

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



Počítačové komunikácie a siete 2. projekt varianta Zeta – Sniffer paketov

Obsah

1	Úvod	2
2	Implementácia	2
2.1	Hlavné informácie	2
2.2	Spracovanie parametrov skriptu	2
2.3	Zobrazenie možných rozhraní	2
2.4	Nadviazanie spojenia s rozhraním	2
2.5	Príjmanie paketov	3
2.6	Spracovávanie paketov	3
2.6.1	Výpis času odchyteného paketu	3
2.6.2	Zistenie ethertypu	3
2.6.3	Spracovanie IPv4 paketov	3
2.6.4	Spracovanie ARP paketov	4
2.6.5	Spracovanie IPv6 paketov	4
2.6.6	Výpis dát paketu	4
2.7	Ukončenie skriptu	4
3	Testovanie skriptu	5
3.1	Testovanie IPv4 TCP	5
3.2	Testovanie IPv4 UDP	6
3.3	Testovanie IPv4 ICMP	7
3.4	Testovanie ARP	8
3.5	Testovanie IPv6 TCP	9
3.6	Testovanie IPv6 UDP	10
3.7	Testovanie ICMPv6	11
4	Záver	11

1 Úvod

Cieľom projektu bolo vytvoriť analyzátor paketov (sniffer paketov), ktorý odchyťava pakety podľa zadáných parametrov skriptu a vytvoriť k nemu upresňujúcu dokumentáciu. Analyzátor paketov slúži všeobecne k odchyťavaniu paketov, ktoré tečú z a do našej siete. Môžeme pomocou neho odhaliť chybné pakety, pomocou ktorých môžeme odhaliť chybné miesta a vďaka tomu udržať efektívny prenos dát[6].

2 Implementácia

2.1 Hlavné informácie

Skript je nazvaný `ipk-sniffer.cpp` a je spustiteľný ako `./ipk-sniffer` spolu s parametrami skriptu. Využíva knižnicu `libpcap`. V skripte sú zahrnuté hlavičkové súbory `pcap.h`, ktorá slúži pre volanie rôznych `pcap` funkcií viz. ďalšie sekcie, taktiež sú tam zahrnuté `netinet` hlavičkové súbory, ktoré sú použité pre prácu s rôznymi dátovými štruktúrami a ešte napríklad hlavičkový súbor `getopt` pre využitie funkcie `getopt_long`. Skript má taktiež niekoľko globálnych premenných do ktorých sa ukladajú hodnoty zo zadáných parametrov skriptu.

2.2 Spracovanie parametrov skriptu

Ako prvá funkcia, ktorá je volaná z `main` funkcie je funkcia `arg_parse`, ktorej úloha je spracovanie všetkých parametrov skriptu. Na spracovanie parametrov bola použitá funkcia `getopt_long`[1], kvôli tomu, že niektoré parametre majú ako dlhú formu tak aj krátku, napr. `--interface` je možné zapísať ako `-i`. Pokiaľ spustíme skript len s jediným parametrom `-i` bez rozhrania, tak sa vypíše zoznam všetkých rozhraní s ktorými je možné pracovať. Pokiaľ spustíme skript bez parametra `-i` tak program končí s chybou: `./ipk-sniffer` alebo `./ipk-sniffer -tcp`. Pomocou parametra môžeme špecifikovať aké pakety chceme odchyťávať, skript podporuje TCP, UDP, ICMP, ICMPv6 a ARP pakety: `-t -u --icmp --arp` a taktiež je možné špecifikovať port na ktorom sa bude odpočúvať: `-p port`. Je možné špecifikovať aj koľko paketov chceme odchytiť: `-n číslo`, pokiaľ tento parameter nieje zadáný, odchyťavame 1 paket.

2.3 Zobrazenie možných rozhraní

Pokiaľ bol skript spustený ako: `./ipk-sniffer -i`, tak sa dostávame do tejto funkcie, ktorá slúži k výpisu všetkých rozhraní. K výpisu je použitá funkcia `pcap_findalldevs`, ktorá vracia zoznam aktívnych rozhraní, ktorý sa následne prechádza a vypisuje sa meno každého rozhrania[5]. Po výpise rozhraní sa program automaticky ukončí.

