

# VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

## FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

### IPK Projekt 2

Varianta ZETA: Sniffer paketů

# Obsah

<b>1</b>	<b>Úvod</b>	<b>2</b>
<b>2</b>	<b>Možnosti spuštění programu</b>	<b>2</b>
<b>3</b>	<b>Návrh aplikace</b>	<b>2</b>
<b>4</b>	<b>Popis implementace</b>	<b>2</b>
<b>5</b>	<b>Testování</b>	<b>3</b>
5.1	Testy jednotlivých typů packetů na ipv4 a ipv6: . . . . .	3
5.1.1	TCP . . . . .	3
5.1.2	UDP . . . . .	4
5.1.3	ICMP . . . . .	5
5.2	Arp . . . . .	6

## 1 Úvod

Cílem packet snifferu je zachytávání packetů na určitém síťovém rozhraní. Zachytávané packety je možno filtrovat dle portu nebo protokolu. Také je možné specifikovat množství zachycených packetů. Hlavička a data packetu jsou vypisovány na standardní výstup.

## 2 Možnosti spuštění programu

Program je nejprve třeba přeložit příkazem `make`. Přeložený spustitelný soubor se nakopíruje do root složky projektu a je tedy možné ho spustit pomocí `./ipk-sniffer`. Povinným přepínačem je `--interface rozhraní` nebo jeho alias `-i rozhraní`. Pokud je zadáno `rozhraní`, je zahájeno odchyťávání packetů. Pokud tento argument není zadán, je vypsán seznam všech dostupných rozhraní. Dále lze pomocí přepínače `-n pocet_packetu` zvolit počet packetů, které se zobrazí. Pokud není přepínač použit, je zobrazen 1 packet. Zobrazené packety lze filtrovat pomocí přepínačů:

- `-p cislo_portu` - Filtruje packety dle čísla portu - může být jak port odesílatele tak příjemce
- `--tcp|-t --udp|-u --icmp --arp` - přepínače pro filtrování packetů používají dané protokoly

Nelze kombinovat přepínač `-p` a zároveň přepínače, které filtrují pouze `icmp` a `arp` (tedy např. `./ipk-sniffer -p 80 --icmp`)

## 3 Návrh aplikace

Konzolová aplikace obsahuje pouze 1 třídu `Program.cs`. Vstupním bodem programu je statická metoda `main`, která se stará pouze o zpracování argumentů a poté předává řízení metodě `SniffPackets`. `SniffPackets` nastaví filtry podle zvolených přepínačů, zachytává jednotlivé packety a pro zpracování dat packetů volá metodu `PrintPacket`. Tato metoda zjistí, o jaký typ packetu se jedná a parsuje ho do daného formátu. Poté sestaví řetězec v požadovaném formátu a vypíše ho na standardní výstup. Pro zformátování dat a hlavičky využívá pomocné metody `PacketDataHex`, `PacketHeader` a `FormatMac`.

## 4 Popis implementace

Zpracování argumentů je prováděno pomocí nuget balíčku `System.Commandline`[4]. Ten umožňuje vytvořit přepínače jako objekty a nastavit jim vlastnosti jako je popis, aliasy, počet povinných argumentů a podobně. Po zpracování argumentů je předáno řízení statické metodě `SniffPackets`.

Pro zachytávání packetů a jejich zpracování je využit nuget balíček `SharpPcap`[7]. Nejprve je ze seznamu rozhraní vybráno rozhraní zvolené uživatelem dle jména. Pokud není rozhraní zadáno, je vypsán seznam všech dostupných rozhraní a program se ukončí. Dalé je nastaven filtr packetů podle uživatelem zadných přepínačů. Tento filtr je ve formátu řetězce, který zpracovává knihovna `SharpPcap`. Pro samotné zachytávání packetů je v cyklu volána metoda `getNextPacket` a to tolikrát, kolik si zvolil uživatel přepínačem `-n`. Výchozí hodnotou je 1. Každý přijatý packet je převeden do datového typu `Packet` a předán pro výpis metodě `PrintPacket`.

