VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Počítačové komunikace a sítě

Sniffer paketů – Varianta ZETA

2019/2020

Tomáš Ďuriš
(xduris05)

Banská Bystrica
18.04.2020

Obsah

1. Úvod

Táto dokumentácia popisuje projekt do predmetu *Počítačové komunikace a sítě.* Našim zadaním bolo vytvoriť jednoduchý sieťový analyzátor v C/C++/C#, ktorý bude schopný na zadanom sieťovom rozhraní zachytávať a filtrovať pakety podľa nastavených filtrov užívateľom. Mnou riešená varianta je implementovaná v jazyku C#, keďže s týmto jazykom som už mal predchádzajúce skúsenosti a prišlo mi, že práve v ňom dokážem podať najlepší výkon a vytvorená aplikácia bude plne funkčná.

Dokumentácia okrem spôsobu implementácie taktiež popisuje použité zdroje a knižnice, naštudovanú literatúru, vlastné spôsoby riešenia problematík a taktiež spôsob otestovania projektu a záverečné zhodnotenie.

2. Naštudovaná literatúra

Pri tvorbe projektu bolo nutné vyhľadať a získať informácie, ktoré by doplnili informácie získané z prednášok. K tomuto som prevažne využíval na prednáškach spomenutú knihu Počítačové sítě (Kurose & Ross, 2014[1]), ktorú som si vypožičal v knižnici v českom vydaní. Kniha mi pomohla pochopiť okrem iného všeobecné princípy fungovania paketov, typy a taktiež aj využitie protokolov. K následnému uplatneniu vedomostí do praxe som využíval prevažne dokumentáciu k použitej knižnici *Sharppcap*[2] a nimi poskytnuté oficálne príklady[3]. Základom pre pochopenie ako má projekt fungovať bolo uvedomenie si ako vlastne fungujú jednotlivé vrstvy, kde sa ktorá informácia nachádza a ako k nim pristupovať.

Tieto informácie, ktoré som získal prevažne z prednášok a naštudovania knihy, mi poskytli ucelený obraz o problematike a následná implementácia už nebola náročná. Taktiež som využíval dokumentáciu ku knižnici pcap, určenej pre C/C++, keďže je priamo podporovaná a využívaná knižnicou Sharppcap určenou pre C#.

3. Implementácia

Ako som už vyššie spomínal, program je napísaný v jazyku C#, za využitia knižnice Sharppcap, ktorá sprostredkuje funkcie odporúčanej knižnice pcap do prostredia jazyku C#. Projekt je napísaný objektovo orientovane. Sú využité základné schopnosti a princípy tejto metodiky. Program pri úspechu končí vypísaním obsahu zadaného počtu paketov v požadovanom formáte. Vypísané sú len pakety, ktoré spĺňajú zadané filtry a nie sú chybné (funkcia ***RawPacket GetNextPacket()*** nám nevrátila hodnotu *null*). V prípade neúspechu (napríklad nesprávne argumenty na vstupe) program končí s návratovým kódom 1 a vypísaním chyby a krátkej nápovedy.

[1] Kurose, J. and Ross, K., 2014. *Počítačové Sítě*. Brno: Computer Press.

[2] https://github.com/chmorgan/sharppcap

[3] https://github.com/chmorgan/sharppcap/tree/master/Examples

3.1. Preklad a spustenie

Na preklad projektu využívam upravený Makefile, ktorý po zadaní požiadavku na preloženie (***make*** prípadne ***make build***) preloží program prostredníctvom príkazu dotnet publish. Program je preložený ako samostatný spustiteľný súbor určený pre OS Linux, ktorý je uložený v koreňovom adresári.

Po preložení je projekt možno spustiť pomocou príkazového riadku po zadaní príkazu:

```
./ ipk-sniffer <args>
```

Pričom ***<args>*** označuje uživateľom zadané argumenty, ktorých kompletné znenie je bližšie popísané v priloženom README.md, v sekcii 3.3 alebo po zadaní argumentu ***-h|--help***. V prípade nesprávneho spustenia programu je spolu s nápovedou vypísaná aj krátka chybová hláška oznamujúca uživateľovi akej chyby sa dopustil, prípadne čo je od neho očakávané.

3.2. Hierarchia súborov

Program sa skladá z nasledujúcich súborov:

- **ipk-sniffer.sln, ipk-sniffer.csproj** - definícia častí projektu a potrebných knižníc pre úspešné zostavenie. Časti špecifické pre vývoj v prostredí C#
- **Sniffer.cs** - hlavné telo programu
- **Args.cs** - kontrola argumentov a uchovanie si potrebných informácií
- **PacketProcessing.cs** - spracovanie paketov a výpis získaných informácií na štandardný výstup
- **DnsCache.cs** – implementácia rozširujúcej cache triedy, na uloženie prekladu IP adries na doménové mená

3.3. Kontrola Argumentov

Kontrola užívateľom zadaných argumentov na vstupe prebieha ručne, keďže mi to prišlo ako najspoľahlivejší spôsob odchytenia rôznych kombinácií, prípadne k jej zakázaniu. V hlavnom tele programu sa vytvorí inštancia triedy **Args**, na ktorú sa neskôr zavolá metóda ***Main()***, ktorá vracia filter v podobe reťazca, ktorý neskôr aplikujeme na zachytávanie paketov. Všetky užívateľom zadané argumenty sú skontrolované cyklom foreach, v ktorom si do pomocných *bool* premenných uložíme informáciu o ich prípadnej inicializácii a hodnoty, ktorá bola zadaná. Program nepripúšťa viacnásobné zadanie toho istého argumentu z dôvodu poskytnutia jednoznačnej a intuitívnej spolupráce s užívateľom. Argumenty môžu byť taktiež zadané v ľubovoľnom poradí.