2.4 Nadviazanie spojenia s rozhraním

Druhá možnosť ako spustiť skript je, že zadáme parametru `--interface` ešte aj názov rozhrania na ktorom budeme odpočúvať. Príklad spustenia: `./ipk-sniffer -i eth0`. V tomto prípade ideme do funkcie `open_session`, kde ako prvé si vytvoríme textový reťazec, ktorý bude reprezentovať náš filter. Postupne prechádzame jednotlivé parametre skriptu, ktoré boli zadané podľa

toho vytvárame filter. Ak chceme zachytávať len TCP a UDP pakety na porte 854, tak filter bude vyzeráť nasledovne: `port 854 and (tcp or udp)`. Keď máme vytvorený filter tak sa prejde k nadviazaniu spojenia pomocou 4 funkcií. Ako prvá funkciu, ktorú použijeme je `pcap_lookupnet` aby sme si získali sieťovú masku pre naše rozhranie aby sme potom vedeli aplikovať náš filter. Ďalšia funkcia je `pcap_open_live`, ktorá nám vytvorí spojenie s našim rozhraním, pričom jej dávame parameter aby sa využíval promiskuitný mód. Keď máme vytvorené spojenie tak si skompilujeme náš vytvorený filter do požadovaného formátu: štruktúry `bpf_program` pomocou funkcie `pcp_compile`. Posledná použitá funkcia v tejto časti je funkcia `pcap_setfilter` pomocou ktorej aplikujeme filter z predchádzajúcej funkcie na naše rozhranie s aktívnym spojením[8].

2.5 Príjmanie paketov

Keď mám nadviazané spojenie s rozhraním tak pomocou funkcie `pcap_loop` začnem prijímať pakety. Stanovím jej, koľko paketov ma odchytiť podľa parametra skriptu a taktiež jej dám funkciu, ktorá sa bude volať, vždy keď `pcap_loop` odchyti paket aby daný paket spracovala.

2.6 Spracovávanie paketov

Vždy keď vyššie uvedená funkcia odchyti paket, tak sa volá funkcia `handle_packet`, ktorá sa postará o jeho spracovanie. Všetky ďalej spomenuté štruktúry sú čerpané z netinet hlavičkových súborov[2] a k nim príslušných dokumentácií.

2.6.1 Výpis času odchyteného paketu

Hneď po odchytení paketu sa volá pomocná funkcia `print_time`, ktorá vypíše čas, kedy bol paket odchytený. Čas je získavaný pomocou funkcií `time` a `localtime` a na formátovanie daného času bola použitá funkcia `strftime`. Milisekundy sú získané pomocou funkcie `gettimeofday`. Táto časť kódu bola inšpirovaná zdrojom [7].

2.6.2 Zistenie ethertypu

Po výpise času odchytenia paketu sa prejde na spracovanie samotného paketu. Ako prvé si odchytený paket pretypujem na štruktúru `ether_header` z ktorej si následovne zistím ethernet type paketu, teda či sa jedná o IPv4, IPv6 alebo ARP paket.

2.6.3 Spracovanie IPv4 paketov

Pokiaľ je odchytený paket typu IPv4, tak k pôvodnému paketu prirátam veľkosť štruktúry `ether_header` a pretypujem si to na štruktúru `iphdr` z ktorej si následne vyčítam zdrojovú a destináčnu IP adresu pomocou funkcie `inet_ntoa`[3] a taktiež si z nej zistím o aký protokol sa jedná: TCP, UDP, ICMP. Ďalej sa program vetví podľa typu protokolu. Pokiaľ sa jedná o TCP protokol tak opäť prichádza k pretypovaniu na štruktúru `tcphdr` ku ktorej sa znova dostanem tak, že ku pôvodnému paketu pričítam veľkosť ethernetovej hlavičky a veľkosť IP hlavičky (posuny v pamäti). Z TCP hlavičky si zistím zdrojový a destinačný port a následne vypisujem prvý riadok dát vo formáte: čas RFC3339 source IP source Port > dest IP dest port, veľkosť paketu, protokol. A hneď potom sa volá funkcia pre výpis dát paketu. Obdobne to je pre UDP protokol, kde využívam štruktúru `udphdr`. Pri ICMP je výpis tiež podobný až na to, že tam sú vynechané porty, vzhľadom na to, že ICMP paket ich neobsahuje.

2.6.4 Spracovanie ARP paketov

Pokiaľ je paket typu ARP, prebehne pretypovanie do štruktúry `ether_arp` z ktorej si znova vytiahnem IP adresu podobne ako v prechádzajúcom prípade a taktiež MAC adresu, ktorú vypisujem v hexadecimálnej forme po bajtoch. Pri ARP paketoch vypisujem: čas RFC3339 source IP (source MAC) > dest IP (dest MAC), veľkosť paketu, ARP a či sa jedná o request alebo reply. Následuje výpis dát paketu.

