

Aplicação de download e configuração e estudo uma rede

Relatório



Universidade do Porto

Faculdade de Engenharia

FEUP

Redes de Computadores

3º ano

Mestrado Integrado em Engenharia Informática e Computação

Turma 4

Carolina Moreira	201303494	up201303494@fe.up.pt
Daniel Fazeres	201502846	up201502846@fe.up.pt
José Peixoto	200603103	ei12134@fe.up.pt

25 de Janeiro de 2017

Conteúdo

1	Introdução	1
2	Equipamento: Switch Cisco	2
3	Equipamento: Router Cisco	2
4	Experiência 1: Configuração de uma rede IP	2
4.1	Configuração de interfaces	2
4.2	Pacotes ARP e endereços MAC	4
4.3	Comandos PING	5
4.4	Interface loopback	5
5	Experiência 2: Duas VLANs num switch	5
5.1	Como configurar uma VLAN?	5
5.2	Domínios de broadcast	5
6	Experiência 3: Configurar um router em Linux	5
6.1	Rotas em cada um dos computadores	5
6.2	Entradas nas tabelas de forwarding	5
6.3	Mensagens ARP observadas	5
6.4	Endereços IP e MAC dos pacotes ICMP	5
7	Experiência 4: Configurar um router em Linux	5
7.1	Como configurar uma rota estática num router comercial? . .	5
7.2	Caminhos percorridos pelos pacotes	5
7.3	NAT	5
7.4	Como configurar NAT num router comercial?	5
8	Experiência 5: DNS	5
8.1	Como configurar um serviço DNS?	5
8.2	Pacotes usados por DNS	5
9	Experiência 7: Aplicação de download	5
10	Experiência 7: Implementação de NAT em Linux	8
11	Conclusões	8
A	Código fonte	9
A.1	Camada de aplicação	9

Resumo

No âmbito da unidade curricular de Redes de Computadores, foi-nos proposto o desenvolvimento de uma aplicação que testasse um protocolo de ligação de dados criado de raiz, transferindo um ficheiro recorrendo à porta de série *RS – 232*. O trabalho permitiu praticar conceitos teóricos no desenho de um protocolo de ligação de dados como o sincronismo e delimitação de tramas, controlo de erros, controlo de fluxo recurso a mecanismos de transparência de dados na transmissão assíncrona.

Findo o projeto, notou-se a importância dos mecanismos que asseguram tolerância a falhas fornecidos pela camada de ligação de dados, uma vez que a camada física não é realmente fiável.

1 Introdução

O objetivo do trabalho realizado nas aulas laboratoriais da disciplina de Redes de Computadores é a implementação de um protocolo de ligação de dados que permita praticar conhecimentos acerca de transmissões de dados entre computadores, programando em baixo nível as características comuns a este tipos de protocolos como a transparência na transmissão de dados de forma assíncrona e organização da informação sob a forma de tramas.

2 Equipamento: Switch Cisco

3 Equipamento: Router Cisco

4 Experiência 1: Configuração de uma rede IP

Nesta experiência, configuraremos apenas uma rede simples de dois computadores ligados entre si por um switch. Esta rede usará o conjunto de protocolos que constituem, em parte, a arquitectura da mais conhecida rede de todas, a Internet. Como é sabido, o protocolo que gere a camada de rede da Internet é o IP (Internet Protocol). Para além do próprio protocolo IP, a Internet usa vários outros protocolos auxiliares para regular o seu funcionamento, nomeadamente ARP, ICMP, e DHCP. O protocolo DHCP é usado para negociar automaticamente a atribuição de um endereço IP a qualquer elemento de uma rede que permita uma identificação única perante outros membros. Este protocolo não será usado, e em vez disso atribuir-se-á manualmente e estaticamente endereços IP a cada um dos computadores das redes desta e subsequente experiências deste trabalho, o que é exequível visto tratarem-se de redes de pequena dimensão, o que não seria o caso noutras instalações de aplicação prática no mundo real.

Os endereços desta experiência serão divididos, como é feito neste protocolo de rede, numa parte respeitante à subrede (subnet), e outra que indicará o endereço de cada computador dentro desta rede. Um pequeno esquema do resultado pretendido é indicado na próxima imagem.