Následne, na základe získaných informácií prebehne vytvorenie inštancie ***CaptureDeviceList.Instance***, do ktorej je priradené užívateľom zadané rozhranie. Ak rozhranie nebolo poskytnuté, vypíše sa zoznam aktívnych rozhraní pomocou cyklu cez všetky inštancie ***CaptureDeviceList***, čo zodpovedá rozhraniam, ktoré vypíše aplikácia WireShark po spustení. Program pokračuje analýzou a komplexným zostavením filtra, ktorý sa využije na filtrovanie prichádzajúcich paketov. Z dôvodu implementácie rozšírenia o protokoly typu IGMP, ICMPv4 a ICMPv6 som musel implementovať rozsiahlejšiu analýzu argumentov. Rozširujúce argumenty sú plne kompatibilné so základnými argumentami, aj ich prípadnou kombináciou s číslom portu. Ako bonus som implementoval taktiež podporu zadania viacerých portov oddelených čiarkov, prípadne aj rozpätie portov. Keďže protokoly IGMP, ICMPv4 a ICMPv6 nepracujú s portom, v ich hlavičke bude uvedené číslo portu ako 0. V prípade kombinácii rozširujúcich argumentov so základnými a zároveň bude uživateľ požadovať filtráciu iba na určitý port, bude tento filter aplikovaný len na TCP alebo UDP paket. V prípade požiadavky na vypísanie len IGMP, ICMP paketov, prípadne ich rôznej kombinácii nebude užívateľ schopný zadať filter na port. V prípade, že dôjde k jeho zadaniu, program sa ukončí s chybou a nápovedou.

V triede Args sa okrem spomínanej metódy ***Main()***, vyskytujú aj metódy ***Help()*** a ***Exit()***. Metóda ***Help()*** vracia reťazec s nápovedou, ktorý

bude vypísaný na štandardný výstup. Metóda ***Exit()*** vypíše chybovú hlášku, nápovedu a následne sa program ukončí s návratovou hodnotou 1.

3.4. Spracovanie jednotlivých paketov

Ak prebehne kontrola argumentov bez ukončenia programu s chybou, nasleduje pripravenie zariadenia na zachytávanie paketov (prostredníctvo metódy ***Open()***, ktorej je predaný mód zachytávania paketov a čas pre zápis do buffera) a aplikácia zostaveného filtra. Pakety sú zachytávané v cykle *for* od 0 do očakávaného počtu zachytených paketov. Paket je zachytený ako inštancia triedy ***RawCapture***. Následne sa tento paket predá funkcii ***onPacketArrival()***, ktorá slúži ako handler pre prichádzajúce pakety a zároveň je to hlavná časť spracovania samostatného paketu.

Ako prvé si z paketu získame jeho čas. Paket rozbalíme použitím metódy ***ParsePacket()*** na základe jeho linkovej vrstvy a dát. Zo získaného paketu už môžeme jednoducho získať jeho IP hlavičku s údajmi o zdrojovej IP adrese a cieľovej IP adrese, z ktorých sa následne pokúsime získať ich doménové meno z implementovanej lokálnej pamäti. Ak ešte nemáme uložený vyhovujúci záznam použijeme vstavanú funkciu ***Dns.GetHostEntry()***. V prípade úspechu pôvodnú IP adresu nahradíme práve týmto menom, následne záznam uložíme do našej pamäti (slovník ***DnsCache.Cache***). Aby sme získali zdrojový alebo cieľový port, je potrebné tento paket rozbaliť ako TCP/UDP paket, kde sa už požadovaná informácia nachádza. Po získaní potrebných informácií vypíšeme hlavičku paketu prostredníctvom metódy ***WriteHeader()***, vypočítame veľkosť hlavičky (ako hlavičku som sa rozhodol brať všetko čo nie sú TCP alebo UDP dáta, v prípade rozširujúcich IGMP/ICMP paketov všetko čo nie je súčasť IGMP/ICMP časti paketu).

3.5. Výpis paketov

Na výpis paketu v požadovanom formáte slúži metóda ***PacketsBytesProcess()***, ktorá prejde cyklom foreach celkový obsah pôvodného paketu aj s údajmi o linkovej vrstve, IP hlavičke, TCP/UDP

dátach a iných častiach. Každý byte je prevedený do jeho hexadecimálnej podoby, je rozlíšený či sa jedná o vytlačiteľný znak, následne prevedený do jeho ASCII podoby a pripojený do reťazca znakov. Po prevedení celého riadku do požadovaného formátu je tento riadok vypísaný pomocou metódy ***WriteString()***. V programe rozlišujem, či sa jedná o hlavičku alebo nie. Ak paket obsahuje aj hlavičku aj dáta, ich časti sú oddelené novým riadkom a počítadlo byteov je správne upravené.