2.6.5 Spracovanie IPv6 paketov

Ak je odchytený paket typu IPv6, tak znova prebehne pretypovanie do štruktúry `ip6_hdr` z ktorej sa ako prvé zistia adresy pomocou funkcie `inet_ntop`[4]. Ďalej sa z tejto štruktúry určí typ protokolu podľa ktorého sa ďalej program vetví. Spracovanie protokolu je podobné ako pri IPv4 adresách. Protokol ICMPv6 má obdobné spracovanie ako ICMP.

2.6.6 Výpis dát paketu

Vždy keď je spracovaný daný protokol paketu alebo hlavička ARP paketu, tak sa volá funkcia `print_packet`, ktorá má na starosti vypísať obsah paketu. Pre výpis sú použité iba klasické `for` cykly s určitými podmienkami aby výpis dát bol rovnaký s výpisom, ktorý je možný vidieť vo Wire-sharku.

```
0x0000: d8 f2 ca 3f 89 b9 00 0c 42 cb 39 5c 08 00 45 00  ...?....B.9\..E.
0x0010: 00 a3 40 09 40 00 38 06 b5 15 a2 9f 85 ea c0 a8  ..@.@.8.....
0x0020: 64 04 01 bb bd f2 0d 6e 19 59 c3 33 aa e5 50 18  d.....n.Y.3..P.
0x0030: 00 59 1a e1 00 00 17 03 03 00 76 e4 4e 73 41 98  .Y.....v.NsA.
0x0040: fd ec 04 9e dd c6 48 fb 91 c7 9b 1c 7b 13 ba 14  .....H.....{...
0x0050: 44 20 9e ae 41 25 25 74 1f c3 76 3d 92 4c 62 7c  D ..A%%t..v=.Lb|
0x0060: dc 9c 2c bb 7f aa ae f7 27 7f c0 a5 7f 6e dc 34  ..,.....'....n.4
0x0070: 27 0e 25 98 bc ed 2f d2 5b bb 31 0b f4 d8 cc f4  '.%.../. [.1....
0x0080: 8b fa ab 61 b9 80 35 11 df 54 c0 6b f1 03 1d bf  ...a..5..T.k....
0x0090: 04 7a 7a 13 ba 69 13 66 29 49 37 c6 14 f9 cc 56  .zz..i.f)I7...V
0x00a0: 2c bc 9a bc 08 03 08 72 d7 df 03 84 7b ef 5c a7  ,.....r.....{.\.
0x00b0: 7f  .
```

Obrázek 1: Výpis paketu

2.7 Ukončenie skriptu

Pokiaľ prebehne všetko úspešne tak skript skončí s návratovou hodnotou 0. Pokiaľ sa v niektorej z častí programu vyskytne nejaká chyba, či už pri spracovaní parametrov alebo volaní funkcií z `pcap` knižnice, tak skript vypíše chybovú hlášku na štandardný chybový výstup a skončí s návratovou hodnotou 1.

3 Testovanie skriptu

Skript bol testovaný priebežne pri implementácií. Na testovanie bol použitý open-source software Wireshark, výstupy skriptu boli porovnávané priamo s výstupmi z daného softwaru.

3.1 Testovanie IPv4 TCP

```
plosnak2@plosnak2-Lenovo-ideapad-330-15ICH:~/Documents/Sniffer$ sudo ./ipk-sniffer -i wlp0s20f3 -t
2021-04-23T12:47:21.382+02:00 192.168.100.4 : 33092 > 34.120.186.93 : 443, length 105 bytes, TCP (IPv4)
0x0000: 00 0c 42 cb 39 5c d8 f2 ca 3f 89 b9 08 00 45 00 ..B.9\...?....E.
0x0010: 00 5b 42 c6 40 00 40 06 f6 54 c0 a8 64 04 22 78 .[B.@.@..T..d."x
0x0020: ba 5d 81 44 01 bb ec a9 7f 18 e8 d1 b9 80 80 18 .].D.....
0x0030: 0c eb 01 d0 00 00 01 01 08 0a e5 bf 7b a7 dd e2 .....{...
0x0040: 3f fa 17 03 03 00 22 b3 d6 b0 55 25 b1 a6 db 05 ?....."....U%....
0x0050: 84 67 95 50 ee 92 8b a7 35 41 87 fc 13 cd fe d0 .g.P....5A.....
0x0060: fa 43 22 02 f5 71 66 bb f0 .C"...qf..
```

