VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Počítačové komunikace a sítě – 2. projekt

Varianta ZETA: Sniffer paketů

23. dubna 2022

Vladislav Mikheda

Obsah

1 Úvod

Cílem projektu je implementování sniffer paketů, který bude poslouchat pakety na určitém rozehraní, které bylo uvedeno uživatelem. Sniffer bude filtrovat a zachytávat pakety podle zadaných parametru filtrace.
Podporuje protokoly:`TCP`, `UDP`, `ARP`, `ICMP` a `ICMPv6`.
A také lze provést filtrace i podle `portu`. Cílem bylo nejen zachytávat pakety a také zanalyzovat a vypsat informaci kterou obsahuje paket.
Na obrázku č:1 je možné vidět spuštění programu na rozhraní `enp4s0f1` na kterém musí být zachycené `TCP` pakety podle `portu 443`.
Ve výstupu je uvedena informace z paketu. Úvodní informace: kdy byl paket přijat, zdrojová MAC adresa,cílová MAC adresa, typ paketu.
Informace je specifikována typem paketu, víc v častí o jednotlivých paketech. Dole uveden výpis bajtů v hexadecimálním a ASCII formátu.

```
> sudo ./ipk-sniffer --interface enp4s0f1 -t -p 443
   timestamp: 2022-04-23T21:16:04.203+02:00
   src MAC: 80:fa:5b:0e:0c:b5
   dst MAC: 38:43:7d:9a:c5:3b
   frame length: 86 bytes (688 bits)
   Type: IPv6
Internet Protocol Version 6 (IPv6)
   Hop Limit: 64
   src IP: 2a02:8308:b083:dc00:7c12:dele:7673:f87d
   dst IP: 2a02:6b8:a::a
Transmission Control Protocol (TCP)
   src port: 47642
   dst port: 443
   Sequence number: 3935170332
   Acknowledgment number: 872647551

0x0000: 38 43 7d 9a c5 3b 80 fa 5b 0e 0c b5 86 dd 60 0c  8C}..;..[.....`.
0x0010: 28 91 00 20 06 40 2a 02 83 08 b0 83 dc 00 7c 12  (.. .@*.......|.
0x0020: de 1e 76 73 f8 7d 2a 02 06 b8 00 0a 00 00 00 00  ..vs.}*.........
0x0030: 00 00 00 00 00 0a ba 1a 01 bb ea 8d ef 1c 34 03  ..............4.
0x0040: 8b 7f 80 10 05 81 33 a6 00 00 01 01 08 0a 36 9c  ......3.......6.
0x0050: 85 40 8e 07 69 51                                .@..iQ
```

Obrázek 1: Příklad spuštění a výstupu

2 Knihovna `Packet Capture`

Pro implementaci projektu byla využita knihovna „pcap“. Knihovna poskytuje vysokoúrovňové rozhraní pro systémy zachycování paketů, přes které jsou dostupné všechny pakety ve síti viz [7].
Knihovna byla vybrána z důvodu doporučení pro projektový úkol. Také po přečtení manuálu viz [7] bylo zjištěno, že knihovna představuje sadu funkcí pro naslouchání a zachycování paketů, skrývá vnitřní implementaci, což značně zjednodušuje práci na projektu . To umožní se soustředit na implementaci práce s pakety bez přímé práce se sokety.
Popis jen hlavních funkcí které byly využity pro projekt:

- `pcap_findalldevs` – funkce vrátí seznam rozhraní které jsou na daném počítače viz [8].
- `pcap_open_live` – funkce vrátí deskriptor zachytávaní paketů viz [5]. Funkce je vyvolaná s parametry:
  - `rozháraní` jméno rozhraní na kterém se budou poslouchat pakety.
  - `délka snímku en(snapshot length)` v kontextu teto funkce reprezentuje se maximální délku paketu který bude zachycen.
  - `promiskuitní režim` je režim ve kterém rozhraní bude přijímat všechny pakety, bez ohledu na to, pro koho jsou určené.
  - `časový limit čtení` parametr ukazuje za kolik milisekund po přijetí paketu a jeho uložení do systémového bufferu, paket bude načten. Pokud například potřebujete číst 3 pakety najednou, můžete nastavit hodnotu tak aby čtení z paketového bufferu se provedlo pro 3 pakety najednou. Nebo můžeme pakety přečíst hned když byly přijaty, pak budou provedeny 3 čtení z bufferu.
- `pcap_datalink` funkce vrátí pro už ověřený deskriptor pro zvolené rozhraní typ záhlaví linkové vrstvě viz [6].
- `pcap_loop` funkce zachycuje zadaný počet paketů v cyklu a vyvolává funkci, která ji předána jako parametr. A pro vyvolanou funkci předává zachycený paket a strukturu `pcap_pkthdr` která obsahuje v sobě délku paketu, čas přijetí paketu v Unix formátu viz [4].