Na výpis a celkovú úpravu taktiež používam metódu ***AlignGenerate()***, ktorá vyhodnotí, či je potrebné vyplniť miesta medzerami aby bol výsledný výstup upravený, a pre užívateľa čo najčitateľnejší.

Po vypísaní všetkých paketov, ktoré boli prepustené filtrom, a keď je naplnený očakávaný počet vypísaných paketov (argument -n na vstupe), je otvorené rozhranie uzavreté a program úspešne ukončený.

4. Zaujímavé a rozširujúce časti implementácie

Za zaujímavú časť považujem hlavne moju implementáciu rozšírenia. Zostavovanie filtra, ktorý by podporoval všetky možné kombinácie protokolov a ich prípadnej filtrácie bolo náročné a vyžadovalo si premyslieť možnosti a zároveň čo by asi užívateľ od môjho programu očakával. Myslím si, že sa mi podarilo vyhovieť všetkým jeho kombináciam a výstup je prehľadný a čitateľný. Implementovaná je taktiež nielen podpora pre IPv4 ale aj IPv6 a to aj pre protokoly rozšírujúce zadanie (ICMPv6). Zároveň je taktiež implementovaná podpora zadania viacerých portov alebo rozpätí portov, na ktorých bude program „odpočúvať“.

Ďalšou zaujímavou situáciou bolo implementovanie triedy ***DnsCache***, ktorá rozširuje pôvodné zadanie o ukladanie prekladu IP adries na doménové mená do slovníka. V slovníku je zachované poradie vkladania a je implementovaný ako cyklický. Jednoducho teda dokážeme pri naplnení kapacity odstrániť najstarší záznam „zo spodu“ a „na vrch“ vložíme najnovší. Kapacitu som obmedzil na 101 záznamov, čo mi prišlo ako úplne postačujúce aj pri dlhšom behu programu a zároveň je nárok na pamäť minimálny.

5. Testovanie a porovnávanie

Ako vzor pre testovanie som používal už existujúci open-source program WireShark, ktorý bol spomenutý aj školskom fóre ako možný etalón pre náš program. Myslím si, že som dosiahol dobrý výsledok, keďže môj výpis je identický s tým, čo vypíše WireShark po spustení s rovnakými filtrami a na rovnakom rozhraní. Ide o rovnaké množstvo paketov, rovnaký obsah paketov a rovnaká vizualizácia dát, pričom môj program podporuje naviac oddelenie hlavičky od dát.

Testovanie bolo realizované ako aj na referenčnom virtuálnom stroji (distribúcia Ubuntu), tak aj na mojom operačnom systéme (distribúcia Arch). V oboch prípadoch bol výsledok zhodný s výsledkom v programe WireShark a program bol plne funkčný. Testovaná bola ako aj verzia IPv4 tak aj verzia IPv6, otestovanie prebehlo aj na rozširujúce protokoly IGMP, ICMPv4 a ICMPv6. Nižšie prikladám výstupy môjho programu a programu WireShark pre rovnaký vstup.

Všetky príkazy boli vykonané s oprávnením správcu. *Za hlavičku som sa rozhodol považovať všetko čo nie sú TCP/UDP dáta (v prípade ICMP/IGMP protokolu, všetko čo nieje súčasťou daného protokolu).* Testovanie prebiehalo na rozhraniach *enp0s3, lo* a *any. Na vygenerovanie paketov boli použité následujúce príkazy:*

- Bežné načítanie HTTP/HTTPS stránky v prehliadači
- *echo -n "test" >/dev/udp/localhost/8000* - UDP a ICMPv4 pakety na konkrétny port.
- *ping ::1* - ICMPv6 paket
- *ssmping google.com* - IGMP paket
- *curl -g -6 "http://[::1]:80/"* - podpora IPv6 na TCP pakete za využitia nástroja curl[4]

[4] https://curl.haxx.se/

Príkaz:

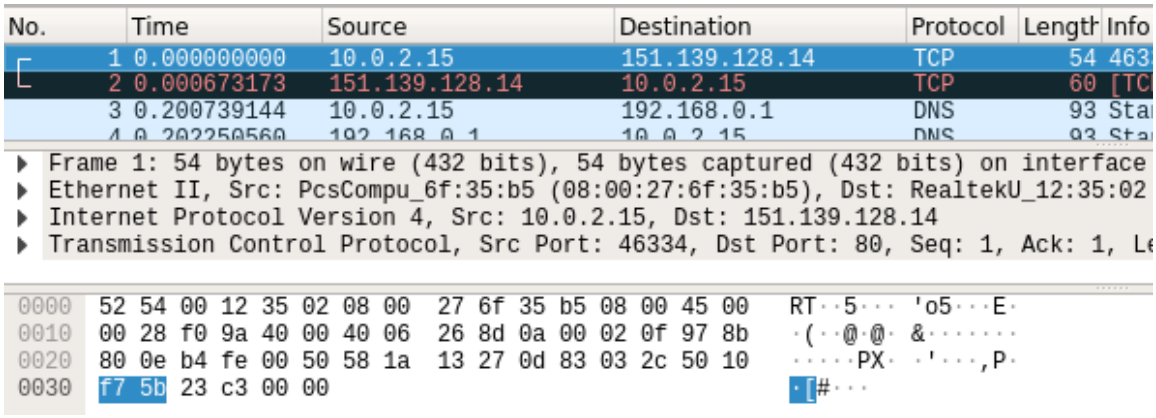
```
./ipk-sniffer -i enp0s3
```

Odchytanie TCP alebo UDP paketu na rozhraní enp0s3 (referenčný virtuálny stroj).