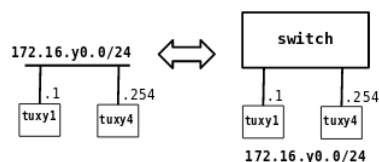


Figura 1: Rede da 1a Experiência

4.1 Configuração de interfaces

Na configuração desta rede, começamos por ligar apenas os computadores TUX1 e TUX4 ao switch Cisco Catalyst 3560. Cada um dos computadores tem duas interfaces expostas, para além da loopback. A interface loopback é uma interface apresentada pelo sistema operativo como qualquer outra interface, mas possuindo o endereço especial `127.0.0.1` (IPv4) e `::1` (IPv6). Pacotes dirigidos a este endereço são direccionados para o próprio computador. Cada interface pode ser configurada independentemente das outras, e o software de rede do sistema operativo encarregar-se-á de encaminhar o tráfego de rede para uma ou outra consoante a rede a que pertence e o

destino do tráfego. Por esta razão, e uma vez que a configuração será manual, atribuiremos um endereço IP juntamente com uma máscara de rede à interface eth0 destes TUXs.

O primeiro passo é desactivar a interface eth1 com o comando

```
ifconfig eth1 down
```

Numa nota à parte, este relatório foi completado depois de uma reconfiguração dos computadores por parte dos responsáveis pela manutenção do laboratório, pelo que, no caso de alguns TUXs, o programa usado para configurar as interfaces foi o *ip* e não o *ifconfig*, para o qual os comandos são diferentes. Para desactivar esta interface com o *ip* usámos o comando

```
ip link set eth1 down
```

Seguidamente, removemos os IPs atribuídos ao TUX1 com:

```
ip addr flush dev eth0
```

após o que atribuímos o endereço pretendido:

```
ip addr add 172.16.10.1/24 dev eth0
```

e confirmamos o resultado com:

```
ip addr show [dev eth0]
```

Nos resultados podemos ver a interface loopback já mencionada, o IP que configurámos na interface eth0, e que a MTU desta interface é 1500 bytes, que é o tamanho máximo de uma frame em ethernet, que é o tipo de rede (camada de rede) que estamos a utilizar. Realizamos o mesmo procedimento no TUX4, atribuindo o endereço IP com:

```
ifconfig eth0 172.16.10.254/24
```

O próximo passo é configurar o router que liga os dois computadores. Para aceder à sua configuração, ligamos o seu terminal a um adaptador cuja saída é uma ligação em série, que depois é ligada à entrada de porta de série do TUX1, exposta no ficheiro */dev/ttyS0* do mesmo. Para termos a certeza que não existem configurações já definidas a interferir no nosso trabalho, fazemos reset do switch. Utilizando o programa *gtkterm* para comunicar com a sua linha de comandos, começamos por entrar em modo de configuração com o comando *enable*, para o qual precisamos de introduzir a password do switch. Escrevemos, depois, os comandos:

```
copy tftp : //192.168.109.1/2 startup - config
```

```
delete flash : vlan.dat
```

4.2 Pacotes ARP e endereços MAC

Em qualquer rede deste tipo, o protocolo *ARP* (Address Resolution Protocol) desempenhará a função necessária de comunicar a atribuição de endereços IP entre vizinhos – NICs (Network Interface Cards), como placas Ethernet, não compreendem endereços IP, emitindo apenas tramas endereçadas pelo ao que se chamam endereços MAC (Media Access Control), atribuídos pela IEEE e garantidamente únicos em qualquer parte do mundo. Quando um computador numa rede deste tipo deseja saber a que endereço MAC corresponde um determinado endereço IP, emite (através de broadcast) um pacote ARP com a pergunta "a quem pertence este endereço IP?". De seguida, o dono do endereço referido responde à questão com outro pacote. O emissor do primeiro pedido de ARP também inclui o seu próprio endereço IP na mensagem, eliminando a necessidade do segundo membro da rede perguntar ele próprio para que endereço de IP deve responder. De notar que os pacotes utilizados nesta troca são pacotes IP, e não simplesmente tramas de Ethernet.