V manuálu je napsáno že funkce `pcap_open_live` je trochu zastaralá a je potřeba radši omezit její využití, může být nahrazená těmito funkcemi: `pcap_create`, `pcap_activate` a t.d. viz [7]. Ale podle testovaní bylo zjištěno že funkce `pcap_open_live` funguje normálně a podporuje staré verze knihovny. Kvůli využití `pcap_open_live` nemohou být využity nové funkce knihovny ale v projektu to není potřeba. Takže v implementaci projektu je použita funkce `pcap_open_live`.

3 Implementace

3.1 Úvod a struktura programu

Pro zpracování projektu byl využit jazyk C++. Program byl implementován s využitím objektové orientovaného přístupu. Bylo zvoleno rozdělení na třídy: `ParseArguments` třída pro analýzu vstupních argumentů, `Sniffer` třída pro zachycení paketů, `ParsePacket` třída pro analýzu již zachycených paketů a pro výpis na standardní vystup analyzované informace.

3.2 Analyzování argumentů

Pro analýzu argumentů jak bylo výše uvedeno slouží třída `ParseArguments` instance které je vytvořena v hlavní funkci programu `main`.
Analýza argumentů je založena na vyžití funkce `getopt_long` která automaticky odděluje jméno argumentů od jejich hodnoty viz [1].
Analýza probíhá v nekonečné smyčce, podmínkou pro výstup ze které je konec argumentů. Určujeme který argument je který pomocí konstrukce `switch`, kde hodnoty argumentu se zapisuji do jednotlivých proměnných tříd. Proměnné třídy mají modifikátor přístupu `privat`.

3.3 Zachycení paketů

Pro jednu z hlavních součásti projektu – zachycení paketů je využita knihovna `pcap` která je už popsaná v sekci 2, nebudou dál popsány její jednotlivé funkce jen se využije jejich název v kontextu popsání detailů implementace.

3.3.1 Třída `Sniffer`

Pro zachycení paketu je navržena třída `Sniffer`. Instance třídy Sniffer je vytvořena ve funkci `main`, nutným parametrem konstruktoru je instance třídy `ParseArguments`. Z třídy `ParseArguments` se pomocí getterů budou použity již analyzované argumenty nastavené uživatelem. Třída obsahuje hlavní metodu `start_sniff` ze které už budou volané jiné metody: bud' na nastavení parametrů a zachycení paketů nebo metoda pro výpis všech rozhraní které jsou získány pomocí funkce `pcap_findalldevs`. Před zachycením paketu je nutné nastavit parametry. Nastavením parametrů je myšleno: získaní deskriptoru a nastaveni filtru.

3.3.2 Získaní deskriptoru

Pro získaní deskriptoru využijeme funkci `pcap_open_live` z knihovny `pcap`. A pro tuto funkci nastavíme:

- `rozhraní` musí být zadané uživatelem
- `délku snímku` na 65535, hodnota byla zvolena podle manuálu viz [7] což odpovídá teoretické maximální délce IP paketu.
- `Režim rozhraní` musíme převést v promiskuitní režim aby nebyly vyhozeny pakety.
- `pro časový limit čtení` bylo zvolené nastavení hodnoty na 100, to nastavení je spíš náhodné, ale byla zvolena taková malá hodnota za účelem rychlého získávání výsledků.

Dál využijeme funkci `pcap_datalink` a zkontrolujeme jestli se shoduje vracený typ s `DLT_EN10MB` co reprezentuje `Ethernet` viz [3].

3.3.3 Nastavení filtru

Pro nastaveni filtru je implementovaná metoda `set_filter`. Ve které na začátku je zpracovaní všech argumentů pro filtraci do jedeného bufferu. Filtrace může být provedena podle portu nebo typu paketu: `UDP, TCP, ICMP, ICMPv6, ARP` nebo spolu. Parametry filtrace jsou zadány uživatelem.

- **je zadán jen typ paketu**, filtrace bude prováděna jen podle zadaných typů paketu.
- **je zadán typ paketu který lze filtrovat podle portu** (`UDP,TCP`) **a port**: filtrace bude prováděna podle zvoleného paketu a podle portu.
- **je zadán typ paketu který nemůže byt filtrován podle portu**(`ARP,ICMP`) **a port** filtrace bude provedena podle zadaného typu paketu a také podle typů paketů `TCP a UDP` podle zvoleného portu.
- **je zadán jen port** filtrace bude provedena podle typu paketu `TCP,UDP` podle zadaného portu.
- **není zadán žádný argument pro filtraci** filtrace bude prováděna podle všech typů paketů

Po dodání argumentů do bufferu je vyvolána funkce `pcap_compile` která zkompiluje filtr a pak `pcap_setfilter` která předem zkompilovaný filtr nastaví do deskriptoru.

3.3.4 Zachycení paketu

Nakonec může být provedeno volaní funkce která udělá všechno samostatně a podle nastavených filtru načte paket který bude přijat na zvoleném rozhraní. Takže voláme funkci `pcap_loop`. A musí ji být předána funkce ve které dál už bude prováděno zpracovaní paketů. Předáváme statickou metodu `packet_parse` tridy `PacketParse` více to bude popsáno v sekci 3.4.