Program **ipk-sniffer**

```
12:44:22.811 student-vm : 46334 > 151.139.128.14 : 80

0x0000:  52 54 00 12 35 02 08 00  27 6f 35 b5 08 00 45 00  RT..5... 'o5...E.
0x0010:  00 28 f0 9a 40 00 40 06  26 8d 0a 00 02 0f 97 8b  .(..@.@. &.......
0x0020:  80 0e b4 fe 00 50 58 1a  13 27 0d 83 03 2c 50 10  .....PX. .'...,P.
0x0030:  f7 5b 23 c3 00 00                                .[#...
```

Program **WireShark**



Príkaz:

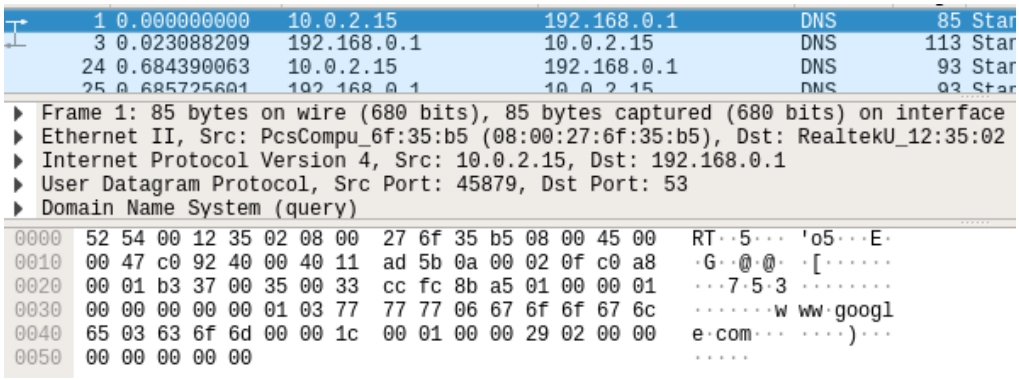
```
./ipk-sniffer -i enp0s3 -u -p 53
```

Odchytenie UDP paketu s filtrom na port 53 (referenčný virtuálny stroj)

Program **ipk-sniffer**

```
12:46:20.138 student-vm : 45879 > OSK : 53

0x0000:  52 54 00 12 35 02 08 00  27 6f 35 b5 08 00 45 00  RT..5... 'o5...E.
0x0010:  00 47 c0 92 40 00 40 11  ad 5b 0a 00 02 0f c0 a8  .G..@.@. .[......
0x0020:  00 01 b3 37 00 35 00 33  cc fc                    ...7.5.3 ..

0x002A:  8b a5 01 00 00 01 00 00  00 00 00 01 03 77 77 77  ........ .....www
0x003A:  06 67 6f 6f 67 6c 65 03  63 6f 6d 00 00 1c 00 01  .google. com.....
0x004A:  00 00 29 02 00 00 00 00  00 00 00                ..)..... ...
```

Program **WireShark**



Príkaz:

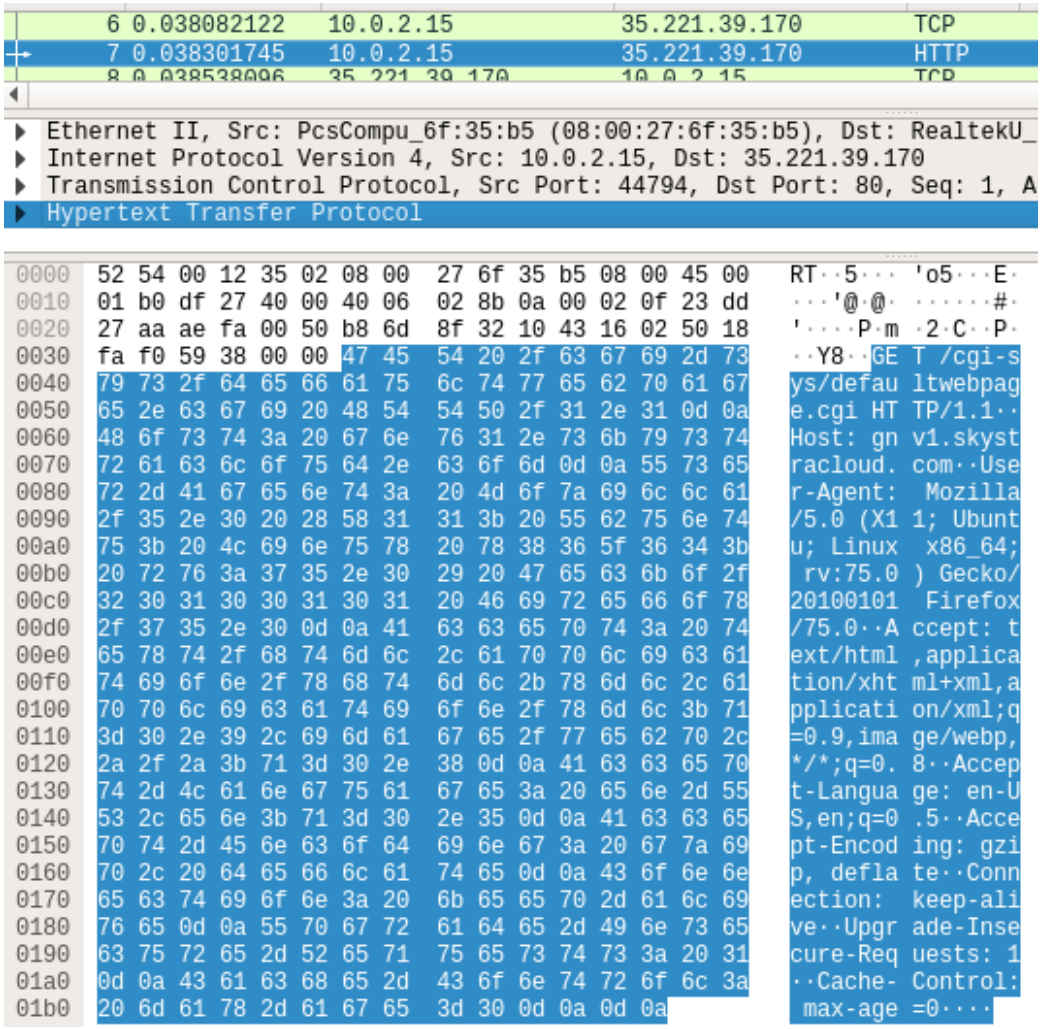
```
./ipk-sniffer -i enp0s3 -u -t -p 80 -n 10
```