Cada membro da rede mantém em cache uma tabela de atribuições de endereços IP, mas esta deve ter uma duração finita, para lidar com reconfigurações na rede e na sua topologia. Quando um computador é configurado inicialmente, pode emitir um pedido ARP pelo seu próprio endereço. Não deverá ocorrer nenhuma resposta, mas todos os seus vizinhos deverão receber a informação relativa ao endereço IP do emissor.

Este protocolo é definido no RFC 826.

4.3 Comandos PING

4.4 Interface loopback

5 Experiência 2: Duas VLANs num switch

5.1 Como configurar uma VLAN?

5.2 Domínios de broadcast

6 Experiência 3: Configurar um router em Linux

6.1 Rotas em cada um dos computadores

6.2 Entradas nas tabelas de forwarding

6.3 Mensagens ARP observadas

6.4 Endereços IP e MAC dos pacotes ICMP

7 Experiência 4: Configurar um router em Linux

7.1 Como configurar uma rota estática num router comercial?

7.2 Caminhos percorridos pelos pacotes

7.3 NAT

7.4 Como configurar NAT num router comercial?

8 Experiência 5: DNS

8.1 Como configurar um serviço DNS?

8.2 Pacotes usados por DNS

9 Experiência 7: Aplicação de download

A primeira parte do segundo trabalho laboratorial consistiu no desenvolvimento de uma aplicação de download recorrendo ao protocolo de transferência de ficheiros FTP especificado pelo RFC959. Para este efeito foi também necessário resolver o endereço de IP para um dado URL de acordo com especificação RFC1738. Separaram-se as componentes do parsing do URL e do cliente de download usando o protocolo FTP respectivamente nos ficheiros `url.c` e `ftp.c`.

Casos de uso principais

O programa implementa uma versão básica de um cliente de FTP com suporte para download de ficheiros de forma anónima ou para um dado par utilizador e password, introduzidos antecipadamente ao caminho de URL do ficheiro. Apesar da introdução do nome de utilizador seguido de password serem facultativos, em alguns casos tornam-se obrigatórios na transferência com sucesso de um ficheiro por FTP caso este não esteja disponível de forma pública e requeira autenticação por parte do utilizador.

```
[user@localhost ftp-downloader]$ ./bin/ftp-downloader
Usage: ./bin/ftp-downloader ftp://[<user>:<password>@]<host>/<url-path>
```

Figura 2: Utilização do programa

Análise de URL

Após a leitura e compreensão do RFC1738, desenhou-se uma estrutura de dados com a finalidade de armazenar a informação extraída da análise de um link URL recebido pela linha de comandos. Quando fornecidos, são guardados na estrutura referida, o nome de utilizador, a password, o host, ip, path e nome do ficheiro em arrays de caracteres independentes. Para além disso, é também definida a porta 21 como a predefinida na ligação de controlo do protocolo de FTP. À semelhança do que é referido no RFC1738, caracteres capitalizados são interpretados como caracteres minúsculos, admitindo ftp da mesma forma que FTP.

Para se poder resolver o endereço de IP armazenado na estrutura de URL, é feita uma chamada à função `gethostbyname` que permite a obtenção do endereço de uma máquina a partir do nome e retorna uma estrutura do tipo `hostent` contendo o endereço na variável `h_addr` que é posteriormente convertido para um array de chars com o auxílio da função `inet_ntoa`.

Cliente de FTP

O cliente de FTP liga-se através de um socket TCP ao servidor de FTP identificado pelo endereço de IP acima mencionado e à porta 21 e estabelece uma ligação de controlo de comunicação, comunicando com uma sequência de comandos de transferência FTP que lhe são enviados ao estilo do protocolo Telnet:

USER `user` envio do nome de utilizador sob a forma de uma string que identifica o utilizador no servidor remoto.

PASS `pass` envio da palavra passe completando a identificação no sistema de identificação e controlo de acesso do servidor.

CWD path indicação do directório que contém o ficheiro requerido para download e sobre o qual se pretende trabalhar.

PASV comando que pede ao servidor para ficar à escuta numa porta de dados diferente da porta usada pelo serviço de controlo. Este comando recebe uma resposta que contém o endereço e a porta na qual o servidor ficou à escuta para poder estabelecer uma outra ligação TCP usada para transferência de dados.

