

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Počítačové komunikace a sítě

Dokumentace 2. projektu - sniffer paketů

Obsah

1	Úvod	2
2	Implementace	2
2.1	Funkce main	3
2.2	Funkce callback	3
2.3	Funkce print_IPv6	4
2.4	Funkce print_IPv4	4
2.5	Funkce print_devices()	4
2.6	Funkce print_data()	4
3	Testování	5
3.1	Přeložení	5
3.2	Příklady spuštění a referenční zařízení	5
4	Závěr	9
5	Zdroje	10

1 Úvod

Cílem tohoto projektu bylo naprogramovat síťový analyzátor, zvaný **packet sniffer**. Sniffer na základně uživatelem zadaného filtru zachytává a vypisuje packety typu TCP, UDP, ICMP a ARP. Podporované verze packetů jsou IPv4 a IPv6.

Program za použití promiskuitního módu síťové karty čte datové pakety (zmíněny výše), používající TCP/IP protokol. Tento protokol se dělí se na čtyři části: aplikační, transportní, síťová vrstva a vrstva síťového rozhraní. My se ale budeme zabývat jen vybranými částmi z těchto vrstev. Na vrstvě síťového rozhraní se podíváme na Ethernet a jeho hlavičku, ze síťové vrstvy máme protokoly jako IP, ARP a ICMP, protokoly typu TCP a UDP jsou z transportní vrstvy a z aplikační vrstvy vybíráme packety na základě protokolu jejich transportní vrstvy. Postup zachycení packetů je popsán níže v sekci **Implementace**.

2 Implementace

Analyzátor je napsán v jazyce C za pomoci knihovny libpcap [6] a nachází se v souboru **sniffer.c**. Pro zpracování jednotlivých hlaviček protokolů jsou používány ostatní knihovny, zejména **netinet/...** knihovny.

Odchycený packet je vypisován na standardní výstup **stdout** ve formátu:

timestamp: ukazuje čas
src MAC: MAC adresa odesilatele
dst MAC: MAC adresa příjemce **frame length:** délka rámce s bytech
src IP: IP adresa odesilatele
dst IP: IP adresa příjemce
Porty se vypisují jen u UDP a TCP packetu.
src port: port odesilatele
dst port: port příjemce

Dále pro ARP rámec a ICMP protokol jsem se rozhodl vypsát následující údaje navíc:

opcode: typ ARP packetu - odpověď/požadavek
type, code: typ ICMP packetu

Dále se jen vypíše obsah packetu jak v ASCII podobě, tak i v hexadecimální podobě.

```
timestamp: 2022-03-11T10:30:33.268856576+0100
src MAC: 00:15:5d:39:9b:f0
dst MAC: 01:00:5e:7f:ff:fa
frame length: 215 bytes
src IP: 172.22.64.1
dst IP: 239.255.255.250
src port: 61969
dst port: 1900
```

Obrázek 1: Příklad výše zmíněného formátu.

Celý soubor je rozdělen do několika funkcí, popsaných níže.

2.1 Funkce main

Ve funkci main se nachází hlavní tělo programu. Na začátku najdu pomocí funkce `pcap_findalldevs()` všechna dostupná zařízení. Poté zpracuju uživatelem předané argumenty. Rozhodl jsem se nepoužít funkci `getopt_long()` a naimplementovat to jednoduchým cyklem.

Po zpracování argumentů můžu na jejich základě sestavit filtr pomocí funkce `strcat`, který použijeme na analýzu.