Odchytenie TCP/UDP paketov v počte 10 s filtrom nastaveným na port 80 (z dôvodu už aj tak obsiahlej dokumentácie testovania som vyňal 1 paket, na ktorom je najlepšie vidno, kde som sa rozhodol rozdeliť hlavičku a telo správy, referenčný virtuálny stroj).

Program **ipk-sniffer**

```
12:50:00.905 student-vm : 44794 > gnv1.skystracloud.com : 80

0x0000:  52 54 00 12 35 02 08 00  27 6f 35 b5 08 00 45 00  RT..5... 'o5...E.
0x0010:  01 b0 df 27 40 00 40 06  02 8b 0a 00 02 0f 23 dd  ...'@.@. ......#.
0x0020:  27 aa ae fa 00 50 b8 6d  8f 32 10 43 16 02 50 18  '....P.m .2.C..P.
0x0030:  fa f0 59 38 00 00                                 ..Y8..

0x0036:  47 45 54 20 2f 63 67 69  2d 73 79 73 2f 64 65 66  GET /cgi -sys/def
0x0046:  61 75 6c 74 77 65 62 70  61 67 65 2e 63 67 69 20  aultwebp age.cgi
0x0056:  48 54 54 50 2f 31 2e 31  0d 0a 48 6f 73 74 3a 20  HTTP/1.1 ..Host:
0x0066:  67 6e 76 31 2e 73 6b 79  73 74 72 61 63 6c 6f 75  gnv1.sky straclou
0x0076:  64 2e 63 6f 6d 0d 0a 55  73 65 72 2d 41 67 65 6e  d.com..U ser-Agen
0x0086:  74 3a 20 4d 6f 7a 69 6c  6c 61 2f 35 2e 30 20 28  t: Mozil la/5.0 (
0x0096:  58 31 31 3b 20 55 62 75  6e 74 75 3b 20 4c 69 6e  X11; Ubu ntu; Lin
0x00A6:  75 78 20 78 38 36 5f 36  34 3b 20 72 76 3a 37 35  ux x86_6 4; rv:75
0x00B6:  2e 30 29 20 47 65 63 6b  6f 2f 32 30 31 30 30 31  .0) Geck o/201001
0x00C6:  30 31 20 46 69 72 65 66  6f 78 2f 37 35 2e 30 0d  01 Firef ox/75.0.
0x00D6:  0a 41 63 63 65 70 74 3a  20 74 65 78 74 2f 68 74  .Accept:  text/ht
0x00E6:  6d 6c 2c 61 70 70 6c 69  63 61 74 69 6f 6e 2f 78  ml,appli cation/x
0x00F6:  68 74 6d 6c 2b 78 6d 6c  2c 61 70 70 6c 69 63 61  html+xml ,applica
0x0106:  74 69 6f 6e 2f 78 6d 6c  3b 71 3d 30 2e 39 2c 69  tion/xml ;q=0.9,i
0x0116:  6d 61 67 65 2f 77 65 62  70 2c 2a 2f 2a 3b 71 3d  mage/web p,*/*;q=
0x0126:  30 2e 38 0d 0a 41 63 63  65 70 74 2d 4c 61 6e 67  0.8..Acc ept-Lang
0x0136:  75 61 67 65 3a 20 65 6e  2d 55 53 2c 65 6e 3b 71  uage: en -US,en;q
0x0146:  3d 30 2e 35 0d 0a 41 63  63 65 70 74 2d 45 6e 63  =0.5..Ac cept-Enc
0x0156:  6f 64 69 6e 67 3a 20 67  7a 69 70 2c 20 64 65 66  oding: g zip, def
0x0166:  6c 61 74 65 0d 0a 43 6f  6e 6e 65 63 74 69 6f 6e  late..Co nnection
0x0176:  3a 20 6b 65 65 70 2d 61  6c 69 76 65 0d 0a 55 70  : keep-a live..Up
0x0186:  67 72 61 64 65 2d 49 6e  73 65 63 75 72 65 2d 52  grade-In secure-R
0x0196:  65 71 75 65 73 74 73 3a  20 31 0d 0a 43 61 63 68  equests:  1..Cach
0x01A6:  65 2d 43 6f 6e 74 72 6f  6c 3a 20 6d 61 78 2d 61  e-Contro l: max-a
0x01B6:  67 65 3d 30 0d 0a 0d 0a                           ge=0....
```