Obrázek 2: IPv4 TCP paket - výstup skriptu

86	17.292021369	192.168.100.4	34.120.186.93	TLSv1.2	105 Application Data
87	17.292134544	192.168.100.4	34.107.195.226	TLSv1.2	105 Application Data
88	17.312292341	34.120.186.93	192.168.100.4	TCP	68 443 → 33092 [ACK] Seq=1 Ack=40 Win=1050
89	17.312608278	34.120.186.93	192.168.100.4	TLSv1.2	105 Application Data
90	17.312641181	34.107.195.226	192.168.100.4	TCP	68 443 → 53090 [ACK] Seq=1 Ack=40 Win=282
91	17.313290499	34.107.195.226	192.168.100.4	TLSv1.2	105 Application Data
92	17.356652045	192.168.100.4	34.107.195.226	TCP	66 53090 → 443 [ACK] Seq=40 Ack=40 Win=2080
93	17.356691080	192.168.100.4	34.120.186.93	TCP	66 33092 → 443 [ACK] Seq=40 Ack=40 Win=330

▶ Frame 86: 105 bytes on wire (840 bits), 105 bytes captured (840 bits) on interface wlp0s20f3, id 0

▶ Ethernet II, Src: IntelCor_3f:89:b9 (d8:f2:ca:3f:89:b9), Dst: Routerbo_cb:39:5c (00:0c:42:cb:39:5c)

▶ Internet Protocol Version 4, Src: 192.168.100.4, Dst: 34.120.186.93

▶ Transmission Control Protocol, Src Port: 33092, Dst Port: 443, Seq: 1, Ack: 1, Len: 39

▶ Transport Layer Security

0000	00 0c 42 cb 39 5c d8 f2 ca 3f 89 b9 08 00 45 00	..B.9\..?....E.
0010	00 5b 42 c6 40 00 40 06 f6 54 c0 a8 64 04 22 78	.[B.@.@..T..d."x
0020	ba 5d 81 44 01 bb ec a9 7f 18 e8 d1 b9 80 80 18	.].D.....
0030	0c eb 01 d0 00 00 01 01 08 0a e5 bf 7b a7 dd e2{...
0040	3f fa 17 03 03 00 22 b3 d6 b0 55 25 b1 a6 db 05	?....."....U%....
0050	84 67 95 50 ee 92 8b a7 35 41 87 fc 13 cd fe d0	.g.P....5A.....
0060	fa 43 22 02 f5 71 66 bb f0	.C"...qf..

Obrázek 3: IPv4 TCP paket - výstup Wiresharku

3.2 Testovanie IPv4 UDP

```
plosnak2@plosnak2-Lenovo-ideapad-330-15ICH:~/Documents/Sniffer$ sudo ./ipk-sniffer -i wlp0s20f3 -u
2021-04-23T12:59:34.822+02:00 192.168.100.1 : 5353 > 224.0.0.251 : 5353, length 154 bytes. UDP (IPv4)
0x0000: 01 00 5e 00 00 fb 6a af ad e8 05 28 08 00 45 00  ..^...j....(..E.
0x0010: 00 8c 97 54 00 00 ff 11 1e 67 c0 a8 64 01 e0 00  ...T.....g..d...
0x0020: 00 fb 14 e9 14 e9 00 78 77 4b 00 00 00 00 00 03  ....xwK.....
0x0030: 00 00 00 00 00 01 08 5f 68 6f 6d 65 6b 69 74 04  ...._homekit.
0x0040: 5f 74 63 70 05 6c 6f 63 61 6c 00 00 0c 80 01 0f  _tcp.local.....
0x0050: 5f 63 6f 6d 70 61 6e 69 6f 6e 2d 6c 69 6e 6b c0  _companion-link.
0x0060: 15 00 0c 80 01 0c 5f 73 6c 65 65 70 2d 70 72 6f  ....._sleep-pro
0x0070: 78 79 04 5f 75 64 70 c0 1a 00 0c 80 01 00 00 29  xy._udp.....)
0x0080: 05 a0 00 00 11 94 00 12 00 04 00 0e 00 cd ba 90  .....
0x0090: 47 27 1e 48 6a af ad e8 05 28  G'.Hj....(
```

Obrázek 4: IPv4 UDP paket - výstup skriptu

98...	751.196963333	192.168.100.1	224.0.0.251	MDNS	154 Standard query 0x0000 PTR _homekit._tcp
98...	751.196963711	fe80::ceb:fd95:feb0...	ff02::fb	MDNS	174 Standard query 0x0000 PTR _homekit._tcp
98...	752.016061077	192.168.100.1	224.0.0.251	MDNS	132 Standard query 0x0000 PTR _raop._tcp.loc
98...	752.016061526	fe80::ceb:fd95:feb0...	ff02::fb	MDNS	152 Standard query 0x0000 PTR _raop._tcp.loc
98...	752.118560539	192.168.100.1	224.0.0.251	MDNS	154 Standard query 0x0000 PTR _homekit._tcp
98...	752.118560869	fe80::ceb:fd95:feb0...	ff02::fb	MDNS	174 Standard query 0x0000 PTR _homekit._tcp

