



# Projekt 2 - Varianta ZETA

Sniffer paketů

---

Adam Bazel

xbazel00

# Obsah

---

Úvod .....	3
Základní informace .....	4
Zpracování filtračního výrazu .....	4
Získávání paket .....	4
Zpracování dat paket.....	4
Neimplementováno.....	4
Testování .....	5
Internetové zdroje .....	6
Obrázky.....	7

# Úvod

---

Jedná se o program, který zachytává pakety. Uživatel může pomocí parametrů určit jaké typy paketů se budou zobrazovat. Veškerá potřebná data o paketu podle zadání se vypisují na standardní výstup.

# Implementace

---

## Základní informace

Řešení sniferu paketů je implementováno v jazyce C. Kromě standardních knihoven jazyka C, byly použity knihovny a hlavičkové soubory `getopt`, `time`, `pcap`, `netinet`, `arpa` a `net`.

Knihovna `getopt` slouží k rozboru argumentů zadaných uživatelem [1]. Knihovna `time` poskytuje struktury pro vypsání času v požadovaném formátu. Přenosná knihovna `pcap` má funkce, které umožňují analyzování paketů a zachycení síťového provozu. Hlavičky rodiny `netinet` a `net` poskytly struktury hlaviček různých typů paketů a ethernetového rámce. Nakonec hlavička `arpa` poskytuje funkce, které umožní z paketu získat jeho zdrojovou a cílovou adresu.

## Zpracování filtračního výrazu

Po zpracování argumentů zadaných uživatelem v hlavním těle programu, následuje vytvoření filtračního výrazu v takovém formátu [2], aby funkce `pcap_compile()` jej dokázala zpracovat.

Celý výraz je vytvořen pomocí funkce `process_filter()` na základě zadaných argumentů do programu. Podle daných argumentů jsou příslušná klíčová slova zapsána do proměnné řetězce.

## Získávání paketů

Nejdříve se musí "otevřít" síťové zařízení, ze kterého chceme zachytávat pakety. Toto je umožněno funkcí `pcap_open_live()` [3][3]. Následně pomocí funkce `pcap_lookupnet()` získáme masku sítě, kterou dále použijeme s filtračním výrazem, zpracovaným ve funkci `process_filter()`, ve funkci `pcap_compile()` pro vytvoření filtračního programu [4]. Vytvořený filtrační program je následně využit funkcí `pcap_setfilter()`, kde je načten do zařízení zachytávající pakety. Dále se použije funkce `pcap_loop()`, která zachytává binární data paketu a ukončí svoji funkci po určitém množství zpracovaných paketů (zadáno uživatelem nebo výchozí hodnotou) [4].

## Zpracování dat paketů

Zpracování paketů má na starosti funkce s názvem `process_packet()`. Po zachycení paketu je tento načten do struktury `iphdr`, která data formátuje a již pouze přistupujeme k atributům této struktury [5]. Při příchodu paketu se tiskne časová stopa, kdy byl daný paket zachycen, na standardní výstup prostřednictvím funkce `print_time()` [6]. Následně se určí typ paketu podle protokolu [4]. Pokud je paket typu TCP, UDP nebo ICMP zjistí se, zda je IP adresa typu IPv4 nebo IPv6 [7]. Následně tiskne jeho příslušnou zdrojovou a cílovou adresu s porty s funkcemi `print_address()` nebo `print_6address()` [8]. Pokud se však jedná o paket, jehož protokol je typu ARP vypíše jeho ethernetovou zdrojovou a cílovou adresu. Poté se tiskne délka bajtů paketu.

Jako poslední se tiskne celý datový obsah paketu. Toto zaručuje funkce `print_data()` [5]. Data jsou tisknuta po bajtech v hexadecimální a ASCII podobě. Řádky obsahují 16 bajtů dat a jsou doprovázeny označením offsetu vypsaných bajtů.

## Neimplementováno

V projektu nebyla implementována podpora paketu internetového protokolu verze 6 (IPv6).

# Testování

Veškeré testování proběhlo na referenčním stroji PDS-VM uvedené v zadání. Výstupní výsledky byly porovnávány s výsledky generovanými programem Wireshark. Příklady uvedené zde:

```
student@student-vm:~/Documents/IPK$ sudo ./packet_sniffer -i enp0s3 -p 80 --tcp
2021-04-25T03:29:14.800+02:00 10.0.2.15 : 52402 > 34.107.221.82 : 80, length 74 bytes
0x0000 52 54 00 12 35 02 08 00 27 7e 8b 8e 08 00 45 00 RT..5...'~...E.
0x0010 00 3c 57 bb 40 00 40 06 d7 34 0a 00 02 0f 22 6b .<W.@.@.4...."k
0x0020 dd 52 cc b2 00 50 71 db 11 6a 00 00 00 00 a0 02 .R...Pq..j.....
0x0030 fa f0 0b fb 00 00 02 04 05 b4 04 02 08 0a 6b f8 .....k.
0x0040 49 cc 00 00 00 00 01 03 03 07 I.....
```

tcp.port == 80

No.	Time	Source	Destination	Protocol	Len
5	0.082128549	10.0.2.15	34.107.221.82	TCP	
6	0.108115817	34.107.221.82	10.0.2.15	TCP	