Program **WireShark**



Príkaz:

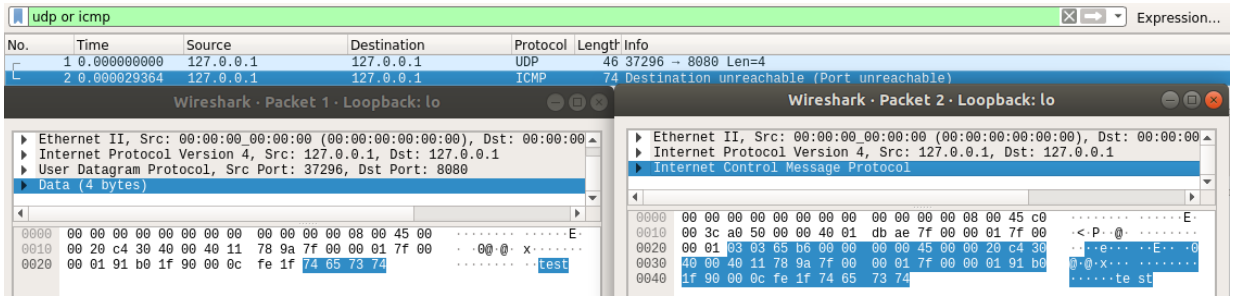
```
./ipk-sniffer -i lo -ic -u -n 2
```

Odchytenie UDP/ICMPv2 paketov na rozhraní localhost v počte 2. Demonštrácia funkčnosti rozšírenia na ICMP pakety (referenčný virtuálny stroj).

Program **ipk-sniffer**

```
12:53:14.698 localhost : 37296 > localhost : 8080

0x0000:  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 00  ........ ......E.
0x0010:  00 20 c4 30 40 00 40 11  78 9a 7f 00 00 01 7f 00  . .0@.@. x.......
0x0020:  00 01 91 b0 1f 90 00 0c  fe 1f                    ........ ..

0x002A:  74 65 73 74                                       test

12:53:14.698 localhost : 0 > localhost : 0

0x0000:  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 c0  ........ ......E.
0x0010:  00 3c a0 50 00 00 40 01  db ae 7f 00 00 01 7f 00  .<.P..@. ........
0x0020:  00 01                                             ..

0x0022:  03 03 65 b6 00 00 00 00  45 00 00 20 c4 30 40 00  ..e..... E.. .0@.
0x0032:  40 11 78 9a 7f 00 00 01  7f 00 00 01 91 b0 1f 90  @.x..... ........
0x0042:  00 0c fe 1f 74 65 73 74                           ....test
```

Program **WireShark**



Príkaz:

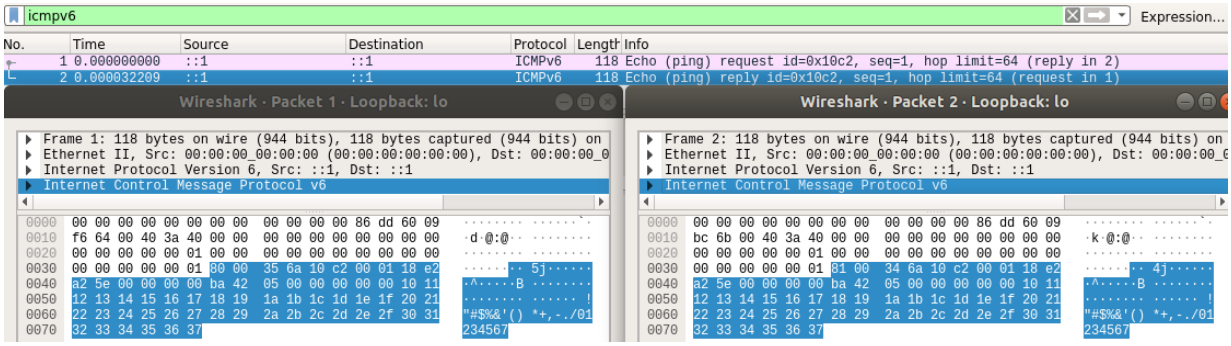
```
./ipk-sniffer -i lo -ic6 -n 2
```

(Podpora IPv6) Odchytenie ICMPv6 paketu na rozhraní localhost. Demonštrácia funkčnosti rozšírenia ICMPv6 (referenčný virtuálny stroj).

