. O TuxY3 foi atribuído com o IP 172.16.Y0.1/24 e o TuxY4 com o IP 172.16.Y0.254/24. A análise dos pacotes trocados durante este teste revelou a presença de pacotes ARP e ICMP.

- O ARP (Address Resolution Protocol) desempenha um papel vital na configuração de ligações, associando o IP de um dispositivo ao seu endereço MAC e facilitando a interação entre a camada de rede e a camada de ligação. Durante a comunicação, um pacote ARP contendo os IPs do dispositivo de destino e de origem é enviado em broadcast. Em resposta, o dispositivo receptor envia de volta um pacote ARP com o seu MAC, sendo estas informações armazenadas nas tabelas ARP dos computadores. Observou-se que, ao eliminar entradas destas tabelas, os dispositivos iniciavam uma nova troca de pacotes ARP para reestabelecer as associações entre os IPs e os MACs anteriormente removidos.

- O ICMP (Internet Control Message Protocol) é utilizado para transmitir mensagens de controlo, indicando o sucesso ou falhas na comunicação com outro endereço IP. Durante a experiência, identificaram-se mensagens de reply e request transmitidas por este protocolo.

Experiência 2: Implementar duas bridges num switch

O propósito central desta experiência foi a configuração de duas Redes Locais (Local Area Networks - LANs), uma integrando os computadores TuxY3 e TuxY4, e a outra exclusivamente com o TuxY2. Utilizaram-se duas bridges num switch para alcançar este objetivo. Inicialmente, foi necessário replicar a configuração de rede previamente estabelecida e atribuir o endereço IP 172.16.Y1.1/24 ao TuxY2. A configuração das bridges foi efetuada no switch, estabelecendo uma ligação entre a sua consola e a porta série do TuxY2. Durante este processo, configuraram-se as bridges Y0 e Y1, utilizando o comando /interface bridge add. Posteriormente, removeu-se as portas padrão ligadas a cada Tux com o comando /interface bridge port remove, e adicionou-se as interfaces do TuxY3 e TuxY4 à bridge Y0, e a do TuxY2 à bridge Y1, através do comando /interface bridge port add.

Durante a experiência, identificaram-se dois distintos domínios de broadcast, cada um correspondendo a uma das bridges/LANs implementadas. Constatou-se que, ao realizar um broadcast a partir do TuxY3, o TuxY4 recebia o sinal, mas o TuxY2 não, por estar numa LAN

diferente. Esta observação foi confirmada ao realizar o mesmo procedimento a partir do TuxY2, verificando-se que este não conseguia comunicar com os outros computadores, estando isolado na sua própria LAN. Esta conclusão foi deduzida através da análise dos pacotes ICMP capturados nos dois computadores. Na primeira situação, registaram-se trocas de mensagens de reply e request, enquanto na segunda, apenas se observaram mensagens reply sem qualquer resposta.

Experiência 3: Configurar um router em Linux

A presente experiência iniciou-se com a continuação da configuração de rede estabelecida na experiência anterior, onde o TuxY3 e o TuxY4 estavam conectados a uma bridge e o TuxY2 a outra. O principal objetivo desta fase foi transformar o TuxY4 num router, com a finalidade de facilitar a comunicação entre o TuxY3 e o TuxY2, que se encontravam em redes locais (LANs) diferentes.

Para atingir este fim, procedeu-se com a ligação da porta E1 do TuxY4 ao switch, atribuindo-lhe o endereço IP 172.16.Y1.253/24. Em seguida, efetuou-se a remoção das portas padrão do switch associadas ao TuxY4, integrando esta interface à bridgeY1. Posteriormente, desativou-se a opção icmp_echo_ignore_broadcasts e ativou-se a funcionalidade de IP forwarding no TuxY4. Através do comando route add, configuraram-se rotas nos computadores TuxY2 e TuxY3, designando como gateway o IP do TuxY4, acessível a cada um deles nas suas respectivas LANs, isto é, 172.16.Y1.253/24 para o TuxY2 e 172.16.Y0.254/24 para o TuxY3.

A configuração das rotas permitiu que o TuxY4 assumisse eficientemente a função de um router, conectando as duas LANs distintas. Esta modificação possibilitou que, ao executar o comando ping entre o TuxY3 e o TuxY2, todos os pacotes ARP e ICMP fossem corretamente transmitidos e recebidos. Nos pacotes ARP e ICMP processados pelo TuxY4, identificou-se o endereço IP do dispositivo de destino, mas com o endereço MAC do TuxY4, evidenciando a sua função de router na transferência de dados entre as duas redes. As tabelas de encaminhamento, criadas com as rotas estabelecidas, garantem que, para cada IP de destino, haja um IP de gateway correspondente, assegurando o correto redirecionamento dos dados pela máquina de origem.

Experiência 4: Configurar um router comercial com NAT

Nesta etapa do projeto, partimos da configuração de rede já estabelecida na experiência anterior, compreendendo duas redes locais (LANs) distintas: uma com o TuxY3 e outra com o TuxY2, e o TuxY4 atuando como router entre ambas. O foco principal foi na configuração e integração de um router comercial com NAT (Network Address Translation) à bridgeY1, para habilitar o acesso à internet.

