

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ



POČÍTAČOVÉ KOMUNIKACE A SÍTĚ
IPK

Sniffer paketů

Autor:
Adam KAŇKOVSKÝ

April 24, 2022

Obsah

.....	1
1. Úvod	3
1.1. Použití.....	3
2. Použité knihovny	4
3. Implementace.....	5
3.1. Funkce main()	5
3.2. Funkce printAllDevs()	5
3.3. Funkce socket_open()	5
3.4. Funkce getPacket()	5
3.5. Funkce packetParser()	5
3.6. Funkce print_ip_header().....	5
3.7. Funkce print_ip6_header().....	6
3.8. Funkce print_tcp_packet()	6
3.9. Funkce print_udp_packet()	6
3.10. Funkce packetPrinter()	6
4. Testování	7
4.1. ARP	7
4.2. TCP.....	7
4.3. UDP.....	8
4.4. ICMP	9
Bibliografie	10

1. Úvod

Zadáním projektu bylo navrhnout a naimplementovat síťový analyzátor, který bude schopný na určitém síťovém rozhraní zachytávat a filtrovat pakety.

Vyfiltrovat jsme mohli pakety těchto protokolů:

- TCP
- UDP
- ICMP
- ARP

1.1. Použití

`./ipk-sniffer [-i rozhraní | --interface rozhraní] {-p port} {[--tcp|-t] [--udp|-u] [--arp] [--icmp] } {-n num}`

Volby v ve složených závorkách {} znamená, že volba je nepovinná. Oproti tomu [] znamená povinnou volbu.

Volby:

-i *rozhraní* (právě jedno rozhraní, na kterém se bude poslouchat. Není-li uvedeno, nebo je bez hodnoty vypíšíou se všechna dostupná zařízení)

-p *číslo* (bude filtrovat pakety na daném rozhraní podle portu)

-t (pouze TCP pakety)

-u (pouze UDP pakety)

--icmp (pouze ICMP pakety)

--arp (pouze ARP rámce)

-n *číslo* (určuje počet packetů které se mají vytisknout)

2. Použité knihovny

Knihovna – o co se v projektu stará

<stdio.h> - práce se standardním vstupem a výstupem

<ctype.h> - funkce isnumber()

<getopt> - funkce getopt_long() pro rozsekávání argumentů programu

<string.h> - práce se stringy

<time.h> - práce s časem (formát času)

<stdlib.h> - základní pomocné funkce

<pcap.h> - knihovna na práci s rozhraními.

<netinet/ip.h> - struktura pro hlavičku ip

<netinet/ip6> - struktura pro hlavičku ip6

<netinet/tcp.h> - struktura pro hlavičku u protokolu tcp

<netinet/udp.h> - struktura pro hlavičku u protokolu udp

<netinet/if_ether.h> - struktura pro hlavičku ethernet

3. Implementace

Program je napsán v jazyce C. Hlavní a jediný zdrojový kód se nachází v souboru **main.c** doplněný hlavičkovým souborem **main.h**.

3.1. Funkce main()

Funkce **main()** je v tomto programu hlavní řídicí funkcí, ze které se v průběhu běhu programu volají pomocné funkce, které řeší problematiku zadání. Nejdříve funkce **main()** pomocí pomocné funkce **getopt_long()** z knihovny <getopt.h> a struktury v hlavičkovém souboru <main.h>, postupně vybírá argumenty programu, z proměnné **argv[]** a ukládá do proměnné **c** znak zkratky argumentu. (Free Software Foundation, Inc., 1993-2022) Tyto zkratky pomocí podmínek vyhodnocujeme. (Wik, 2021) K vyhodnocování argumentu **-i** se v některých případech využívá funkce **printAllDevs()**. Po vyhodnocení všech argumentů si nastavíme filtr snifferu. Poté zavoláme funkce **socket_open()** a **getPacket()** vysvětlené v dalším kroku. Nakonec už pouze ukončíme odposlouchávání a ukončíme program.