Metoda `PrintPacket` využívá knihovnu `PacketDotNet`[6], která je součástí balíčku `SharpPcap`. Tato knihovna umožňuje zjistit, o jaký typ packetu se jedná a extrahovat jak data, tak hlavičku packetu do správného datového typu (`IPPacket`, `UdpPacket` apod.). K hodnotám z hlavičky lze poté přistupovat pomocí vlastností objektu a sestavit z nich řetězec v požadovaném formátu. Pro každý packet je vypisován čas přijetí, zdrojová a cílová IP adresa, délka packetu a data ve formát hexa i ascii. Pro TCP a UDP packety je navíc vypsán zdrojový a cílový port. Pro arp rámec je namísto ip adresy vypsána zdrojová a cílová mac adresa nacházející se v ethernetovém rámci, která je upravena do formátu oddělených bytů dvojtečkou[2]. Datum je naformátováno

dle RFC3339[1]. Data jsou zformátovány metodou `PrintHex` a poté upraveny do požadovaného formátu pomocí `linq`. Pro formátování jsou implementovány pomocné metody `PacketHeader`, `PacketDataHex` `FormatMac`. Výsledný řetězec je vypsán na standardní výstup.

## 5 Testování

Testování bylo prováděno na referenční virtuálce. Packety byly zasílány pomocí příkazu `sendip[3]`, jehož přepínači lze zvolit typ zasílaného packetu. Pouze arp rámce byly zaslány pomocí příkazu `nping[5]`, protože `sendip` nepodporuje.

### 5.1 Testy jednotlivých typů packetů na ipv4 a ipv6:

#### 5.1.1 TCP

```
2021-04-21T14:41:54.407+00:00 127.0.0.1 : 0 > 172.217.23.196 : 0, length 58 bytes
0x0000  80 2a a8 0b 5e 68 08 00 27 7e 8b 8e 08 00 45 00  .*..^h.. '~....E.
0x0010  00 2c c6 91 00 00 ff 06 b1 9b 7f 00 00 01 ac d9  ,.....
0x0020  17 c4 00 00 00 00 4a 0f da 71 00 00 00 00 50 02  .....J. .q....P.
0x0030  ff ff 6e fc 00 00 64 61 74 61                      ..n...da ta
```

Obrázek 1: TCP ipv4 zachycený snifferem

108	28.642202100	127.0.0.1	172.217.23.196	TCP
Frame 108: 58 bytes on wire (464 bits), 58 bytes captured (464 bits) on				
0000	80 2a a8 0b 5e 68 08 00	27 7e 8b 8e 08 00 45 00	.*..^h.. '~....E.	
0010	00 2c c6 91 00 00 ff 06	b1 9b 7f 00 00 01 ac d9	,.....	
0020	17 c4 00 00 00 00 4a 0f	da 71 00 00 00 00 50 02	.....J. .q....P.	
0030	ff ff 6e fc 00 00 64 61	74 61	..n...da ta	

Obrázek 2: TCP ipv4 ve wiresharku

```
2021-04-21T14:47:43.677+00:00 ::1 : 0 > fe80::bca5:aa0:d747:71f9 : 0, length 79 bytes
0x0000  44 8a 5b cd c4 cd 08 00 27 7e 8b 8e 86 dd 60 00  D.[..... '~....`.
0x0010  00 00 00 19 06 20 00 00 00 00 00 00 00 00 00 00  .....
0x0020  00 00 00 00 00 01 fe 80 00 00 00 00 00 00 bc a5  .....
0x0030  0a a0 d7 47 71 f9 00 00 00 00 4f 91 ae 03 00 00  ...Gq... ..O....
0x0040  00 00 50 02 ff ff 92 9c 00 00 64 61 74 61 32    ..P..... ..data2
```

Obrázek 3: TCP ipv6 zachycený snifferem

111	24.434385487	:::1	fe80::bca5:aa0:d747...	TCP	79	0	-	0	[SYN]	Seq=0	Win=65535	Len=5
Frame 111: 79 bytes on wire (632 bits), 79 bytes captured (632 bits) on interface enp0s3, id 0												
0000	44 8a 5b cd c4 cd 08 00	27 7e 8b 8e 86 dd 60 00	D.[.....'~.....'									
0010	00 00 00 19 06 20 00 00	00 00 00 00 00 00 00 00	.....									
0020	00 00 00 00 00 01 fe 80	00 00 00 00 00 00 bc a5	.....									
0030	0a a0 d7 47 71 f9 00 00	00 00 4f 91 ae 03 00 00	...Gq... ..0.....									
0040	00 00 50 02 ff ff 92 9c	00 00 64 61 74 61 32	..P..... ..data2									

Obrázek 4: TCP ipv6 ve wiresharku

## 5.1.2 UDP

2021-04-21T14:40:19.047+00:00 127.0.0.1 : 0 > 172.217.23.196 : 0, length 46 bytes												
0x0000	80 2a a8 0b 5e 68 08 00	27 7e 8b 8e 08 00 45 00	.*...^h.. '~....E.									
0x0010	00 20 59 d3 00 00 ff 11	1e 5b 7f 00 00 01 ac d9	..Y..... .[.....									
0x0020	17 c4 00 00 00 00 0c	e3 74 64 61 74 61	..... .tdata									