A fase inicial envolveu a conexão de uma entrada do router comercial à fonte de energia e outra ao switch, integrando esta interface à bridgeY1, seguindo o procedimento das experiências anteriores. Posteriormente, alterou-se a ligação do TuxY2, que antes se conectava à consola do switch, passando agora a conectar-se à consola do router comercial, para que se procedesse à sua configuração. Na consola do router, configuraram-se os IPs para cada interface com o comando /ip address add. Concluiu-se o processo criando rotas default em cada um dos Tuxes, direcionando-os para o router, e uma rota que permitia ao router conectar-se a cada uma das LANs através do TuxY4.

Identificaram-se dois cenários distintos durante a experiência. No primeiro, sem uma conexão direta entre o TuxY2 e o TuxY4 e na ausência de redirecionamentos ICMP, os pacotes do TuxY2 para o TuxY3 foram encaminhados através do router pela rota default até ao seu destino. No segundo cenário, com as rotas e redirecionamentos ICMP reativados, os pacotes não transitaram pelo router, optando pelo TuxY4 como intermediário direto entre as duas LANs. Conclui-se que os pacotes ICMP desempenham um papel crucial na otimização das conexões na rede, fornecendo redirecionamentos quando necessário.

O NAT é um mecanismo que transforma endereços de uma rede interna em um endereço público único e vice-versa. Quando um pacote é enviado para fora da rede local, utiliza-se o endereço público como remetente. As respostas recebidas no endereço público são posteriormente convertidas de volta para o endereço interno original. Isso não só reduz a necessidade de endereços públicos, como também ajuda a mitigar a escassez de endereços IP únicos, um problema comum no uso extensivo do IPv4. Observou-se que com o NAT ativo, a conexão à internet era viável, enquanto que desativado, o router não conseguia traduzir o endereço de destino recebido num endereço da rede interna, impedindo a receção de respostas.

Experiência 5: DNS

Nesta etapa do projeto, partiu-se da infraestrutura de rede já estabelecida nas experiências anteriores com o propósito de configurar o Sistema de Nomes de Domínio (DNS). O objetivo principal foi viabilizar o acesso a websites da internet a partir dos seus nomes de domínio dentro da rede formada. A funcionalidade desta configuração foi verificada utilizando o comando ping ao endereço do google.com.

Para a realização desta experiência, procedeu-se com a modificação do arquivo /etc/resolv.conf em cada um dos computadores Tux, inserindo a linha ‘nameserver 172.16.1.1’. Este endereço IP é referente ao router disponível no laboratório, que fornece acesso à rede externa.

Durante a experiência, observou-se que as interações com os pacotes DNS precedem quaisquer outros protocolos na rede. O DNS desempenha um papel crucial, convertendo os nomes de domínio em endereços IP correspondentes, os quais são subsequentemente utilizados por todos os outros protocolos de rede. Portanto, ao efetuar um ping para o google.com, o processo inicia-se com a resolução do nome de domínio para o seu respectivo endereço IP por meio do DNS, seguido pela comunicação normal de outros protocolos.

Experiência 6: Ligações TCP

Esta experiência, inserida no contexto das configurações de rede previamente estabelecidas, teve como objetivo a análise do envio e receção de pacotes, explorando a estrutura interna destes e os mecanismos de controlo de fluxo e congestionamento inerentes às conexões TCP. Utilizou-se, para tal, a aplicação de download desenvolvida nas fases anteriores.

No início da experiência, procedeu-se à compilação e utilização da aplicação de download no TuxY3 para descarregar um ficheiro de dimensão considerável. Observou-se inicialmente a troca de pacotes DNS para conversão do nome do servidor em endereço IP. Seguiu-se a interação FTP SYN-ACK request/response, essencial para o estabelecimento da ligação entre cliente e servidor, culminando com a transferência de pacotes FTP Data, caracterizados por

números de sequência progressivos. A operação finalizou-se com o envio do pacote FTP FIN-ACK.

A aplicação FTP estabelece duas conexões TCP distintas: uma para a transmissão de comandos de controlo ao servidor e outra para a receção do ficheiro. Estas conexões, baseadas no protocolo ARQ, são orientadas à comunicação e asseguram a transferência de dados, incorporando mecanismos de controlo de fluxo e congestionamento.

O mecanismo ARQ (Automatic Repeat Request) é empregado para o controlo de erros. Este utiliza mensagens ACK (acknowledge) para confirmar a receção adequada de pacotes e utiliza timeouts para monitorizar o tempo de receção. Em caso de ultrapassar o timeout, presume-se a perda do pacote, que é retransmitido.

O controlo de fluxo, por outro lado, permite ao receptor gerir o fluxo de receção de pacotes. Este processo envolve a comunicação da window size em bytes em cada pacote, possibilitando ao emissor avaliar a quantidade de dados que pode enviar sem sobrecarregar o receptor.