3.2. Funkce printAllDevs()

Zde je využívána funkce **pcap_findalldevs()** z knihovny <pcap.h>, která nám vyhledá všechna dostupná zařízení a vloží je to proměnné. Tato proměnná je pomocí jednoduchého cyklu tisknutá na standardní výstup. (Guru, 2014)

3.3. Funkce socket_open()

V této funkci jsou využívány pomocné funkce z knihovny <pcap.h>. Nejdříve otestujeme pomocí funkce **pcap_lookupnet()** jestli interface, který jsme si vybrali, je mezi dostupnými zařízeními a zjistíme si její masku a net adresu. Následně si otevřeme relaci pomocí **pcap_open_live()**. Další v pořadí je kompilace filtru (zajišťovaná funkcí **pcap_compile()** předaného z funkce **main()**. Potom už je třeba tento zkompilovaný filtr předat aktuální relaci. (Carstens, 2002)

3.4. Funkce getPacket()

Tato funkce je určena k tomu, aby packety které získá pomocí funkce **pcap_loop()**, předávala callback funkci **packetParser()**, která se už poté stará o jejich zpracování. (Carstens, 2002)

3.5. Funkce packetParser()

Funkce slouží k strukturalizaci jednotlivých částí packetu a zjištění požadovaných hodnot. Nejdříve si pomocí funkce **localtime()** z knihovny <time.h> převedeme čas z hlavičky packetu na místní čas a naformátujeme pomocí **strftime()**. (Monica, 2018). Následně pomocí struktury **ether_header** získáme source i destination MAC adresy. Poté už kroky musíme dělit podle protokolů se kterými budeme pracovat. Pomocí **eth->ether_type** získáme typ protokolu a poté už pouze voláme pomocné funkce pro dané protokoly. (Moon, 2020) Po rozdělení na IPv6, IPv4 a ARP, si u protokolů **ip** zjistíme podprotokoly, podle kterých voláme pomocné funkce, na výpis dat daného protokolu. Funkci končíme voláním funkce na výpis celého packetu **packetPrinter()**.

3.6. Funkce print_ip_header()

Tato funkce nám pomocí struktury **iphdr** z knihovny <netinet/ip.h> source a destination IP adresu ve formátu IPv4. (Moon, 2020)

3.7. Funkce `print_ip6_header()`

Tato funkce nám pomocí struktury `ip6_hdr` z knihovny `<netinet/ip6.h>` source a destination IP adresu ve formátu IPv6. (Moon, 2020)

3.8. Funkce `print_tcp_packet()`

Funkce je určená pro výpis portů ze struktury `tcphdr` z knihovny `<netinet/tcp.h>`. (Moon, 2020)

3.9. Funkce `print_udp_packet()`

Funkce je určená pro výpis portů ze struktury `udphdr` z knihovny `<netinet/udp.h>`. (Moon, 2020)

3.10. Funkce `packetPrinter()`

Poslední funkce je používána na tisknutí celého packetu, podobně jak je to třeba u aplikace Wireshark. První v prvním sloupečku tabulky tiskneme vždy popořadě offset řádku v hexadecimálním formátu `0x0000`. Následuje 16 hexadecimálních dvojic s daty. Poté tiskneme data pomocí tisknutelných znaků. Netisknutelné znaky nahrazujeme tečkou. Takto pokračujeme po řádcích až na konec packetu. (Moon, 2020)

4. Testování

Testování probíhalo na zařízení s operačním systémem ubuntu 22.04 LTS. Jako testovací variace byl použit program Wireshark.

4.1. ARP

```
(base) adam@adam-02:/media/adam/5E387D9B387D72BF/Dokumenty/vš/4.semestr/ipk/IPK_Projekt2$ sudo ./ipk-sniffer -i enp24s0 --arp -n 10
timestamp: 2022-04-24T21:52:55+02:00
src MAC: ff:ff:ff:ff:ff:ff
dst MAC: 00:d8:61:16:bb:7b
frame length: 42 bytes

0x0000: ff ff ff ff ff ff 00 d8 61 16 bb 7b 08 06 00 01 ..... a..{....
0x0010: 08 00 06 04 00 01 00 d8 61 16 bb 7b c0 a8 01 19 ..... a..{....

▶ Frame 1: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface e
▶ Ethernet II, Src: Micro-St_16:bb:7b (00:d8:61:16:bb:7b), Dst: Broadcast (ff:ff:ff
  ▶ Destination: Broadcast (ff:ff:ff:ff:ff:ff)
  ▶ Source: Micro-St_16:bb:7b (00:d8:61:16:bb:7b)
    Type: ARP (0x0806)
  ▶ Address Resolution Protocol (ARP Announcement)
```