Frame 9853: 154 bytes on wire (1232 bits), 154 bytes captured (1232 bits) on interface wlp0s20f3, id 0			
Ethernet II, Src: 6a:af:ad:e8:05:28 (6a:af:ad:e8:05:28), Dst: IPv4mcast_fb (01:00:5e:00:00:fb)			
Internet Protocol Version 4, Src: 192.168.100.1, 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 6a af	ad e8 05 28 08 00 45 00	..^...j....(..E.
0010	00 8c 97 54 00 00 ff 11	1e 67 c0 a8 64 01 e0 00	...T.....g..d...
0020	00 fb 14 e9 14 e9 00 78	77 4b 00 00 00 00 00 03xwK.....
0030	00 00 00 00 00 01 08 5f	68 6f 6d 65 6b 69 74 04_homekit.
0040	5f 74 63 70 05 6c 6f 63	61 6c 00 00 0c 80 01 0f	_tcp.local.....
0050	5f 63 6f 6d 70 61 6e 69	6f 6e 2d 6c 69 6e 6b c0	_companion-link.
0060	15 00 0c 80 01 0c 5f 73	6c 65 65 70 2d 70 72 6f_sleep-pro
0070	78 79 04 5f 75 64 70 c0	1a 00 0c 80 01 00 00 29	xy._udp.....)
0080	05 a0 00 00 11 94 00 12	00 04 00 0e 00 cd ba 90
0090	47 27 1e 48 6a af ad e8	05 28	G'.Hj....(

Obrázek 5: IPv4 UDP paket - výstup Wiresharku

3.3 Testovanie IPv4 ICMP

```
plosnak2@plosnak2-Lenovo-ideapad-330-15ICH:~/Documents/Sniffer$ sudo ./ipk-sniffer -i wlp0s20f3 --icmp
2021-04-23T13:04:34.822+02:00 192.168.100.4 > 8.8.8.8, length 98 bytes ICMPv4
0x0000: 00 0c 42 cb 39 5c d8 f2 ca 3f 89 b9 08 00 45 00  ..B.9\...?....E.
0x0010: 00 54 b3 af 40 00 40 01 52 3d c0 a8 64 04 08 08  .T..@.@.R=..d...
0x0020: 08 08 08 00 26 24 00 01 00 01 c2 a9 82 60 00 00  ....&$.....`..
0x0030: 00 00 cc fc 01 00 00 00 00 00 10 11 12 13 14 15  .....
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 6: IPv4 ICMP paket - výstup skriptu

No.	Time	Source	Destination	Protocol	Length	Info
13...	1050.9247982...	192.168.100.4	8.8.8.8	ICMP	98	Echo (ping) request id=0x0001, seq=1/256
13...	1050.9418244...	8.8.8.8	192.168.100.4	ICMP	98	Echo (ping) reply id=0x0001, seq=1/256

Frame 13399: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface wlp0s20f3, id 0	
Ethernet II, Src: IntelCor_3f:89:b9 (d8:f2:ca:3f:89:b9), Dst: Routerbo_cb:39:5c (00:0c:42:cb:39:5c)	
Internet Protocol Version 4, Src: 192.168.100.4, Dst: 8.8.8.8	
Internet Control Message Protocol	

0000	00 0c 42 cb 39 5c d8 f2 ca 3f 89 b9 08 00 45 00	..B.9\...?....E.
0010	00 54 b3 af 40 00 40 01 52 3d c0 a8 64 04 08 08	.T..@.@.R=..d...
0020	08 08 08 00 26 24 00 01 00 01 c2 a9 82 60 00 00&\$.....`..
0030	00 00 cc fc 01 00 00 00 00 00 10 11 12 13 14 15
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 7: IPv4 ICMP paket - výstup skriptu

3.4 Testovanie ARP