Program **ipk-sniffer**

```
12:56:56.344 ip6-localhost : 0 > ip6-localhost : 0

0x0000:  00 00 00 00 00 00 00 00  00 00 00 00 86 dd 60 09  ........ ......`.
0x0010:  f6 64 00 40 3a 40 00 00  00 00 00 00 00 00 00 00  .d.@:@.. ........
0x0020:  00 00 00 00 00 01 00 00  00 00 00 00 00 00 00 00  ........ ........
0x0030:  00 00 00 00 00 01                                 ......

0x0036:  80 00 35 6a 10 c2 00 01  18 e2 a2 5e 00 00 00 00  ..5j.... ...^....
0x0046:  ba 42 05 00 00 00 00 00  10 11 12 13 14 15 16 17  .B...... ........
0x0056:  18 19 1a 1b 1c 1d 1e 1f  20 21 22 23 24 25 26 27  ........  !"#$%&'
0x0066:  28 29 2a 2b 2c 2d 2e 2f  30 31 32 33 34 35 36 37  ()*+,-./ 01234567

12:56:56.344 ip6-localhost : 0 > ip6-localhost : 0

0x0000:  00 00 00 00 00 00 00 00  00 00 00 00 86 dd 60 09  ........ ......`.
0x0010:  bc 6b 00 40 3a 40 00 00  00 00 00 00 00 00 00 00  .k.@:@.. ........
0x0020:  00 00 00 00 00 01 00 00  00 00 00 00 00 00 00 00  ........ ........
0x0030:  00 00 00 00 00 01                                 ......

0x0036:  81 00 34 6a 10 c2 00 01  18 e2 a2 5e 00 00 00 00  ..4j.... ...^....
0x0046:  ba 42 05 00 00 00 00 00  10 11 12 13 14 15 16 17  .B...... ........
0x0056:  18 19 1a 1b 1c 1d 1e 1f  20 21 22 23 24 25 26 27  ........  !"#$%&'
0x0066:  28 29 2a 2b 2c 2d 2e 2f  30 31 32 33 34 35 36 37  ()*+,-./ 01234567
```

Program **WireShark**



Príkaz:

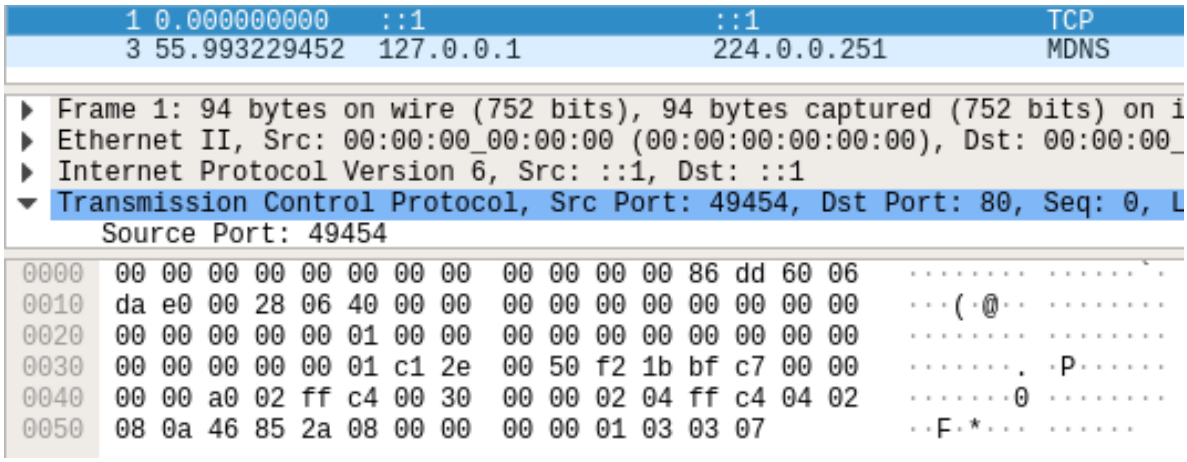
```
./ipk-sniffer -i lo
```

(Podpora IPv6) Odchytenie TCP/UDP paketu na rozhraní localhost (IPv6 verzia, referenčný virtuálny stroj).

Program **ipk-sniffer**

```
12:59:25.953 ip6-localhost : 49454 > ip6-localhost : 80

0x0000:  00 00 00 00 00 00 00 00  00 00 00 00 86 dd 60 06  ........ ......`.
0x0010:  da e0 00 28 06 40 00 00  00 00 00 00 00 00 00 00  ...(.@.. ........
0x0020:  00 00 00 00 00 01 00 00  00 00 00 00 00 00 00 00  ........ ........
0x0030:  00 00 00 00 00 01 c1 2e  00 50 f2 1b bf c7 00 00  ........ .P......
0x0040:  00 00 a0 02 ff c4 00 30  00 00 02 04 ff c4 04 02  .......0 ........
0x0050:  08 0a 46 85 2a 08 00 00  00 00 01 03 03 07        ..F.*... ......
```

Program **WireShark**



Príkaz:

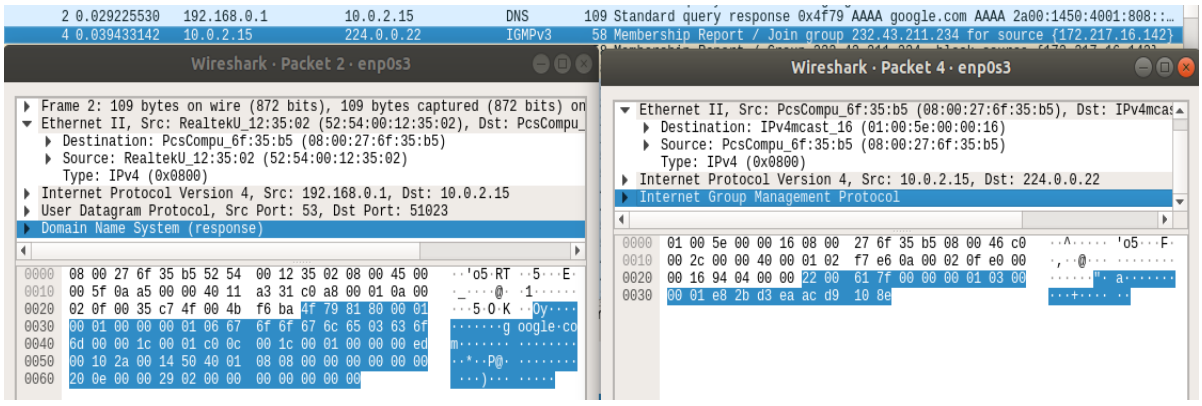
```
./ipk-sniffer -i enp0s3 -ig -ic -u -t -p 53,80 -pr 442-444,900-999 -n 5
```

Komplexný príklad odchytenia piatich paketov s rôznymi protokolmi zadaných užívateľom (Zobrazenie dvoch demonštračných paketov: UDP paketu spĺňajúceho filtry na port a IGMP paketu). Demonštrácia podpory aplikovania filtru na port, avšak je aplikovaný len na TCP/UDP, podpora IGMP. Podpora zadania viacerých portov, podpora zadania viacerých rozpätí portov a ich vzájomná kombinácia (referenčný virtuálny stroj).

Program **ipk-sniffer**

```
15:11:22.127 192.168.0.1 : 53 > student-vm : 51023

0x0000:  08 00 27 6f 35 b5 52 54  00 12 35 02 08 00 45 00  ..'o5.RT ..5...E.
0x0010:  00 5f 0a a5 00 00 40 11  a3 31 c0 a8 00 01 0a 00  ._....@. .1......
0x0020:  02 0f 00 35 c7 4f 00 4b  f6 ba                    ...5.O.K ..

0x002A:  4f 79 81 80 00 01 00 01  00 00 00 01 06 67 6f 6f  Oy...... .....goo
0x003A:  67 6c 65 03 63 6f 6d 00  00 1c 00 01 c0 0c 00 1c  gle.com. ........
0x004A:  00 01 00 00 00 ed 00 10  2a 00 14 50 40 01 08 08  ........ *..P@...
0x005A:  00 00 00 00 00 00 20 0e  00 00 29 02 00 00 00 00  ...... . ..).....
0x006A:  00 00 00                                        ...

15:11:22.137 student-vm : 0 > igmp.mcast.net : 0

0x0000:  01 00 5e 00 00 16 08 00  27 6f 35 b5 08 00 46 c0  ..^..... 'o5...F.
0x0010:  00 2c 00 00 40 00 01 02  f7 e6 0a 00 02 0f e0 00  .,..@... ........
0x0020:  00 16 94 04 00 00                               ......

0x0026:  22 00 61 7f 00 00 00 01  03 00 00 01 e8 2b d3 ea  ".a..... .....+..
0x0036:  ac d9 10 8e                                     ....
```

Program **WireShark**



6. Referencie

1. Gal, T. and Morgan, C., 2020. *Sharppcap - A Packet Capture Framework For .NET.* [online] Codeproject.com. Dostupné na: https://www.codeproject.com/articles/12458/sharppcap-a-packet-capture-framework-for-net [cit. 2020-04-23].
2. GitHub. 2020. *Chmorgan/Sharppcap*. [online] Dostupné na: https://github.com/chmorgan/sharppcap [cit. 2020-04-23].
3. GitHub. 2020. Chmorgan/Sharppcap. [online]. Dostupné na: https://github.com/chmorgan/sharppcap/tree/master/Examples [cit. 2020-04-23].
4. Kurose, J. a Ross, K., 2014. *Počítačové Sítě*. Brno: Computer Press. ISBN 9788025138250.
5. Matoušek, P. 2014. *Síťové aplikace a jejich architektura*. VUTIUM. ISBN 9788021437661
6. Sharppcap.sourceforge.net. 2020. *Sharppcap: Sharppcap*. [online] Dostupné na: http://sharppcap.sourceforge.net/htmldocs/SharpPcap/index.html [cit. 2020-04-23].
7. Tcpdump.org. 2020. *Manpage Of PCAP*. [online] Dostupné na: https://www.tcpdump.org/manpages/pcap.3pcap.html [cit. 2020-04-23].
8. Tcpdump.org. 2020. *Programming With Pcaptcpdump/LIBPCAP Public Repository.* [online] Dostupné na: https://www.tcpdump.org/pcap.html [cit. 2020-04-23].