O controlo de congestionamento é realizado pelos emissores e recorre a métodos como Selective Repeat, Additive Increase e Slow Start. Estes ajustam o volume de pacotes enviados com base na detecção de perdas na rede.

Na segunda fase da experiência, iniciou-se um download no TuxY3 e, posteriormente, outro no TuxY2. Observou-se uma redução na velocidade de transferência do TuxY3 para aproximadamente metade do valor inicial com o início do segundo download, refletindo a ação do controlo de congestionamento e a rápida convergência para valores estáveis durante a transferência simultânea.

Os gráficos gerados a partir da velocidade de transferência de pacotes evidenciaram claramente a influência dos mecanismos de controlo na ligação TCP. Após fases de incremento acentuado no volume de pacotes (Additive Increase ou Slow Start), notou-se uma redução para cerca de metade desse volume, em resposta aos mecanismos de controlo implementados.

Conclusões

Ao concluir com êxito todas as metas definidas, este projeto proporcionou uma oportunidade valiosa para expandir e consolidar nosso conhecimento acerca dos protocolos envolvidos na transmissão de dados em redes de computadores. Este aprofundamento incluiu uma análise pormenorizada da forma como os dados são disseminados na camada de rede e do processo pelo qual estes transitam para a camada de ligação.

Anexos:

Anexo 1: Código

1.1 - download.h

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

#define MAX_LENGTH  500
#define FTP_PORT    21

/* Server responses */
#define SV_READY4AUTH           220
#define SV_READY4PASS           331
#define SV_LOGINSUCCESS         230
#define SV_PASSIVE              227
#define SV_READY4TRANSFER       150
#define SV_TRANSFER_COMPLETE    226
#define SV_GOODBYE              221

/* Parser regular expressions */
```

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

/* Default login for case 'ftp://<host>/<url-path>' */
#define DEFAULT_USER        "anonymous"
#define DEFAULT_PASSWORD    "password"

/* Parser output */
struct URL {
    char host[MAX_LENGTH];      // 'ftp.up.pt'
    char resource[MAX_LENGTH];  // 'parrot/misc/canary/warrant-canary-0.txt'
    char file[MAX_LENGTH];      // 'warrant-canary-0.txt'
    char user[MAX_LENGTH];      // 'username'
    char password[MAX_LENGTH];  // 'password'
    char ip[MAX_LENGTH];        // 193.137.29.15
};

/* Machine states that receives the response from the server */
typedef enum {
    START,
    SINGLE,
    MULTIPLE,
    END
} ResponseState;

int parse(char *input, struct URL *url);
int createSocket(char *ip, int port);
int authConn(const int socket, const char *user, const char *pass);
int readResponse(const int socket, char *buffer);
int passiveMode(const int socket, char* ip, int *port);
int requestResource(const int socket, char *resource);
int getResource(const int socketA, const int socketB, char *filename);
int closeConnection(const int socketA, const int socketB);
```