Dále už se jen připravíme na samotnou analýzu [2]. `pcap_open_live()` nám otevře zařízení pro zachytávání. `pcap_lookupnet()` nám ověří, zda zařízení podporuje ethernet hlavičky. Funkce `pcap_compile()` a `pcap_setfilter()` nám aplikují filtr a nakonec funkce `pcap_loop()` zachytává samotné packety.

```
//PCAP HANDLE SETUP
handle = pcap_open_live(arg, BUFSIZ, 1, 1000, errbuf);
if(handle == NULL){
    printf("%s\n", errbuf);
    return 1;
}

pcap_set_immediate_mode(handle, 1);
int link_type;

//support only devices with ethernet headers
if (pcap_datalink(handle) != DLT_EN10MB) {
    fprintf(stderr, "Device %s doesn't provide Ethernet headers - not supported\n", arg);
    return 1;
}

//check the chosen device
if(pcap_lookupnet(arg, &sin, &netmask, errbuf) < 0){
    printf("%s\n", pcap_geterr(handle));
    return 1;
}
//printf("%s\n", filter);
//apply the filter
if (pcap_compile(handle, &bpf, (char *) filter, 0, netmask)){
    printf("%s\n", pcap_geterr(handle));
    return 1;
}

if (pcap_setfilter(handle, &bpf) < 0) {
    printf("%s\n", pcap_geterr(handle));
    return 1;
}
//printf("opening %s interface\n", arg);

//start sniffing
pcap_loop(handle, num, callback, errbuf);

pcap_close(handle);
return 0;
}
```

Obrázek 2: Kód pro zachytávání packetů ve funkci main()

2.2 Funkce callback

Tahle funkce se vyskytuje ve volání funkce `pcap_loop()` a zajišťuje určitý postup, co se má stát když je nějaký packet zachycen [3]. Rozděluje se na část s deklarací všech hlavičkových struktur a na část s funkcí `switch()`.

Jak již bylo výše zmíněno, hlavičky jsou z knihoven `netinet` a je jich celkem 10. K určitým strukturám je potřeba přičíst „offset“ [1], abych přistupoval ke správným datům.

```
unsigned short iphdrlen = iph->ihl*4;
struct tcphdr *tcp=(struct tcphdr*)(buffer + iphdrlen + sizeof(struct ethhdr));
```

Obrázek 3: Příklad offsetu TCP hlavičky. K velikosti struktury ethernet hlavičky je přičtena proměnlivá velikost IP hlavičky.

Ještě před jakýmkoli vypisováním je spočítána časová známka [5]. Na tohle jsem použil funkce z knihovny `time.h`.

Funkce `switch()` na základě ethernet typu rozlišuje typy packetů IPv4, IPv6 a ARP. V případě IPv4 a IPv6 je nadále i porovnáván typ protokolu v IP hlavičce. Ve funkci `switch()` se vypisují IP adresy, popř. porty a další data, a samotný packet. Pro výpis packetu je dedikována funkce `print_data()` popsána níže. Opět ale je potřeba předávat data s „offsetem“.

2.3 Funkce `print_IPv6`

Tahle funkce se zaměřuje na výpis Internet protokol adresy verze 6. Za pomoci knihovny `arpa/inet.h` převedu [4] adresu uloženou v `ip6h` struktuře.

2.4 Funkce `print_IPv4`

Funkce má stejnou funkci jako `print_IPv6`. Pomocí knihovny `in.h` je adresa převedena na řetězec a vypsána.

2.5 Funkce `print_devices()`

Funkce vypíše argumentem předaný jednosměrně vázaný seznam, obsahující zařízení pro zachytávání.

2.6 Funkce `print_data()`

Tahle funkce vypíše obsah packetu [1] (ethernet hlavička, IP/ARP hlavička, protokolová hlavička a data) v hexadecimální a ASCII podobě. Skládá se z jednoho cyklu, který vypíše vždy 16 (pokud není méně) znaků v hexadecimální podobě a zároveň těchto 16 znaků i v ASCII podobě. Netisknutelné znaky jsou nahrazeny tečkou.

Na začátku každého řádku je také hexadecimálně vypsán offset již vypsáných bajtů.

```
if (i!=0 && i%16==0){
    for(j=i-16 ; j<i ; j++)
    {
        if(buffer[j]>=32 && buffer[j]<=128)
            printf("%c", (unsigned char)buffer[j]);
        else
            printf(".");
    }
    printf("\n");
    if (line<10){
        printf("0x00%d: ", line*10);
    }
    else if (line<100){
        printf("0x0%d: ", line*10);
    }
    else{
        printf("0x%d: ", line*10);
    }
    ++line;
}

printf("%02x ", (unsigned char) buffer[i]);
```

Obrázek 4: Kód pro výpis ASCII a hexadecimální (poslední řádek) podoby symbolů packetu.