Frame 5: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface enp0s3, id 0  
Ethernet II, Src: PcsCompu\_7e:8b:8e (08:00:27:7e:8b:8e), Dst: RealtekU\_12:35:02 (08:00:02:08:00:12:35:02)  
Internet Protocol Version 4, Src: 10.0.2.15, Dst: 34.107.221.82  
Transmission Control Protocol, Src Port: 52402, Dst Port: 80, Seq: 0, Len: 0

```
0000 52 54 00 12 35 02 08 00 27 7e 8b 8e 08 00 45 00 RT..5...'~...E.
0010 00 3c 57 bb 40 00 40 06 d7 34 0a 00 02 0f 22 6b .<W.@.@.4...."k
0020 dd 52 cc b2 00 50 71 db 11 6a 00 00 00 00 a0 02 .R...Pq..j.....
0030 fa f0 0b fb 00 00 02 04 05 b4 04 02 08 0a 6b f8 .....k.
0040 49 cc 00 00 00 00 01 03 03 07 I.....
```

Obrázek 1: Porovnání výsledků vyhledání UDP paket s určitým portem

```
student@student-vm:~/Documents/IPK$ sudo ./packet_sniffer -i enp0s3 --arp
2021-04-25T03:23:06.288+02:00 ff:ff:ff:ff:ff:ff > 08:00:27:7e:8b:8e, length 42 bytes
0x0000 ff ff ff ff ff ff 08 00 27 7e 8b 8e 08 06 00 01 .....~.....
0x0010 08 00 06 04 00 01 08 00 27 7e 8b 8e 0a 00 02 0f .....~.....
0x0020 00 00 00 00 00 00 01 01 01 01 .....~.....
```

arp

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	PcsCompu_7e:8b:8e	Broadcast	ARP	42	Who has 1.1.1.1
2	1.000603273	PcsCompu_7e:8b:8e	Broadcast	ARP	42	Who has 1.1.1.1

Frame 1: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface enp0s3, id 0  
Ethernet II, Src: PcsCompu\_7e:8b:8e (08:00:27:7e:8b:8e), Dst: Broadcast (ff:ff:ff:ff:ff:ff)  
Address Resolution Protocol (request)

```
0000 ff ff ff ff ff ff 08 00 27 7e 8b 8e 08 06 00 01 .....~.....
0010 08 00 06 04 00 01 08 00 27 7e 8b 8e 0a 00 02 0f .....~.....
0020 00 00 00 00 00 00 01 01 01 01 .....~.....
```

Obrázek 2: Porovnání výsledků vyhledání ARP paket

# Internetové zdroje

---

- [1] *Getopt Long Option Example (The GNU C Library)* [online]. [cit. 25. 4. 2021]. Dostupné z: [https://www.gnu.org/software/libc/manual/html\\_node/Getopt-Long-Option-Example.html](https://www.gnu.org/software/libc/manual/html_node/Getopt-Long-Option-Example.html)
- [2] KEARY, Tim. tcpdump Cheat Sheet - Complete With Full Examples, *Comparitech* [online]. 31. 3 2021 [cit. 25. 4. 2021]. Dostupné z: <https://www.comparitech.com/net-admin/tcpdump-cheat-sheet/>
- [3] PEI, Liu Pei Pei. Isanotes/libpcap-tutorial, *GitHub* [online]. [cit. 24. 4. 2021]. Dostupné z: <https://github.com/Isanotes/libpcap-tutorial/blob/master/simplesniffer.c>
- [4] CARSTENS, Tim. Programming with pcap, *Programming with pcap TCPDUMP/LIBPCAP public repository* [online]. 2002 [cit. 25. 4. 2021]. Dostupné z: <https://www.tcpdump.org/pcap.html>
- [5] MOON, Silver. How to code a Packet Sniffer in C with Libpcap on Linux *BinaryTides* [online]. 31. 7. 2020 [accessed. 24. 4 2021]. Dostupné z: <https://www.binarytides.com/packet-sniffer-code-c-libpcap-linux-sockets/>
- [6] DENIS, Frank. rfc3339.c *Gist* [online]. [cit. 24. 4 2021]. Dostupné z: <https://gist.github.com/jedisct1/b7812ae9b4850e0053a21c922ed3e9dc>
- [7] BIRO, Ross, Fred N. KEMPEN, Donald BECKER, Alan COX, Arnt GULBRANDSEN, Richard UNDERWOOD, Stefan BECKER and Jorge CWIK. */net/ipv4/ip\_input.c* [online]. 2020 [cit. 25. 4. 2021]. Dostupné z: [https://students.mimuw.edu.pl/SO/Linux/Kod/net/ipv4/ip\\_input.c.html](https://students.mimuw.edu.pl/SO/Linux/Kod/net/ipv4/ip_input.c.html)
- [8] YUAN, Tony. yuan901202/vuw\_nwen302\_ethernet\_packet\_sniffer, *GitHub* [online]. 22. 8. 2015 [cit. 25. 4. 2021]. Dostupné z: [https://github.com/yuan901202/vuw\\_nwen302\\_ethernet\\_packet\\_sniffer/blob/master/eps.c](https://github.com/yuan901202/vuw_nwen302_ethernet_packet_sniffer/blob/master/eps.c)

# Obrázky

---

Obrázek 1: Porovnání výsledků vyhledání UDP paket s určitým portem

Obrázek 2: Porovnání výsledků vyhledání ARP paket