```
plosnak2@plosnak2-Lenovo-ideapad-330-15ICH:~/Documents/Sniffer$ sudo ./ipk-sniffer -i wlp0s20f3 --arp
2021-04-23T13:10:49.318+02:00 192.168.100.252 (00:0c:42:cb:39:5c) > 192.168.100.4 (00:00:00:00:00:00), length 60 bytes, ARP packet (request)
0x0000: d8 f2 ca 3f 89 b9 00 0c 42 cb 39 5c 08 06 00 01 ...?....B.9\....
0x0010: 08 00 06 04 00 01 00 0c 42 cb 39 5c c0 a8 64 fc .....B.9\..d.
0x0020: 00 00 00 00 00 00 c0 a8 64 04 00 00 00 00 00 .....d.....
0x0030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....

```

Obrázek 8: ARP paket - výstup skriptu

17...	1425.9290207...	Routerbo_cb:39:5c	IntelCor_3f:89:b9	ARP	60 Who has 192.168.100.4? Tell 192.168.100.4
17...	1425.9290455...	IntelCor_3f:89:b9	Routerbo_cb:39:5c	ARP	42 192.168.100.4 is at d8:f2:ca:3f:89:b9
17...	1456.1388545...	Routerbo_cb:39:5c	IntelCor_3f:89:b9	ARP	60 Who has 192.168.100.4? Tell 192.168.100.4
17...	1456.1388654...	IntelCor_3f:89:b9	Routerbo_cb:39:5c	ARP	42 192.168.100.4 is at d8:f2:ca:3f:89:b9
17...	1487.0643641...	Routerbo_cb:39:5c	IntelCor_3f:89:b9	ARP	60 Who has 192.168.100.4? Tell 192.168.100.4

Hardware size: 6
Protocol size: 4
Opcode: request (1)
Sender MAC address: Routerbo_cb:39:5c (00:0c:42:cb:39:5c)
Sender IP address: 192.168.100.252
Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
Target IP address: 192.168.100.4

0000 d8 f2 ca 3f 89 b9 00 0c 42 cb 39 5c 08 06 00 01 ...?.... B.9\....
0010 08 00 06 04 00 01 00 0c 42 cb 39 5c c0 a8 64 fc B.9\..d.
0020 00 00 00 00 00 00 c0 a8 64 04 00 00 00 00 00 d.....
0030 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Obrázek 9: ARP paket - výstup Wiresharku

3.5 Testovanie IPv6 TCP

```
plosnak2@plosnak2-Lenovo-ideapad-330-15ICH:~/Documents/Sniffer$ sudo ./ipk-sniffer -i wlp0s20f3 --tcp -p 854
2021-04-23T13:18:20.262+02:00 fe80::80b6:4111:9979:d48c : 854 > ff02::fb : 5541, length 91 bytes, TCP (IPv6)
0x0000: 33 33 00 00 00 fb d8 f2 ca 3f 89 b9 86 dd 60 00 33.....?....
0x0010: 00 00 00 25 06 40 fe 80 00 00 00 00 00 80 b6 ...%.@.....
0x0020: 41 11 99 79 d4 8c ff 02 00 00 00 00 00 00 00 A..y.....
0x0030: 00 00 00 00 00 fb 03 56 15 a5 00 00 00 00 00 .....V.....
0x0040: 00 00 50 02 20 00 0a 80 00 00 54 45 53 54 20 2d ..P. ....TEST -
0x0050: 20 49 50 76 36 20 2b 20 54 43 50             IPv6 + TCP
```

Obrázek 10: IPv6 TCP paket na porte 854 - výstup skriptu

25...	1876.6864059...	fe80::80b6:4111:997...	ff02::fb	TCP	91 [TCP Retransmission] 854 → 5541 [SYN] S
25...	1877.1172113...	162.159.133.234	192.168.100.4	TLSv1.2	150 Application Data
25...	1877.1172379...	192.168.100.4	162.159.133.234	TCP	54 48626 → 443 [ACK] Seq=3190 Ack=458582 W
25...	1877.7933344...	192.168.100.4	162.159.133.234	TLSv1.2	108 Application Data
25...	1877.8288797...	162.159.133.234	192.168.100.4	TCP	60 443 → 48626 [ACK] Seq=458582 Ack=3244 W

▶ Frame 25274: 91 bytes on wire (728 bits), 91 bytes captured (728 bits) on interface wlp0s20f3, id 0					
▶ Ethernet II, Src: IntelCor_3f:89:b9 (d8:f2:ca:3f:89:b9), Dst: IPv6mcast_fb (33:33:00:00:00:fb)					
▶ Internet Protocol Version 6, Src: fe80::80b6:4111:9979:d48c, Dst: ff02::fb					
▶ Transmission Control Protocol, Src Port: 854, Dst Port: 5541, Seq: 0, Len: 17					

0000	33 33 00 00 00 fb d8 f2 ca 3f 89 b9 86 dd 60 00	33.....?....`
0010	00 00 00 25 06 40 fe 80 00 00 00 00 00 80 b6	..%.@..
0020	41 11 99 79 d4 8c ff 02 00 00 00 00 00 00 00	A..y.....
0030	00 00 00 00 00 fb 03 56 15 a5 00 00 00 00 00V.....
0040	00 00 50 02 20 00 0a 80 00 00 54 45 53 54 20 2d	..P.TEST -
0050	20 49 50 76 36 20 2b 20 54 43 50	IPv6 + TCP

Obrázek 11: IPv6 TCP paket na porte 854 - výstup Wiresharku

3.6 Testovanie IPv6 UDP