1.2 - download.c

```
2  #include "download.h"
3
4  int parse(char *input, struct URL *url) {
5      regex_t regex;
6      int isMatched;
7
8      // Check if input matches BAR regex
9      regcomp(&regex, BAR, 0);
10     isMatched = regexec(&regex, input, 0, NULL, 0);
11     regfree(&regex);
12     if (isMatched) return -1;
13
14     // Check if input matches AT regex
15     regcomp(&regex, AT, 0);
16     isMatched = regexec(&regex, input, 0, NULL, 0);
17     regfree(&regex);
18
19     if (isMatched) { //ftp://<host>/<url-path>
20         sscanf(input, HOST_REGEX, url->host);
21         strcpy(url->user, DEFAULT_USER);
22         strcpy(url->password, DEFAULT_PASSWORD);
23     } else { // ftp://[<user>:<password>@]<host>/<url-path>
24         sscanf(input, HOST_AT_REGEX, url->host);
25         sscanf(input, USER_REGEX, url->user);
26         sscanf(input, PASS_REGEX, url->password);
27     }
28
29     sscanf(input, RESOURCE_REGEX, url->resource);
30     strcpy(url->file, strrchr(input, '/') + 1);
31
32     struct hostent *h;
```

```
33      if (strlen(url->host) == 0) return -1;
34      if ((h = gethostbyname(url->host)) == NULL) {
35          printf("Invalid hostname '%s'\n", url->host);
36          exit(-1);
37      }
38      strcpy(url->ip, inet_ntoa(*((struct in_addr *) h->h_addr)));
39
40      return !(strlen(url->host) && strlen(url->user) &&
41             strlen(url->password) && strlen(url->resource) && strlen(url-
    >file));
42  }
43
44  int createSocket(char *ip, int port) {
45      int sockfd;
46      struct sockaddr_in server_addr;
47
48      // Initialize server_addr to zeros
49      memset(&server_addr, 0, sizeof(server_addr));
50
51      // Set the values for server_addr
52      server_addr.sin_family = AF_INET;
53      server_addr.sin_addr.s_addr = inet_addr(ip);
54      server_addr.sin_port = htons(port);
55
56      // Create the socket
57      sockfd = socket(AF_INET, SOCK_STREAM, 0);
58      if (sockfd < 0) {
59          perror("socket()");
60          exit(EXIT_FAILURE);
61      }
62
63      // Connect to the server
64      if (connect(sockfd, (struct sockaddr *) &server_addr,
    sizeof(server_addr)) < 0) {
65          perror("connect()");
66          exit(EXIT_FAILURE);
67      }
68
69      return sockfd;
70  }
71
72  int authConn(const int socket, const char* user, const char* pass) {
73      char answer[MAX_LENGTH];
74
75      // Send user command
```

```
76      dprintf(socket, "user %s\n", user);
77      if (readResponse(socket, answer) != SV_READY4PASS) {
78          printf("Unknown user '%s'. Abort.\n", user);
79          exit(EXIT_FAILURE);
80      }
81
82      // Send password command
83      dprintf(socket, "pass %s\n", pass);
84      return readResponse(socket, answer);
85  }
86
87  int passiveMode(const int socket, char *ip, int *port) {
88      char answer[MAX_LENGTH];
89      int ipParts[4], portParts[2];
90
91      // Send PASV command
92      dprintf(socket, "PASV\n");
93      if (readResponse(socket, answer) != SV_PASSIVE) return -1;
94
95      // Parse the response
96      sscanf(answer, PASSIVE_REGEX, &ipParts[0], &ipParts[1], &ipParts[2],
    &ipParts[3], &portParts[0], &portParts[1]);
97
98      // Calculate the port number
99      *port = portParts[0] * 256 + portParts[1];
100
101     // Format the IP address
102     sprintf(ip, "%d.%d.%d.%d", ipParts[0], ipParts[1], ipParts[2],
    ipParts[3]);
103
104     return SV_PASSIVE;
105 }
106
107 int readResponse(const int socket, char* buffer) {
108     char byte;
109     int index = 0, responseCode;
110     ResponseState state = START;
111     memset(buffer, 0, MAX_LENGTH);
112
113     while (state != END) {
114         read(socket, &byte, 1);
115
116         if (state == START) {
117             if (byte == ' ') state = SINGLE;
118             else if (byte == '-') state = MULTIPLE;
```

```
119             else if (byte == '\n') state = END;
120             else buffer[index++] = byte;
121         } else if (state == SINGLE) {
122             if (byte == '\n') state = END;
123             else buffer[index++] = byte;
124         } else if (state == MULTIPLE) {
125             if (byte == '\n') {
126                 memset(buffer, 0, MAX_LENGTH);
127                 state = START;
128                 index = 0;
129             } else buffer[index++] = byte;
130         }
131     }
132 
133     sscanf(buffer, RESPCODE_REGEX, &responseCode);
134     return responseCode;
135 }
136 
137 int requestResource(const int socket, char *resource) {
138     char answer[MAX_LENGTH];
139 
140     // Send RETR command
141     dprintf(socket, "retr %s\n", resource);
142 
143     return readResponse(socket, answer);
144 }
145 
146 int getResource(const int socketA, const int socketB, char *filename) {
147     FILE *fd = fopen(filename, "wb");
148     if (fd == NULL) {
149         perror("Error opening or creating file");
150         exit(EXIT_FAILURE);
151     }
152 
153     char buffer[MAX_LENGTH];
154     ssize_t bytes;
155     while ((bytes = read(socketB, buffer, MAX_LENGTH)) > 0) {
156         if (fwrite(buffer, 1, bytes, fd) != bytes) {
157             perror("Error writing to file");
158             fclose(fd);
159             exit(EXIT_FAILURE);
160         }
161     }
162 
163     fclose(fd);
```

```
164
165     return readResponse(socketA, buffer);
166 }
167
168 int closeConnection(const int socketA, const int socketB) {
169     char answer[MAX_LENGTH];
170
171     // Send QUIT command
172     dprintf(socketA, "quit\n");
173     if (readResponse(socketA, answer) != SV_GOODBYE) return -1;
174
175     // Close the sockets
176     int closeA = close(socketA);
177     int closeB = close(socketB);
178
179     return closeA || closeB;
180 }
181
182 int main(int argc, char *argv[]) {
183     if (argc != 2) {
184         printf("Usage: ./download ftp://[<user>:<password>@]<host>/<url-
    path>\n");
185         exit(EXIT_FAILURE);
186     }
187
188     struct URL url;
189     memset(&url, 0, sizeof(url));
190     if (parse(argv[1], &url) != 0) {
191         printf("Parse error. Usage: ./download
    ftp://[<user>:<password>@]<host>/<url-path>\n");
192         exit(EXIT_FAILURE);
193     }
194
195     printf("Host: %s\nResource: %s\nFile: %s\nUser: %s\nPassword: %s\nIP
    Address: %s\n", url.host, url.resource, url.file, url.user, url.password,
    url.ip);
196
197     char answer[MAX_LENGTH];
198     int socketA = createSocket(url.ip, FTP_PORT);
199     if (socketA < 0 || readResponse(socketA, answer) != SV_READY4AUTH) {
200         printf("Socket to '%s' and port %d failed\n", url.ip, FTP_PORT);
201         exit(EXIT_FAILURE);
202     }
203
204     if (authConn(socketA, url.user, url.password) != SV_LOGINSUCCESS) {
```

```
205         printf("Authentication failed with username = '%s' and password =
    '%s'.\n", url.user, url.password);
206         exit(EXIT_FAILURE);
207     }
208 
209     int port;
210     char ip[MAX_LENGTH];
211     if (passiveMode(socketA, ip, &port) != SV_PASSIVE) {
212         printf("Passive mode failed\n");
213         exit(EXIT_FAILURE);
214     }
215 
216     int socketB = createSocket(ip, port);
217     if (socketB < 0) {
218         printf("Socket to '%s:%d' failed\n", ip, port);
219         exit(EXIT_FAILURE);
220     }
221 
222     if (requestResource(socketA, url.resource) != SV_READY4TRANSFER) {
223         printf("Unknown resouce '%s' in '%s:%d'\n", url.resource, ip, port);
224         exit(EXIT_FAILURE);
225     }
226 
227     if (getResource(socketA, socketB, url.file) != SV_TRANSFER_COMPLETE) {
228         printf("Error transfering file '%s' from '%s:%d'\n", url.file, ip,
    port);
229         exit(EXIT_FAILURE);
230     }
231 
232     if (closeConnection(socketA, socketB) != 0) {
233         printf("Sockets close error\n");
234         exit(EXIT_FAILURE);
235     }
236 
237     return 0;
238 }
```