RETR filename comando **retrieve** que pede ao servidor que inicie a transmissão de uma cópia do ficheiro especificado pelo campo **filename** usando a nova ligação estabelecida ao endereço e porta recebidos pelo comando anterior.

Uma vez estabelecida a ligação dedicada de transmissão de dados, estes vão sendo recebidos de forma ordenada pelo cliente de FTP e armazenados em disco.

Casos de uso

Um possível caso de uso pode ser o download de um ficheiro de forma anónima do URL `ftp://ftp.up.pt/pub/CentOS/filelist.gz`:

```
[user@localhost ftp-downloader]$ ./bin/ftp-downloader ftp://ftp.up.pt/pub/CentOS/filelist.gz

The IP received to ftp.up.pt was 193.136.37.8
220 Bem-vindo à Universidade do Porto
Bytes sent: 16
Info: USER anonymous

331 Please specify the password.
Bytes sent: 7
Info: PASS

230 Login successful.
Bytes sent: 17
Info: CWD pub/CentOS/

250 Directory successfully changed.
Bytes sent: 6
Info: PASV

227 Entering Passive Mode (193,136,37,8,169,170)
IP: 193.136.37.8
PORT: 43434
Bytes sent: 18
Info: RETR filelist.gz

150 Opening BINARY mode data connection for filelist.gz (3722620 bytes).
226 File send OK.
Bytes sent: 6
Info: QUIT
```

Figura 3: Utilização anónima para download de um ficheiro

Também foi testado o download de um ficheiro quando é requerida a autenticação do utilizador no servidor de FTP:

```
[user@localhost ftp-downloader]$ ./bin/ftp-downloader ftp://ei12134[REDACTED]@tom.fe.up.pt/public_html/event_manager/index.php

The IP received to tom.fe.up.pt was 192.168.50.138
220 FTP for Alf/Tom/Crazy/Pinguim
Bytes sent: 14
Info: USER ei12134

331 Please specify the password.
Bytes sent: 20
Info: PASS [REDACTED]

230 Login successful.
Bytes sent: 32
Info: CWD public_html/event_manager/

250 Directory successfully changed.
Bytes sent: 6
Info: PASV

227 Entering Passive Mode (192,168,50,138,106,139).
IP: 192.168.50.138
PORT: 27275
Bytes sent: 16
Info: RETR index.php

150 Opening BINARY mode data connection for index.php (48 bytes).
226 Transfer complete.
Bytes sent: 6
Info: QUIT
```

Figura 4: Utilização autenticada no download de um ficheiro

Ligações TCP abertas pela aplicação

Ligação de controlo

Fases de abertura de uma ligação TCP

Mecanismo de ARQ em TCP

Congestionamento na ligação TCP

Influência de uma segunda ligação TCP

10 Experiência 7: Implementação de NAT em Linux

// TODO

11 Conclusões

O projeto pode ser sumariamente descrito pelo seu principal propósito que é o desenvolvimento de um protocolo de ligação de dados e o seu teste aquando da obtenção de sucesso na transferência de ficheiros entre dois

computadores, com mecanismos de recuperação face a algumas situações de erros.

Findo o projeto, consideramos que atingimos os objetivos definidos tendo também implementado alguns dos elementos de valorização especificados no guião. Esta abordagem prática permitiu também uma melhor consciência do funcionamento e dos problemas inerentes às redes comunicações entre computadores, abordados nas aulas teóricas.

Referências

- [1] Andrew S. Tanenbaum, David J. Wetherall, *Computer Networks*, Prentice Hall, 5th edition, 2011.
- [2] Network Working Group, *Architectural Principles of the Internet*, June 1996.