```
plosnak2@plosnak2-Lenovo-Ideapad-330-15ICH:~/Documents/Sniffer$ sudo ./ipk-sniffer -i wlp0s20f3 --udp -p 854
2021-04-23T13:25:38.438+02:00 fe80::80b6:4111:9979:d48c : 854 > ff02::fb : 5541, length 79 bytes, UDP (IPv6)
0x0000: 33 33 00 00 00 fb d8 f2 ca 3f 89 b9 86 dd 60 00 33.....?....`
0x0010: 00 00 00 19 11 40 fe 80 00 00 00 00 00 00 80 b6 .....@.....
0x0020: 41 11 99 79 d4 8c ff 02 00 00 00 00 00 00 00 00 A..y.....
0x0030: 00 00 00 00 00 fb 03 56 15 a5 00 19 79 69 54 45 .....V....yiTE
0x0040: 53 54 20 2d 20 49 50 76 36 20 2b 20 55 44 50 ST - IPv6 + UDP
```

Obrázek 12: IPv6 UDP paket na porte 854 - výstup skriptu

32...	2314.4232217...	fe80::80b6:4111:9979...	ff02::fb	UDP	79 854 → 5541 Len=17
32...	2316.3911277...	162.159.133.234	192.168.100.4	TLSv1.2	453 Application Data
32...	2316.3911559...	192.168.100.4	162.159.133.234	TCP	54 48626 → 443 [ACK] Seq=3784 Ack=548643 W...
32...	2316.3911281...	162.159.133.234	192.168.100.4	TLSv1.2	116 Application Data
32...	2316.3912029...	192.168.100.4	162.159.133.234	TCP	54 48626 → 443 [ACK] Seq=3784 Ack=548705 W...
32...	2316.3911282...	162.159.133.234	192.168.100.4	TLSv1.2	109 Application Data
32...	2316.3912145...	192.168.100.4	162.159.133.234	TCP	54 48626 → 443 [ACK] Seq=3784 Ack=548760 W...

▶ Frame 32351: 79 bytes on wire (632 bits), 79 bytes captured (632 bits) on interface wlp0s20f3, id 0	
▶ Ethernet II, Src: IntelCor_3f:89:b9 (d8:f2:ca:3f:89:b9), Dst: IPv6mcast_fb (33:33:00:00:00:fb)	
▶ Internet Protocol Version 6, Src: fe80::80b6:4111:9979:d48c, Dst: ff02::fb	
▶ User Datagram Protocol, Src Port: 854, Dst Port: 5541	
▶ Data (17 bytes)	

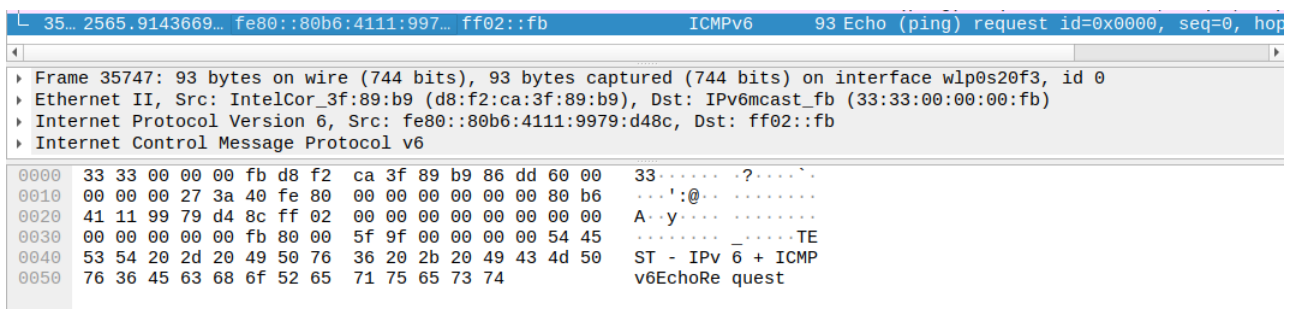
0000	33 33 00 00 00 fb d8 f2 ca 3f 89 b9 86 dd 60 00	33.....?....`
0010	00 00 00 19 11 40 fe 80 00 00 00 00 00 00 80 b6@.....
0020	41 11 99 79 d4 8c ff 02 00 00 00 00 00 00 00 00	A..y.....
0030	00 00 00 00 00 fb 03 56 15 a5 00 19 79 69 54 45V....yiTE
0040	53 54 20 2d 20 49 50 76 36 20 2b 20 55 44 50	ST - IPv 6 + UDP

Obrázek 13: IPv6 TCP paket na porte 854 - výstup Wiresharku

3.7 Testovanie ICMPv6