3 Testování

Testování probíhalo na poskytnuté Virtual Machine. Operační systém je Ubuntu 20.04.

3.1 Přeložení

Pomocí příkazu `make` vytvoříme spustitelný soubor `ipk-sniffer`. `make-clean` vymaže všechny binární a spustitelné soubory.

```
student@student-vm:~/shared$ make
gcc -std=gnu99 -g -c -o sniffer.o sniffer.c
gcc -std=gnu99 -g sniffer.o -lpcap -o ipk-sniffer
```

3.2 Příklady spuštění a referenční zařízení

Níže jsou některé příklady spuštění a vypsání na přiložené virtuálce. Jako referenční zařízení pro kontrolu jsem zvolil Wireshark. Formát spuštění a jaké argumenty jsou povoleny je popsáno v souboru `README.md`.

```
timestamp: 2022-03-13T13:43:16.117834160+0100
src MAC: 08:00:27:ca:e4:d4
dst MAC: 52:54:00:12:35:02
frame length: 98 bytes
src IP: 10.0.2.15
dst IP: 216.58.201.78
type: 8
code: 0

0x0000: 52 54 00 12 35 02 08 00 27 ca e4 d4 08 00 45 00 RT..5...'.....E.
0x0010: 00 54 98 5b 40 00 40 01 f4 b5 0a 00 02 0f d8 3a .T.[@.@.....:
0x0020: c9 4e 08 00 30 9b 00 03 00 01 e4 e6 2d 62 00 00 .N..0.....-b..
0x0030: 00 00 f2 44 04 00 00 00 00 00 10 11 12 13 14 15 ...D.....
0x0040: 16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25 .....!#$%
0x0050: 26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 34 35 &'()*+,-./012345
0x0060: 36 37                                     67
```

Obrázek 5: Výpis ICMPv4 packetu vyvolaný pomocí příkazu `ping`.

	1	0.000000000	10.0.2.15	216.58.201.78	ICMP	98 Echo (ping) request	id=0x0
	2	0.042867948	216.58.201.78	10.0.2.15	ICMP	98 Echo (ping) reply	id=0x0
	3	1.001788000	10.0.2.15	216.58.201.78	ICMP	98 Echo (ping) request	id=0x0
	4	1.039161789	216.58.201.78	10.0.2.15	ICMP	98 Echo (ping) reply	id=0x0

Frame 1: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface enp0s3, id 0							
Ethernet II, Src: PcsCompu_ca:e4:d4 (08:00:27:ca:e4:d4), Dst: RealtekU_12:35:02 (52:54:00:12:35:02)							
Internet Protocol Version 4, Src: 10.0.2.15, Dst: 216.58.201.78							
Internet Control Message Protocol							

0000	52	54	00	12	35	02	08	00	27	ca	e4	d4	08	00	45	00	RT..5...'.....E.
0010	00	54	98	5b	40	00	40	01	f4	b5	0a	00	02	0f	d8	3a	.T.[@.@.....:
0020	c9	4e	08	00	30	9b	00	03	00	01	e4	e6	2d	62	00	00	.N..0.....-b..
0030	00	00	f2	44	04	00	00	00	00	10	11	12	13	14	15		...D.....
0040	16	17	18	19	1a	1b	1c	1d	1e	1f	20	21	22	23	24	25!#\$%
0050	26	27	28	29	2a	2b	2c	2d	2e	2f	30	31	32	33	34	35	&'()*+,-./012345
0060	36	37															67

Obrázek 6: Ten samý packet zachycený wiresharkem.

```
timestamp: 2022-03-11T14:19:10.1824038176+0100
src MAC: 08:00:27:ca:e4:d4
dst MAC: 52:54:00:12:35:02
frame length: 74 bytes
src IP: 10.0.2.15
dst IP: 35.224.170.84
src port: 51340
dst port: 80

0x0000: 52 54 00 12 35 02 08 00 27 ca e4 d4 08 00 45 00 RT..5...'.....E.
0x0010: 00 3c 1d e1 40 00 40 06 42 98 0a 00 02 0f 23 e0 .<..@.@.B.....#.
0x0020: aa 54 c8 8c 00 50 89 4d 3b 0d 00 00 00 00 a0 02 .T...P.M;.....
0x0030: fa f0 da 71 00 00 02 04 05 b4 04 02 08 0a 43 73 ...q.....Cs
0x0040: a3 75 00 00 00 00 01 03 03 07 .u.....
```

Obrázek 7: Výpis IPv4 TCP packetu vyvolaný pomocí příkazu ping.