A Código fonte

A.1 Camada de aplicação

netlink.c

```
1 void receiver_stats()
2 #include <sys/types.h>
3 #include <sys/stat.h>
4 #include <fcntl.h>
5 #include <stdio.h>
6 #include <string.h>
7 #include <stdlib.h>
8 #include <termios.h>
9 #include <unistd.h>
10
11 #include "packets.h"
12 #include "file.h"
13 #include "netlink.h"
14 #include "serial_port.h"
15
16 struct file file_to_send;
17 int max_retries = 3;
18
19 void help(char **argv)
20 {
21     fprintf(stderr, "Usage: %s [OPTIONS] <serial port>\n",
22             argv[0]);
23     fprintf(stderr, "\n Program options:\n");
24     fprintf(stderr, " -t <FILEPATH>\t\t\ttransmit file over the
25             serial port\n");
26     fprintf(stderr, " -i\t\t\t\ttransmit data read from stdin\n
27             ");
```

```

25     fprintf(stderr, "  -b <BAUDRATE>\t\tbaudrate of the serial
        port\n");
26     fprintf(stderr,
27         "  -p <DATASIZE>\t\tmaximum bytes of data transfered
        each frame\n");
28     fprintf(stderr, "  -r <RETRY>\t\tnumber of retry attempts\
        n");
29 }
30
31 int parse_serial_port_arg(int index, char **argv)
32 {
33     if ((strcmp("/dev/ttyS0", argv[index]) != 0)
34         && (strcmp("/dev/ttyS1", argv[index]) != 0)
35         && (strcmp("/dev/ttyS4", argv[index]) != 0)) {
36         fprintf(stderr, "Error: bad serial port value\n");
37         return -1;
38     }
39
40     return index;
41 }
42
43 int parse_baudrate_arg(int baudrate_index, char **argv)
44 {
45     if (strcmp("B50", argv[baudrate_index]) == 0) {
46         serial_port_baudrate = B50;
47         return 0;
48     } else if (strcmp("B75", argv[baudrate_index]) == 0) {
49         serial_port_baudrate = B75;
50         return 0;
51     } else if (strcmp("B110", argv[baudrate_index]) == 0) {
52         serial_port_baudrate = B110;
53         return 0;
54     } else if (strcmp("B134", argv[baudrate_index]) == 0) {
55         serial_port_baudrate = B134;
56         return 0;
57     } else if (strcmp("B150", argv[baudrate_index]) == 0) {
58         serial_port_baudrate = B150;
59         return 0;
60     } else if (strcmp("B200", argv[baudrate_index]) == 0) {
61         serial_port_baudrate = B200;
62         return 0;
63     } else if (strcmp("B300", argv[baudrate_index]) == 0) {
64         serial_port_baudrate = B300;
65         return 0;
66     } else if (strcmp("B600", argv[baudrate_index]) == 0) {
67         serial_port_baudrate = B600;
68         return 0;
69     } else if (strcmp("B1200", argv[baudrate_index]) == 0) {
70         serial_port_baudrate = B1200;
71         return 0;
72     } else if (strcmp("B1800", argv[baudrate_index]) == 0) {
73         serial_port_baudrate = B1800;
74         return 0;
75     } else if (strcmp("B2400", argv[baudrate_index]) == 0) {

```

```

76     serial_port_baudrate = B2400;
77     return 0;
78 } else if (strcmp("B4800", argv[baurdate_index]) == 0) {
79     serial_port_baudrate = B4800;
80     return 0;
81 } else if (strcmp("B9600", argv[baurdate_index]) == 0) {
82     serial_port_baudrate = B9600;
83     return 0;
84 } else if (strcmp("B19200", argv[baurdate_index]) == 0) {
85     serial_port_baudrate = B19200;
86     return 0;
87 } else if (strcmp("B38400", argv[baurdate_index]) == 0) {
88     serial_port_baudrate = B38400;
89     return 0;
90 }
91 fprintf(stderr, "Error: bad serial port baudrate value\n")
92 ;
93 fprintf(stderr,
94     "Valid baudrates: B110, B134, B150, B200, B300, B600,
95     B1200, B1800, B2400, B4800, B9600, B19200, B38400\n"
96     );
97 return -1;
98 }
99
100 void parse_max_packet_size(int packet_size_index, char **
101     argv)
102 {
103     int val = atoi(argv[packet_size_index]);
104     if (val > FRAME_SIZE || val < 0)
105         max_data_transfer = FRAME_SIZE;
106     else
107         max_data_transfer = val;
108
109 #ifdef NETLINK_DEBUG_MODE
110     fprintf(stderr, "\nparse_max_packet_size:\n");
111     fprintf(stderr, "    max_packet_size=%d\n", max_data_transfer
112 );
113 #endif
114 }
115
116 void parse_max_retries(int packet_size_index, char **argv)
117 {
118     int val = atoi(argv[packet_size_index]);
119     if (val <= 0)
120         max_retries = 4;
121     else
122         max_retries = 1 + val;
123
124 #ifdef NETLINK_DEBUG_MODE
125     fprintf(stderr, "\nmax_retries:\n");
126     fprintf(stderr, "    max_retries=%d\n", max_retries);
127 #endif
128 }