Anexo 2: Comandos

2.1 - Experiência 1

- TuxY3: Ativar eth0 e configurar IP com ifconfig eth0 172.16.Y0.1/24

- TuxY4: Ativar eth0 e configurar IP com ifconfig eth0 172.16.Y0.254/24

- Teste de conectividade: TuxY53 pinga o TuxY4 com ping 172.16.Y0.254

- Teste de conectividade: TuxY3 pinga o TuxY4 com ping 172.16.Y0.1

- Teste e limpeza da tabela ARP: TuxY3 executa arp -a, remove a entrada com arp -d 172.16.Y0.254 e verifica a tabela com arp -a (vazia).

2.2 - Experiência 2

- TuxY2: Ativar eth0 e configurar IP com ifconfig eth0 172.16.Y1.1/24

- Console do Switch:

/system reset-configuration

/interface bridge add name=bridgeY0

/interface bridge add name=bridgeY1

/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=bridgeY0 interface=ether1

/interface bridge port add bridge=bridgeY0 interface=ether2

/interface bridge port add bridge=bridgeY1 interface=ether3

/interface bridge port print

- Teste de conectividade: TuxY3 pinga o TuxY4 com ping 172.16.Y0.254 (responde).
- Teste de conectividade: TuxY3 pinga o TuxY4 com ping 172.16.Y1.1 (não responde).
- Teste de conectividade: TuxY3 ping -b 172.16.Y0.255 (broadcast).
- Teste de conectividade: TuxY4 ping -b 172.16.Y1.255 (broadcast).

2.3 - Experiência 3

- TuxY4: Ativar eth1 e configurar IP com ifconfig eth1 172.16.Y1.253/24

- Console do Switch:

/interface bridge port remove [find interface=ether4]

/interface bridge port add bridge=bridgeY1 interface=ether4

- Configuração de Redirecionamento e Roteamento no TuxY4:

echo 1 > /proc/sys/net/ipv4/ip_forward

echo 0 > /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts

- TuxY2: Adicionar rota para a rede: route add -net 172.16.Y0.0/24 gw 172.16.Y1.253
- TuxY3: Adicionar rota para a rede: route add -net 172.16.Y1.0/24 gw 172.16.Y0.253
- Teste de conectividade: TuxY3 pinga com ping 172.16.Y0.254 (responde).
- Teste de conectividade: TuxY3 pinga com ping 172.16.Y1.253 (responde).
- Teste de conectividade: TuxY3 pinga com ping 172.16.Y1.1 (responde).

- Limpar arp table:

TuxY2:

arp -d 172.16.Y1.253

TuxY3:

arp -d 172.16.Y0.254

TuxY4:

arp -d 172.16.Y0.1

arp -d 172.16.Y1.1

2.4 - Experiência 4

- Console do Switch:

/interface bridge port remove [find interface=ether5]

/interface bridge port add bridge=bridgeY1 interface=ether5

- Console do Router:

/system reset-configuration

/ip address add address=172.16.1.Y9/24 interface=ether1

/ip address add address=172.16.Y1.254/24 interface=ether2

/ip route add dst-address=172.16.Y0/24 gateway=172.16.Y1.253

/ip route add dst-address=0.0.0.0/0 gateway=172.16.1.254

- TuxY2: route add default gw 172.16.Y1.254
- TuxY3: route add default gw 172.16.Y0.254
- TuxY4: route add default gw 172.16.Y1.254
- Teste de conectividade: TuxY3 pinga com ping 172.16.Y0.254 (responde).
- Teste de conectividade: TuxY3 pinga com ping 172.16.Y1.1 (responde).
- Teste de conectividade: TuxY3 pinga com ping 172.16.Y1.254 (responde).
- TuxY2:

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

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

route del -net 172.16.Y0.0 gw 172.16.Y1.253 netmask 255.255.255.0

ping 172.16.Y0.1 (responde)

traceroute -n 172.16.Y0.1

route add -net 172.16.Y0.0/24 gw 172.16.Y1.253

traceroute -n 172.16.Y0.1

echo 1 > /proc/sys/net/ipv4/conf/eth0/accept_redirects

echo 1 > /proc/sys/net/ipv4/conf/all/accept_redirects

- Teste de conectividade: TuxY3 pinga com ping 172.16.1.254 (responde).
- Console do Switch:

/ip firewall nat disable 0

- Teste de conectividade: TuxY3 pinga com ping 172.16.1.254 (não responde).

- Console do Switch:

/ip firewall nat enable 0

2.5 - Experiência 5

- Em todos os Tuxs, no ficheiro /etc/resolv.conf adicionar a seguinte linha: nameserver 172.16.1.1

- DNS configurado: ping google.com

Anexo 3: Capturas do Wireshark

3.1 - Experiência 1

10	10.998354470	HewlettPacka_61:2f:…	Broadcast	ARP	42 Who has 172.16.60.254? Tell 172.16.60.1
11	10.998459581	HewlettPacka_c5:61:…	HewlettPacka_61:2f:…	ARP	60 172.16.60.254 is at 00:21:5a:c5:61:bb
12	10.998476972	172.16.60.1	172.16.60.254	ICMP	98 Echo (ping) request id=0x2b2c, seq=1/256, ttl=64 (reply in 13)
13	10.998561410	172.16.60.254	172.16.60.1	ICMP	98 Echo (ping) reply id=0x2b2c, seq=1/256, ttl=64 (request in 12)
14	12.011130047	172.16.60.1	172.16.60.254	ICMP	98 Echo (ping) request id=0x2b2c, seq=2/512, ttl=64 (reply in 15)
15	12.011231806	172.16.60.254	172.16.60.1	ICMP	98 Echo (ping) reply id=0x2b2c, seq=2/512, ttl=64 (request in 14)

3.2 - Experiência 2

12	10.998476972	172.16.60.1	172.16.60.254	ICMP	98 Echo (ping) request id=0x2b2c, seq=1/256, ttl=64 (reply in 13)
13	10.998561410	172.16.60.254	172.16.60.1	ICMP	98 Echo (ping) reply id=0x2b2c, seq=1/256, ttl=64 (request in 12)
14	12.011130047	172.16.60.1	172.16.60.254	ICMP	98 Echo (ping) request id=0x2b2c, seq=2/512, ttl=64 (reply in 15)
15	12.011231806	172.16.60.254	172.16.60.1	ICMP	98 Echo (ping) reply id=0x2b2c, seq=2/512, ttl=64 (request in 14)
16	12.013392905	Routerboardc_1c:8b:…	Spanning-tree-(for-…	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8b:e3 Cost = 0 Port = 0x0002

16	28.760189338	172.16.61.1	172.16.61.255	ICMP	98 Echo (ping) request id=0x2fbd, seq=1/256, ttl=64 (no response found!)
17	29.763251180	172.16.61.1	172.16.61.255	ICMP	98 Echo (ping) request id=0x2fbd, seq=2/512, ttl=64 (no response found!)
18	30.032603868	Routerboardc_1c:8b:…	Spanning-tree-(for-…	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8b:e8 Cost = 0 Port = 0x0001
19	30.787249832	172.16.61.1	172.16.61.255	ICMP	98 Echo (ping) request id=0x2fbd, seq=3/768, ttl=64 (no response found!)
20	31.811250790	172.16.61.1	172.16.61.255	ICMP	98 Echo (ping) request id=0x2fbd, seq=4/1024, ttl=64 (no response found!)

3.3 - Experiência 3

13	15.390013284	HewlettPacka_61:2f:…	Broadcast	ARP	60 Who has 172.16.60.254? Tell 172.16.60.1
14	15.390040382	HewlettPacka_c5:61:…	HewlettPacka_61:2f:…	ARP	42 172.16.60.254 is at 00:21:5a:c5:61:bb
15	15.390155271	172.16.60.1	172.16.61.1	ICMP	98 Echo (ping) request id=0x3923, seq=1/256, ttl=64 (reply in 16)
16	15.390456566	172.16.61.1	172.16.60.1	ICMP	98 Echo (ping) reply id=0x3923, seq=1/256, ttl=63 (request in 15)