The image shows a Wireshark packet capture of a SYN packet. The packet list pane shows five packets, with packet 105 selected. The packet details pane shows the following information:

- Frame 105: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface enp0s3, id 0
- Ethernet II, Src: PcsCompu_ca:e4:d4 (08:00:27:ca:e4:d4), Dst: RealtekU_12:35:02 (52:54:00:12:35:02)
- Internet Protocol Version 4, Src: 10.0.2.15, Dst: 35.224.170.84
- Transmission Control Protocol, Src Port: 51340, Dst Port: 80, Seq: 0, Len: 0

The packet bytes pane shows the following data:

```
0000 52 54 00 12 35 02 08 00 27 ca e4 d4 08 00 45 00 RT..5...'.....E.
0010 00 3c 1d e1 40 00 40 06 42 98 0a 00 02 0f 23 e0 .<..@.@.B.....#.
0020 aa 54 c8 8c 00 50 89 4d 3b 0d 00 00 00 00 a0 02 .T...P.M;.....
0030 fa f0 da 71 00 00 02 04 05 b4 04 02 08 0a 43 73 ...q.....Cs
0040 a3 75 00 00 00 00 01 03 03 07 .u.....
```

Obrázek 8: Ten samý packet zachycený wiresharkem.

```

timestamp: 2022-03-11T14:16:53.1824038176+0100
src MAC: 08:00:27:ca:e4:d4
dst MAC: 52:54:00:12:35:02
frame length: 42 bytes
src IP: 10.0.2.15
dst IP: 10.0.2.2
opcode: 1

0x0000: 52 54 00 12 35 02 08 00 27 ca e4 d4 08 06 00 01 RT..5...'.
0x0010: 08 00 06 04 00 01 08 00 27 ca e4 d4 0a 00 02 0f .....'.
0x0020: 00 00 00 00 00 00 0a 00 02 02 .....

```

Obrázek 9: Výpis ARP packetu vyvolaný pomocí příkazu ping.

No.	Time	Source	Destination	Protocol	Length	Info
44	21.428120905	142.251.36.142	10.0.2.15	ICMP	98	Echo (ping) reply id=0
45	22.409801358	10.0.2.15	142.251.36.142	ICMP	98	Echo (ping) request id=0
46	22.429961795	142.251.36.142	10.0.2.15	ICMP	98	Echo (ping) reply id=0
47	22.505613926	PcsCompu_ca:e4:d4	RealtekU_12:35:02	ARP	42	who has 10.0.2.2? Tell 10
48	22.506150337	RealtekU_12:35:02	PcsCompu_ca:e4:d4	ARP	60	10.0.2.2 is at 52:54:00:1
49	23.411052431	10.0.2.15	142.251.36.142	ICMP	98	Echo (ping) request id=0
50	23.433951728	142.251.36.142	10.0.2.15	ICMP	98	Echo (ping) reply id=0
51	24.413174813	10.0.2.15	142.251.36.142	ICMP	98	Echo (ping) request id=0
52	24.438027942	142.251.36.142	10.0.2.15	ICMP	98	Echo (ping) reply id=0
53	25.417216758	10.0.2.15	142.251.36.142	ICMP	98	Echo (ping) request id=0

▶ Frame 47: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface enp0s3, id 0
 ▶ Ethernet II, Src: PcsCompu_ca:e4:d4 (08:00:27:ca:e4:d4), Dst: RealtekU_12:35:02 (52:54:00:12:35:02)
 ▶ Address Resolution Protocol (request)

0000	52 54 00 12 35 02 08 00 27 ca e4 d4 08 06 00 01	RT..5...'.
0010	08 00 06 04 00 01 08 00 27 ca e4 d4 0a 00 02 0f'.
0020	00 00 00 00 00 00 0a 00 02 02