```

```

125 int parse_flags(int* t_index, int* i_index, int* b_index,
126               int* p_index,
127               int* r_index, int argc, char **argv)
128 {
129     for (size_t i = 0; i < (argc - 1); i++) {
130         if ((strcmp("-t", argv[i]) == 0)) {
131             *t_index = i;
132         } else if ((strcmp("-i", argv[i]) == 0)) {
133             *i_index = i;
134         } else if ((strcmp("-b", argv[i]) == 0)) {
135             *b_index = i;
136         } else if ((strcmp("-p", argv[i]) == 0)) {
137             *p_index = i;
138         } else if ((strcmp("-r", argv[i]) == 0)) {
139             *r_index = i;
140         } else if ((argv[i][0] == '-')) {
141             return -1;
142         }
143     }
144     #ifdef NETLINK_DEBUG_MODE
145     fprintf(stderr, "\nparse_flags(): flag indexes\n");
146     fprintf(stderr, "  -t=%d\n  -i=%d\n  -b=%d\n  -p=%d\n  -r=%d\n",
147             *t_index, *i_index, *b_index, *p_index, *r_index);
148     #endif
149     return 0;
150 }
151
152 int parse_args(int argc, char **argv, int *is_transmitter)
153 {
154     #ifdef NETLINK_DEBUG_MODE
155     fprintf(stderr, "\nparse_args(): received arguments\n");
156     fprintf(stderr, "  argc=%d\n  argv=%s\n", argc, *argv);
157     #endif
158
159     if (argc < 2) {
160         return -1;
161     }
162
163     if (argc == 2)
164         return parse_serial_port_arg(1, argv);
165
166     int t_index = -1, i_index = -1, b_index = -1, p_index =
167         -1, r_index = -1;
168
169     if (parse_flags(&t_index, &i_index, &b_index, &p_index, &
170                   r_index, argc,
171                   argv)) {
172         fprintf(stderr, "Error: bad flag parameter\n");
173         return -1;
174     }
175
176     if (t_index > 0 && t_index < argc - 1) {

```

```

175     if (read_file_from_disk(argv[t_index + 1], &file_to_send
176         ) < 0) {
177         return -1;
178     }
179     *is_transmitter = 1;
180 } else {
181     if (i_index > 0 && i_index < argc - 1) {
182         if (read_file_from_stdin(&file_to_send) < 0) {
183             return -1;
184         }
185         *is_transmitter = 1;
186     }
187 }
188 if (b_index > 0 && b_index < argc - 1) {
189     if (parse_baudrate_arg(b_index + 1, argv) != 0) {
190         return -1;
191     }
192 }
193
194 if (p_index > 0 && p_index < argc - 1) {
195     parse_max_packet_size(p_index + 1, argv);
196 }
197
198 if (r_index > 0 && r_index < argc - 1) {
199     parse_max_retries(r_index + 1, argv);
200 }
201
202 return parse_serial_port_arg(argc - 1, argv);
203 }
204
205 int main(int argc, char **argv)
206 {
207     int port_index = -1;
208     int is_transmitter = 0;
209
210     if ((port_index = parse_args(argc, argv, &is_transmitter))
211         < 0) {
212         help(argv);
213         exit(EXIT_FAILURE);
214     }
215
216     if (is_transmitter) {
217         fprintf(stderr, "transmitting %s\n", file_to_send.name);
218         return send_file(argv[port_index], &file_to_send,
219             max_retries);
220     } else {
221         fprintf(stderr, "receiving file\n");
222 #ifdef NETLINK_DEBUG_MODE
223         fprintf(stderr, "\tserial_port_baudrate:%d\n",
224             serial_port_baudrate);
225         fprintf(stderr, "\tis_transmitter:%d\n", is_transmitter)
226             ;
227 #endif

```



```
224     return receive_file(argv[port_index], max_retries);
225 }
226 }
```