3.4 - Experiência 4

20	19.253596524	HewlettPacka_61:2d:…	HewlettPacka_61:2f:…	ARP	42 Who has 172.16.10.254? Tell 172.16.10.1
21	19.253747940	HewlettPacka_61:2f:…	HewlettPacka_61:2d:…	ARP	60 172.16.10.254 is at 00:21:5a:61:2f:24
22	19.332858911	HewlettPacka_61:2f:…	HewlettPacka_61:2d:…	ARP	60 Who has 172.16.10.1? Tell 172.16.10.254
23	19.332871762	HewlettPacka_61:2d:…	HewlettPacka_61:2f:…	ARP	42 172.16.10.1 is at 00:21:5a:61:2d:ef
24	20.009954682	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8c:9a Cost = 0 Port = 0x8001
25	22.012126301	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8c:9a Cost = 0 Port = 0x8001
26	24.014312097	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8c:9a Cost = 0 Port = 0x8001
27	26.016053283	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8c:9a Cost = 0 Port = 0x8001
28	28.018653378	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8c:9a Cost = 0 Port = 0x8001
29	30.020828559	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8c:9a Cost = 0 Port = 0x8001
30	32.022992984	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8c:9a Cost = 0 Port = 0x8001
31	34.025186183	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8c:9a Cost = 0 Port = 0x8001
32	35.078864878	172.16.10.1	172.16.11.1	ICMP	98 Echo (ping) request id=0x31ab, seq=1/256, ttl=64 (reply in 33)
33	35.079178885	172.16.11.1	172.16.10.1	ICMP	98 Echo (ping) reply id=0x31ab, seq=1/256, ttl=63 (request in 32)

35	36.085631308	172.16.10.1	172.16.11.1	ICMP	98 Echo (ping) request id=0x31ab, seq=2/512, ttl=64 (reply in 36)
36	36.085886160	172.16.11.1	172.16.10.1	ICMP	98 Echo (ping) reply id=0x31ab, seq=2/512, ttl=63 (request in 35)
37	37.109629430	172.16.10.1	172.16.11.1	ICMP	98 Echo (ping) request id=0x31ab, seq=3/768, ttl=64 (reply in 38)
38	37.109895805	172.16.11.1	172.16.10.1	ICMP	98 Echo (ping) reply id=0x31ab, seq=3/768, ttl=63 (request in 37)

48	46.165635260	172.16.10.1	172.16.11.254	ICMP	98 Echo (ping) request id=0x31b2, seq=2/512, ttl=64 (reply in 49)
49	46.165917349	172.16.11.254	172.16.10.1	ICMP	98 Echo (ping) reply id=0x31b2, seq=2/512, ttl=63 (request in 48)
50	47.189635407	172.16.10.1	172.16.11.254	ICMP	98 Echo (ping) request id=0x31b2, seq=3/768, ttl=64 (reply in 51)
51	47.189936703	172.16.11.254	172.16.10.1	ICMP	98 Echo (ping) reply id=0x31b2, seq=3/768, ttl=63 (request in 50)

```
 5 2.423954532   172.16.11.1      172.16.10.1      ICMP      98 Echo (ping) request  id=0x2f7a, seq=2/512, ttl=64 (reply in 7)
 6 2.424122495   172.16.11.254    172.16.11.1      ICMP     126 Redirect             (Redirect for host)
 7 2.424322306   172.16.10.1      172.16.11.1      ICMP      98 Echo (ping) reply    id=0x2f7a, seq=2/512, ttl=63 (request in 5)
 8 3.447954620   172.16.11.1      172.16.10.1      ICMP      98 Echo (ping) request  id=0x2f7a, seq=3/768, ttl=64 (reply in 10)
 9 3.448110222   172.16.11.254    172.16.11.1      ICMP     126 Redirect             (Redirect for host)
10 3.448347117   172.16.10.1      172.16.11.1      ICMP      98 Echo (ping) reply    id=0x2f7a, seq=3/768, ttl=63 (request in 8)
```