Obrázek 10: Ten samý packet zachycený wiresharkem.


```

timestamp: 2022-03-11T13:59:48.3939156336+0100
src MAC: 08:00:27:ca:e4:d4
dst MAC: 52:54:00:12:35:02
frame length: 70 bytes
src IP: 10.0.2.15
dst IP: 192.168.0.1
src port: 36779
dst port: 53

0x0000: 52 54 00 12 35 02 08 00 27 ca e4 d4 08 00 45 00 RT..5...'.....E.
0x0010: 00 38 67 34 40 00 40 11 06 c9 0a 00 02 0f c0 a8 .8g4@.@.....
0x0020: 00 01 8f ab 00 35 00 24 cc ed 1d cc 01 00 00 01 .....5.$.....
0x0030: 00 00 00 00 00 00 06 67 6f 6f 67 6c 65 03 63 6f .....google.co
0x0040: 6d 00 00 01 00 01 m.....

```

Obrázek 11: Výpis IPv4 UDP packetu vyvolaný pomocí příkazu ping.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.0.2.15	192.168.0.1	DNS	70	Standard query 0x1dcc
2	0.000208367	10.0.2.15	192.168.0.1	DNS	70	Standard query 0x9c07
3	0.040399115	192.168.0.1	10.0.2.15	DNS	234	Standard query response
4	0.052111256	192.168.0.1	10.0.2.15	DNS	222	Standard query response
5	0.053110613	10.0.2.15	142.251.36.142	ICMP	98	Echo (ping) request
6	0.075889333	142.251.36.142	10.0.2.15	ICMP	98	Echo (ping) reply
7	1.054594084	10.0.2.15	142.251.36.142	ICMP	98	Echo (ping) request
8	1.071678960	142.251.36.142	10.0.2.15	ICMP	98	Echo (ping) reply
9	2.055564340	10.0.2.15	142.251.36.142	ICMP	98	Echo (ping) request
10	2.075035006	142.251.36.142	10.0.2.15	ICMP	98	Echo (ping) reply

▶ Frame 1: 70 bytes on wire (560 bits), 70 bytes captured (560 bits) on interface enp0s3, id 0
 ▶ Ethernet II, Src: PcsCompu_ca:e4:d4 (08:00:27:ca:e4:d4), Dst: RealtekU_12:35:02 (52:54:00:12:35:02)
 ▶ Internet Protocol Version 4, Src: 10.0.2.15, Dst: 192.168.0.1
 ▶ User Datagram Protocol, Src Port: 36779, Dst Port: 53
 ▶ Domain Name System (query)

```

0000  52 54 00 12 35 02 08 00 27 ca e4 d4 08 00 45 00 RT..5...'.....E.
0010  00 38 67 34 40 00 40 11 06 c9 0a 00 02 0f c0 a8 .8g4@.@.....
0020  00 01 8f ab 00 35 00 24 cc ed 1d cc 01 00 00 01 .....5.$.....
0030  00 00 00 00 00 00 06 67 6f 6f 67 6c 65 03 63 6f .....google.co
0040  6d 00 00 01 00 01 m.....

```

Obrázek 12: Ten samý packet zachycený wiresharkem.

```

timestamp: 2022-03-14T13:58:42.3447689184+0100
src MAC: 00:15:5d:e3:32:51
dst MAC: 33:33:00:00:00:fb
frame length: 102 bytes
src IP: fe80::b7:e8b6:c1e7:6801
dst IP: ff02::fb
src port: 5353
dst port: 5353

0x0000: 33 33 00 00 00 fb 00 15 5d e3 32 51 86 dd 60 0f 33.....].2Q..`.
0x0010: 38 06 00 30 11 01 fe 80 00 00 00 00 00 00 00 b7 8..0...♦.....
0x0020: e8 b6 c1 e7 68 01 ff 02 00 00 00 00 00 00 00 00 ....h.....
0x0030: 00 00 00 00 00 fb 14 e9 14 e9 00 30 79 ce 00 00 .....0y...
0x0040: 00 00 00 01 00 00 00 00 00 00 00 0b 5f 67 6f 6f 7 ....._goog
0x0050: 6c 65 63 61 73 74 04 5f 74 63 70 05 6c 6f 63 61 lecast._tcp.loca
0x0060: 6c 00 00 0c 00 01 l.....