Obrázek 5: UDP ipv4 zachycený snifferem

12	3.463456858	127.0.0.1	172.217.23.196	UDP
Frame 12: 46 bytes on wire (368 bits), 46 bytes captured (368 bits) on				
0000	80 2a a8 0b 5e 68 08 00	27 7e 8b 8e 08 00 45 00	.*...^h.. '~....E.	
0010	00 20 59 d3 00 00 ff 11	1e 5b 7f 00 00 01 ac d9	. Y..... .[.....	
0020	17 c4 00 00 00 00 0c	e3 74 64 61 74 61	..... .tdata	

Obrázek 6: UDP ipv4 ve wiresharku

2021-04-21T14:50:20.414+00:00 :::1 : 0 > fe80::bca5:aa0:d747:71f9 : 0, length 70 bytes												
0x0000	44 8a 5b cd c4 cd 08 00	27 7e 8b 8e 86 dd 60 00	D.[.....'~.....'									
0x0010	00 00 00 10 11 20 00 00	00 00 00 00 00 00 00 00	.....									
0x0020	00 00 00 00 00 01 fe 80	00 00 00 00 00 00 bc a5	.....									
0x0030	0a a0 d7 47 71 f9 00 00	00 00 00 10 43 3e 64 61	...Gq... ....C>da									
0x0040	74 61 5f 75 64 70		ta_udp									

Obrázek 7: UDP ipv6 zachycený snifferem

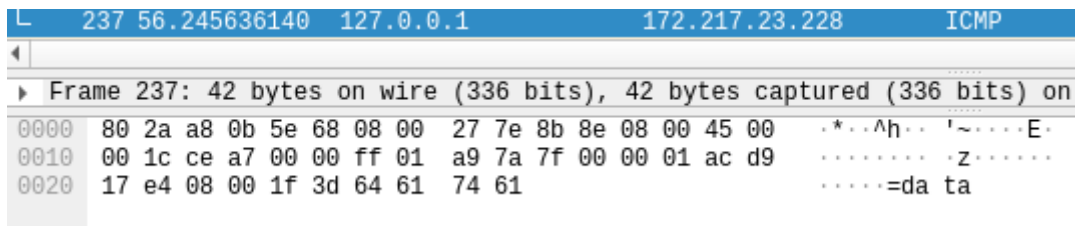
9 1.740642011 :::1																fe80::bca5:aa0:d747... UDP																													
Frame 9: 70 bytes on wire (560 bits), 70 bytes captured (560 bits) on int																																													
0000	44	8a	5b	cd	c4	cd	08	00	27	7e	8b	8e	86	dd	60	00	D.[.....'~.....'																												
0010	00	00	00	10	11	20	00	00	00	00	00	00	00	00	00	00	.....																												
0020	00	00	00	00	00	01	fe	80	00	00	00	00	00	00	bc	a5	.....																												
0030	0a	a0	d7	47	71	f9	00	00	00	00	00	10	43	3e	64	61	...Gq... ....C>da																												
0040	74	61	5f	75	64	70																										ta_udp													

Obrázek 8: UDP ipv6 ve wiresharku

### 5.1.3 ICMP

```
2021-04-21T14:38:10.948+00:00 127.0.0.1 > 172.217.23.228, length 42 bytes
0x0000  80 2a a8 0b 5e 68 08 00 27 7e 8b 8e 08 00 45 00  .*...^h.. '~....E.
0x0010  00 1c ce a7 00 00 ff 01 a9 7a 7f 00 00 01 ac d9  ....Z.....
0x0020  17 e4 08 00 1f 3d 64 61 74 61  ....=da ta
```

Obrázek 9: ICMPv4 zachycený snifferem



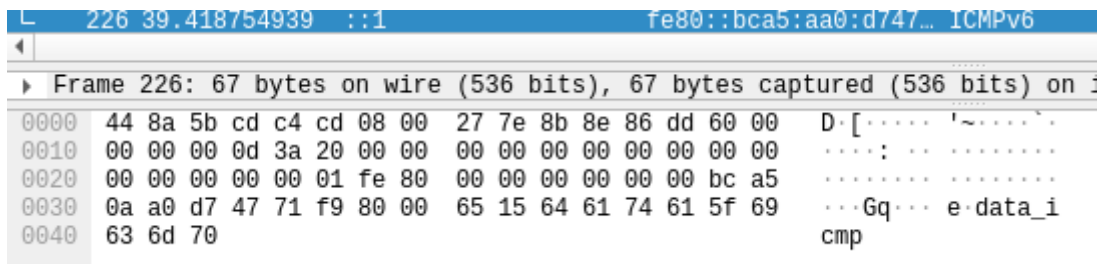
Wireshark interface showing a captured ICMPv4 packet. The packet list pane displays 'Frame 237: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on'. The packet details pane shows the ICMP header and data. The packet bytes pane shows the raw data in hexadecimal and ASCII.