3.5 - Experiência 6

```
10 2.512028859  172.16.50.1      193.136.28.10    DNS       76 Standard query 0xeb7f A netlab1.fe.up.pt
11 2.512799479  193.136.28.10    172.16.50.1      DNS      286 Standard query response 0xeb7f A netlab1.fe.up.pt A 192.168.109.136 NS cns2.fe.up.pt NS ns2.f
12 2.512867923  172.16.50.1      192.168.109.136  TCP       74 39632 → 21 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=2668342817 TSecr=0 WS=128
13 2.513718372  192.168.109.136  172.16.50.1      TCP       74 21 → 39632 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM TSval=1469991566 TSecr=2
14 2.513738276  172.16.50.1      192.168.109.136  TCP       66 39632 → 21 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=2668342818 TSecr=1469991566
15 2.515746402  192.168.109.136  172.16.50.1      FTP      100 Response: 220 Welcome to netlab-FTP server
16 2.515755970  172.16.50.1      192.168.109.136  TCP       66 39632 → 21 [ACK] Seq=1 Ack=35 Win=64256 Len=0 TSval=2668342820 TSecr=1469991568
17 2.515794592  172.16.50.1      192.168.109.136  FTP       77 Request: USER rcom
18 2.516490344  192.168.109.136  172.16.50.1      TCP       66 21 → 39632 [ACK] Seq=35 Ack=12 Win=65280 Len=0 TSval=1469991569 TSecr=2668342820
19 2.516567657  192.168.109.136  172.16.50.1      FTP      100 Response: 331 Please specify the password.
20 2.516593778  172.16.50.1      192.168.109.136  FTP       77 Request: PASS rcom
21 2.517118490  192.168.109.136  172.16.50.1      TCP       66 21 → 39632 [ACK] Seq=69 Ack=23 Win=65280 Len=0 TSval=1469991569 TSecr=2668342820
22 2.526107738  192.168.109.136  172.16.50.1      FTP       89 Response: 230 Login successful.
23 2.526152157  172.16.50.1      192.168.109.136  FTP       72 Request: PASV
24 2.526860620  192.168.109.136  172.16.50.1      TCP       66 21 → 39632 [ACK] Seq=92 Ack=29 Win=65280 Len=0 TSval=1469991579 TSecr=2668342830
25 2.527056034  192.168.109.136  172.16.50.1      FTP      120 Response: 227 Entering Passive Mode (192,168,109,136,184,191).
26 2.527112395  172.16.50.1      192.168.109.136  TCP       74 54120 → 47295 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=2668342831 TSecr=0 WS=128
27 2.527692560  192.168.109.136  172.16.50.1      TCP       74 47295 → 54120 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM TSval=1469991580 TSec
28 2.527708414  172.16.50.1      192.168.109.136  TCP       66 54120 → 47295 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=2668342832 TSecr=1469991580
29 2.527719309  172.16.50.1      192.168.109.136  FTP       87 Request: RETR files/crab.mp4
30 2.528387963  192.168.109.136  172.16.50.1      TCP       66 21 → 39632 [ACK] Seq=146 Ack=50 Win=65280 Len=0 TSval=1469991580 TSecr=2668342832
31 2.528542869  192.168.109.136  172.16.50.1      FTP      144 Response: 150 Opening BINARY mode data connection for files/crab.mp4 (88123184 bytes).
32 2.528842625  192.168.109.136  172.16.50.1      FTP-DA… 1514 FTP Data: 1448 bytes (PASV) (RETR files/crab.mp4)
33 2.528855336  172.16.50.1      192.168.109.136  TCP       66 54120 → 47295 [ACK] Seq=1 Ack=1449 Win=64128 Len=0 TSval=2668342833 TSecr=1469991581
34 2.528963519  192.168.109.136  172.16.50.1      FTP-DA… 1514 FTP Data: 1448 bytes (PASV) (RETR files/crab.mp4)
```

```
46479 10.021393339  172.16.50.1      192.168.109.136  TCP       66 54120 → 47295 [ACK] Seq=1 Ack=88123186 Win=613504 Len=0 TSval=2668350325 TSecr=1469999072
46480 10.022207889  192.168.109.136  172.16.50.1      FTP       90 Response: 226 Transfer complete.
46481 10.022217527  172.16.50.1      192.168.109.136  TCP       66 39632 → 21 [ACK] Seq=50 Ack=248 Win=64256 Len=0 TSval=2668350326 TSecr=1469999074
46482 10.022255520  172.16.50.1      192.168.109.136  FTP       72 Request: QUIT
46483 10.023637945  192.168.109.136  172.16.50.1      TCP       66 21 → 39632 [ACK] Seq=248 Ack=56 Win=65280 Len=0 TSval=1469999075 TSecr=2668350326
46484 10.023678033  192.168.109.136  172.16.50.1      FTP       80 Response: 221 Goodbye.
46485 10.023708065  192.168.109.136  172.16.50.1      TCP       66 21 → 39632 [FIN, ACK] Seq=262 Ack=56 Win=65280 Len=0 TSval=1469999075 TSecr=2668350326
46486 10.023709461  172.16.50.1      192.168.109.136  TCP       66 54120 → 47295 [FIN, ACK] Seq=1 Ack=88123186 Win=640512 Len=0 TSval=2668350328 TSecr=14699990
46487 10.023728179  172.16.50.1      192.168.109.136  TCP       66 39632 → 21 [FIN, ACK] Seq=56 Ack=263 Win=64256 Len=0 TSval=2668350328 TSecr=1469999075
46488 10.024301360  192.168.109.136  172.16.50.1      TCP       66 47295 → 54120 [ACK] Seq=88123186 Ack=2 Win=65280 Len=0 TSval=1469999076 TSecr=2668350328
46489 10.024335582  192.168.109.136  172.16.50.1      TCP       66 21 → 39632 [ACK] Seq=263 Ack=57 Win=65280 Len=0 TSval=1469999076 TSecr=2668350328
46490 12.012645019  Routerboardc_1c:95:… Spanning-tree-(for-… STP       60 RST. Root = 32768/0/c4:ad:34:1c:95:d0  Cost = 0  Port = 0x8001
46491 12.235609210  172.16.50.1      193.136.28.10    DNS       86 Standard query 0x322d PTR 93.243.107.34.in-addr.arpa
46492 12.236458122  193.136.28.10    172.16.50.1      DNS      444 Standard query response 0x322d PTR 93.243.107.34.in-addr.arpa PTR 93.243.107.34.bc.googleuser
```