```

Obrázek 13: Příklad výpisu IPv6 UDP zachycený volným sniffováním

No.	Time	Source	Destination	Protocol	Length	Info
16	17.613723	172.25.128.1	224.0.0.251	MDNS	82	Standard query 0x0000 PTR _googlecast._tcp.
17	17.614135	fe80::b7:e8b6:c1e7:...	ff02::fb	MDNS	102	Standard query 0x0000 PTR _googlecast._tcp.
18	17.614143	fe80::b7:e8b6:c1e7:...	ff02::fb	MDNS	102	Standard query 0x0000 PTR _googlecast._tcp.
19	17.614908	172.25.128.1	224.0.0.251	MDNS	82	Standard query 0x0000 PTR _googlecast._tcp.
20	17.614910	172.25.128.1	224.0.0.251	MDNS	82	Standard query 0x0000 PTR _googlecast._tcp.

>	Frame 17: 102 bytes on wire (816 bits), 102 bytes captured (816 bits) on interface \Device\NPF_{BA4BE701-15BA-4725-95C3-EDAB7}
>	Ethernet II, Src: Microsof_e3:32:51 (00:15:5d:e3:32:51), Dst: IPv6mcast_fb (33:33:00:00:00:fb)
>	Internet Protocol Version 6, Src: fe80::b7:e8b6:c1e7:6801, Dst: ff02::fb
>	User Datagram Protocol, Src Port: 5353, Dst Port: 5353
>	Multicast Domain Name System (query)

0000	33 33 00 00 00 fb 00 15 5d e3 32 51 86 dd 60 0f 33.....].2Q..`.
0010	38 06 00 30 11 01 fe 80 00 00 00 00 00 00 00 b7 8..0...♦.....
0020	e8 b6 c1 e7 68 01 ff 02 00 00 00 00 00 00 00 00h.....
0030	00 00 00 00 00 fb 14 e9 14 e9 00 30 79 ce 00 000y...
0040	00 00 00 01 00 00 00 00 00 00 0b 5f 67 6f 6f 67_goog
0050	6c 65 63 61 73 74 04 5f 74 63 70 05 6c 6f 63 61 lecast._tcp.loca
0060	6c 00 00 0c 00 01 l.....

Obrázek 14: Ten samý packet zachycený Wiresharkem.

Jak si můžeme všimnout, mnou výše vypsáné packety jsou shodné s výstupem Wiresharku.

4 Závěr

Projekt mě velice bavil, co se týče programové ale i teoretické části. Naučil jsem se jak vypadají různé hlavičky transportní, linkové a síťové vrstvy a poznal jsem nové knihovny v C.

5 Zdroje

Reference

- [1] BinaryTides: *How to code a Packet Sniffer in C with Libpcap on Linux*. [online], [viděno 4.3.2022].
URL <<https://www.binarytides.com/packet-sniffer-code-c-libpcap-linux-sockets/>>
- [2] Carstens, T.: *Programming with pcap*. [online], [viděno 4.3.2022].
URL <<https://www.tcpdump.org/pcap.html>>
- [3] IBM: *pcap_loop Subroutine*. [online], [viděno 4.3.2022].
URL <<https://www.ibm.com/docs/en/aix/7.2?topic=p-pcap-loop-subroutine>>
- [4] linuxhint: *inet_ntop function example*. [online], [viděno 6.3.2022].
URL <<https://linuxhint.com/c-init-ntop-function/>>
- [5] StackOverflow: *Using strftime in C, how can I format time exactly like a Unix timestamp?* [online], [viděno 8.3.2022].
URL <<https://stackoverflow.com/questions/1551597/using-strftime-in-c-how-can-i-format-time-exactly-like-a-unix-timestamp>>
- [6] TcpDump: *Man page of pcap*. [online], [viděno 3.3.2022].
URL <<https://www.tcpdump.org/manpages/pcap.3pcap.html>>