Offset	Hex	ASCII
0000	ff ff ff ff ff ff 00 d8 61 16 bb 7b 08 06 00 01 a..{....
0010	08 00 06 04 00 01 00 d8 61 16 bb 7b c0 a8 01 19 a..{....
0020	00 00 00 00 00 00 c0 a8 01 19

4.2. TCP

```
(base) adam@adam-02:/media/adam/5E387D9B387D72BF/Dokumenty/vš/4.semestr/ipk/IPK_Projekt2$ sudo ./ipk-sniffer -i enp24s0 -p 5353 --udp -n 5
timestamp: 2022-04-24T22:02:40+02:00
src MAC: 00:d8:61:16:bb:7b
dst MAC: 01:00:5e:00:00:fb
frame length: 87 bytes
src IP: 192.168.1.25
dst IP: 224.0.0.251
src port: 5353
dst port: 5353

0x0000: 01 00 5e 00 00 fb 00 d8 61 16 bb 7b 08 00 45 00 ..^..... a..{..E.
0x0010: 00 49 5b b6 40 00 ff 11 7d 30 c0 a8 01 19 e0 00 .I[.@... }0.....
0x0020: 00 fb 14 e9 14 e9 00 35 a3 03 00 00 00 00 01 .....5 .....
0x0030: 00 00 00 00 00 00 10 5f 73 70 6f 74 69 66 79 2d ....._spotify-
```

▶ Frame 1: 87 bytes on wire (696 bits), 87 bytes captured (696 bits) on interface e
 ▶ Ethernet II, Src: Micro-St_16:bb:7b (00:d8:61:16:bb:7b), Dst: IPv4mcast_fb (01:00
 ▶ Internet Protocol Version 4, Src: 192.168.1.25, Dst: 224.0.0.251
 ▶ User Datagram Protocol, Src Port: 5353, Dst Port: 5353
 ▶ Multicast Domain Name System (query)

0000	01 00 5e 00 00 fb 00 d8 61 16 bb 7b 08 00 45 00	..^..... a..{..E.
0010	00 49 5b b6 40 00 ff 11 7d 30 c0 a8 01 19 e0 00	.I[.@... }0.....
0020	00 fb 14 e9 14 e9 00 35 a3 03 00 00 00 00 00 015
0030	00 00 00 00 00 00 10 5f 73 70 6f 74 69 66 79 2d_ spotify-
0040	63 6f 6e 6e 65 63 74 04 5f 74 63 70 05 6c 6f 63	connect _tcp loc
0050	61 6c 00 00 0c 00 01	al.....

enp24s0: <live capture in progress> Packets: 2 · Displayed: 2 (100.0%) Profile: Default

4.3.UDP

```

timestamp: 2022-04-24T22:10:47+02:00
src MAC: 44:ff:ba:12:33:2c
dst MAC: 00:d8:61:16:bb:7b
frame length: 112 bytes
src IP: 37.157.6.234
dst IP: 192.168.1.25
src port: 443
dst port: 55418

0x0000:  00 d8 61 16 bb 7b 44 ff  ba 12 33 2c 08 00 45 00  ..a..{D.  ..3,..E.
0x0010:  00 62 be bc 40 00 32 06  9b 91 25 9d 06 ea c0 a8  .b..@.2.  ..%.....
0x0020:  01 19 01 bb d8 7a 08 82  61 6d 69 e8 42 bc 80 18  .....z.. ami.B...
  