```
plosnak2@plosnak2-Lenovo-ideapad-330-15ICH:~/Documents/Sniffer$ sudo ./ipk-sniffer -i wlp0s20f3 --icmp
2021-04-23T13:29:49.734+02:00 fe80::80b6:4111:9979:d48c > ff02::fb, length 93 bytes, ICMPv6
0x0000: 33 33 00 00 00 fb d8 f2 ca 3f 89 b9 86 dd 60 00 33.....?....`
0x0010: 00 00 00 27 3a 40 fe 80 00 00 00 00 00 80 b6 ...':@.....
0x0020: 41 11 99 79 d4 8c ff 02 00 00 00 00 00 00 00 00 A..y.....
0x0030: 00 00 00 00 00 fb 80 00 5f 9f 00 00 00 00 54 45 ....._.....TE
0x0040: 53 54 20 2d 20 49 50 76 36 20 2b 20 49 43 4d 50 ST - IPv6 + ICMP
0x0050: 76 36 45 63 68 6f 52 65 71 75 65 73 74 v6EchoRequest
```

Obrázek 14: IPv6 ICMPv6 - výstup skriptu



```
35... 2565.9143669... fe80::80b6:4111:997... ff02::fb ICMPv6 93 Echo (ping) request id=0x0000, seq=0, hop
Frame 35747: 93 bytes on wire (744 bits), 93 bytes captured (744 bits) on interface wlp0s20f3, id 0
Ethernet II, Src: IntelCor_3f:89:b9 (d8:f2:ca:3f:89:b9), Dst: IPv6mcast_fb (33:33:00:00:00:fb)
Internet Protocol Version 6, Src: fe80::80b6:4111:9979:d48c, Dst: ff02::fb
Internet Control Message Protocol v6
0000 33 33 00 00 00 fb d8 f2 ca 3f 89 b9 86 dd 60 00 33.....?....`
0010 00 00 00 27 3a 40 fe 80 00 00 00 00 00 80 b6 ...':@.....
0020 41 11 99 79 d4 8c ff 02 00 00 00 00 00 00 00 00 A..y.....
0030 00 00 00 00 00 fb 80 00 5f 9f 00 00 00 00 54 45 ....._.....TE
0040 53 54 20 2d 20 49 50 76 36 20 2b 20 49 43 4d 50 ST - IPv6 + ICMP
0050 76 36 45 63 68 6f 52 65 71 75 65 73 74 v6EchoRe quest
```

Obrázek 15: IPv6 ICMPv6 - výstup Wiresharku

4 Záver

Projekt bol pre mňa veľmi zaujímavý, zo začiatku trocha odstrašujúci ale keď si človek nájde dobré materiály a načíta dokumentáciu tak to nieje nič nezvládnuteľné. Hodnotím ho ako jeden z najlepších zatiaľ na FITE. Čo sa týka skriptu ako takého, myslím si, že všetku funkcionality zo zadania zvláda, bol poriadne otestovaný ako na lokálnom tak aj na referenčnom stroji.

Literatura

- [1] getopt_long(3) - Linux man page. [online], dokumentácia. Dostupné z: https://linux.die.net/man/3/getopt_long
- [2] Header Files for UNIX-Type Functions. [online], dokumentácia. Dostupné z: https://www.ibm.com/docs/en/i/7.4?topic=ssw_ibm_i_74/apis/unix13.htm
- [3] inet_ntoa(3) - Linux man page. [online], dokumentácia. Dostupné z: https://linux.die.net/man/3/inet_ntoa
- [4] inet_ntop(). [online], dokumentácia. Dostupné z: http://www.qnx.com/developers/docs/6.5.0/index.jsp?topic=%2Fcom.qnx.doc.neutrino_lib_ref%2Fi%2Finet_ntop.html
- [5] pcap_findalldevs example. [online], publikované 21. januára 2014, autor neuvodený. Dostupné z: <http://embeddedguruji.blogspot.com/2014/01/pcapfindalldevs-example.html>
- [6] Mižiková, A.: Čo je Packet Sniffing? [online], publikované 21. novembra 2017, upravené 8. júna 2018. Dostupné z: <https://blogit.sk/co-je-packet-sniffing/>
- [7] chux Reinststate Monica: I'm trying to build an RFC3339 timestamp in C. How do I get the timezone offset? [online], publikované 13. februára 2018, vlákno s odpoveďou na danú otázku. Dostupné z: <https://stackoverflow.com/a/48772690>
- [8] Tanwar, P.: Capturing Packets in Your C Program, with libpcap. [online], publikované 1. februára 2011. Dostupné z: <https://www.opensourceforu.com/2011/02/capturing-packets-c-program-libpcap/>