Offset	Hex	ASCII
0000	80 2a a8 0b 5e 68 08 00 27 7e 8b 8e 08 00 45 00	. * . . ^ h . . ' ~ . . . . E .
0010	00 1c ce a7 00 00 ff 01 a9 7a 7f 00 00 01 ac d9	. . . . . . . . Z . . . . . .
0020	17 e4 08 00 1f 3d 64 61 74 61	. . . . . = d a t a

Obrázek 10: ICMPv4 ve wiresharku

```
2021-04-21T14:53:47.969+00:00 ::1 > fe80::bca5:aa0:d747:71f9, length 67 bytes
0x0000  44 8a 5b cd c4 cd 08 00 27 7e 8b 8e 86 dd 60 00  D.[..... '~....`.
0x0010  00 00 00 0d 3a 20 00 00 00 00 00 00 00 00 00 00  ....:..
0x0020  00 00 00 00 00 01 fe 80 00 00 00 00 00 00 bc a5  ....
0x0030  0a a0 d7 47 71 f9 80 00 65 15 64 61 74 61 5f 69  ...Gq... e.data_i
0x0040  63 6d 70  cmp
```

Obrázek 11: ICMPv6 zachycený snifferem



Wireshark interface showing a captured ICMPv6 packet. The packet list pane displays 'Frame 226: 67 bytes on wire (536 bits), 67 bytes captured (536 bits) on'. The packet details pane shows the ICMPv6 header and data. The packet bytes pane shows the raw data in hexadecimal and ASCII.

Offset	Hex	ASCII
0000	44 8a 5b cd c4 cd 08 00 27 7e 8b 8e 86 dd 60 00	D.[..... '~....`.
0010	00 00 00 0d 3a 20 00 00 00 00 00 00 00 00 00 00	....:..
0020	00 00 00 00 00 01 fe 80 00 00 00 00 00 00 bc a5	.....
0030	0a a0 d7 47 71 f9 80 00 65 15 64 61 74 61 5f 69	...Gq... e.data_i
0040	63 6d 70	cmp

Obrázek 12: ICMPv6 ve wiresharku

## 5.2 Arp

U arp rámců jsou místo zdrojové a cílové IP adresy vypisovány MAC adresy

```
2021-04-21T19:03:25.562+00:00 08:00:27:7e:8b:8e > ff:ff:ff:ff:ff:ff, 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 c0 a8 01 b6  ..... '~.....
0x0020  00 00 00 00 00 00 c0 a8 01 e3  ..... ..
```

Obrázek 13: ARP zachycený snifferem

23	4.841245411	PcsCompu_7e:8b:8e	Broadcast	ARP	42	Who has 192.168.1.227? Tell 192.168.1.182
Frame 23: 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)						
0000	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 c0 a8 01 b6  ..... '~.....
0020	00	00	00	00	00 00 c0 a8	01 e3  ..... ..

Obrázek 14: ARP ve wiresharku

## Reference

- [1] C# DateTime to RFC3339/ISO 8601. [online], [vid. 2020-04-22]. Dostupné z: <https://sebnilsson.com/blog/c-datetime-to-rfc3339-iso-8601/>
- [2] Mac Address format from string. [online], [vid. 2020-04-22]. Dostupné z: <https://stackoverflow.com/a/22372723>
- [3] sendip(1) - Linux man page. [online], [vid. 2020-04-22]. Dostupné z: <https://linux.die.net/man/1/sendip>
- [4] Dotnet: System.Commandline. [online], [vid. 2020-04-22]. Dostupné z: <https://github.com/dotnet/command-line-api>
- [5] MartinGarcia, L.: nping(1) — Linux manual page. [online], [vid. 2020-04-22]. Dostupné z: <https://man7.org/linux/man-pages/man1/nping.1.html>
- [6] Morgan, C.: Packet net. [online], [vid. 2020-04-22]. Dostupné z: <https://github.com/chmorgan/packetnet>
- [7] Morgan, C.: SharpPcap. [online], [vid. 2020-04-22]. Dostupné z: <https://github.com/chmorgan/sharppcap>