```

▶ Frame 83: 112 bytes on wire (896 bits), 112 bytes captured (896 bits) on interfac
 ▶ Ethernet II, Src: zte_12:33:2c (44:ff:ba:12:33:2c), Dst: Micro-St_16:bb:7b (00:d8
 ▶ Internet Protocol Version 4, Src: 37.157.6.234, Dst: 192.168.1.25
 ▶ Transmission Control Protocol, Src Port: 443, Dst Port: 55418, Seq: 1, Ack: 1, Le
 ▶ Transport Layer Security

0000	00 d8 61 16 bb 7b 44 ff ba 12 33 2c 08 00 45 00	..a..{D. ..3,..E.
0010	00 62 be bc 40 00 32 06 9b 91 25 9d 06 ea c0 a8	.b..@.2. ..%.....
0020	01 19 01 bb d8 7a 08 82 61 6d 69 e8 42 bc 80 18z.. ami.B...
0030	00 11 79 cb 00 00 01 01 08 0a 62 25 1c fe 9a f2	..y..... .b%....
0040	13 e9 17 03 03 00 29 b5 46 ad 0e ad c6 85 ce 0b) . F.....
0050	99 94 c5 49 5e 6d 72 26 e9 dc c5 cb d8 1c f4 39	...I^mr&9
0060	fd cf 47 44 20 9d 6f c5 e2 4c 56 81 9f 46 68 f8	..GD .o. .LV..Fh.


4.4. ICMP

```
(base) adam@adam-02:/media/adam/5E387D9B387D72BF/Dokumenty/vš/4.semestr/ipk/IPK_Projekt2$ sudo ./ipk-sniffer -i any --icmp -n 5
timestamp: 2022-04-24T22:19:12+02:00
src MAC: 00:d8:61:16:bb:7b
dst MAC: 00:04:00:01:00:06
frame length: 100 bytes

0x0000: 00 04 00 01 00 06 00 d8 61 16 bb 7b 00 00 08 00 ..... a..{....
0x0010: 45 00 00 54 5d 28 40 00 40 01 68 76 c0 a8 01 19 E..T](@. @.hv....
0x0020: 8e fb 24 4e 08 00 b7 8a 00 01 00 01 c0 b0 65 62 ..$N.... .....eb
0x0030: 00 00 00 00 52 8d 09 00 00 00 00 00 10 11 12 13 ....R... .....
0x0040: 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 ..... !"#
0x0050: 24 25 26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 $%&'()*+ ,-./0123
```

- ▶ Frame 1: 100 bytes on wire (800 bits), 100 bytes captured (800 bits) on interface
- ▶ Linux cooked capture v1
- ▶ Internet Protocol Version 4, Src: 192.168.1.25, Dst: 142.251.36.78
- ▶ Internet Control Message Protocol

0000	00 04 00 01 00 06 00 d8 61 16 bb 7b 00 00 08 00 a..{....
0010	45 00 00 54 5d 28 40 00 40 01 68 76 c0 a8 01 19	E..T](@. @.hv....
0020	8e fb 24 4e 08 00 b7 8a 00 01 00 01 c0 b0 65 62	..\$N....eb
0030	00 00 00 00 52 8d 09 00 00 00 00 00 10 11 12 13R...
0040	14 15 16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 !"#
0050	24 25 26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 32 33	\$%&'()*+ ,-./0123
0060	34 35 36 37	4567

 Protocol (ip.proto), 1 byte Packets: 12 · Displayed: 12 (100.0%) Profile: Default

5. Reference

Bibliografie

Carstens, T. (2002). *tcpdump*. Načteno z pcap: <https://www.tcpdump.org/pcap.html>

Free Software Foundation, Inc. (1993-2022). *GNU*. Retrieved from Example of Parsing Long Options with getopt_long: https://www.gnu.org/software/libc/manual/html_node/Getopt-Long-Option-Example.html

Guru, E. (21. January 2014). *embeddedguruji*. Načteno z pcap_findalldevs example: <http://embeddedguruji.blogspot.com/2014/01/pcapfindalldevs-example.html>

Monica, R. (13. February 2018). *Stackoverflow*. Načteno z <https://stackoverflow.com/questions/48771851/im-trying-to-build-an-rfc3339-timestamp-in-c-how-do-i-get-the-timezone-offset>

Moon, S. (31. July 2020). *BinaryTides*. Načteno z How to code a Packet Sniffer in C with Libpcap on Linux: <https://www.binarytides.com/packet-sniffer-code-c-linux/>

Wik, L. E. (2021, August 13). *cfengine*. Retrieved from Optional arguments with getopt_long(3): <https://cfengine.com/blog/2021/optional-arguments-with-getopt-long/>