3.4 Zpracovaní paketů

Po zachycení paketu dostáváme ho jako ukazatel na sekvenci bitu. Jsou tří varianty zpracovaní: využit už vytvořené rozhraní struktur z knihoven, vytvořit svoje struktury nebo pracovat s paketem jako se sekvencí bitů co je nebezpečné. Byl zvolen způsob s použitím struktur z knihoven. A budeme provádět zpracovaní paketu postupně:

- Na začátku zpracujeme čas přijetí paketu.
- Pak zpracujeme `Ethernet` hlavičku.
- Podle typu získaného z `Ethernet` hlavičky zjistíme typ paketu.
- A zpracujeme už příští hlavičku bud' to je `IPv4, IPv6, ARP`.
- Když paket je typu `IPv4, IPv6` zjistíme příští typ `TCP,UDP,ICMP` a zpracujeme je.
- vypíšeme data a celý obsah paketu v hexadecimálním a ASCII formátu.

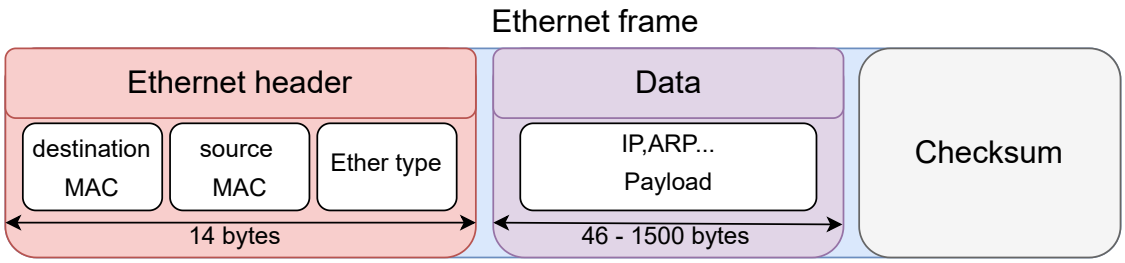
3.4.1 Trida `ParsePacket`

`Parcepacket` je třída která obsahuje metody pro zpracování paketu a vypíše už zpracované data na standardní vystup. Je to třída která obsahuje jen statické metody a to kvůli využití funkce `pcap_loop`, která nepřijímá jako parametr ne statickou metodu.

3.5 Zpracování času zachycení paketu

Čas přijetí paketu je uložen v struktuře `pcap_pkthdr` v Unix formátu – počet sekund od `00:00:00 1.01.1970` viz [10] Podle zadání jej musíme vypsat v `RFC3339` formátu. Pro převod je využita metoda `return_RFC_time`. Pomocí funkce `localtime` formát času je převeden do lokálního a pak už bude zapsán do bufferu v `RFC3339` formátu.

3.5.1 Zpracování Ethernet hlavický

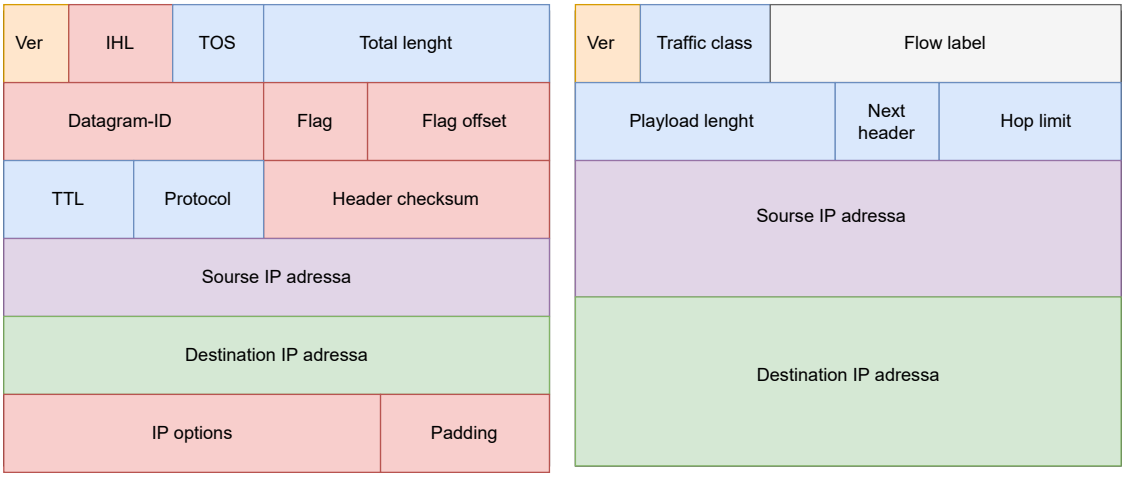


Obrázek 2: Ethernet frame

Jak jde vidět z obrázku č:2 z `Ethernet` hlavický může byt získána cílová `MAC` adresa, zdrojová `MAC` adresa a typ příští hlavičky. Pro zpracování bude použita struktura `ether_header` z knihovny `netinet/ether.h`. Pro získání `MAC` adresy v formátu s dvojtečkou je využita metoda `mac_parse`, ve které pomocí funkce `sprintf` do bufferu se zapisují `MAC` adresu po jednotlivých bitech které jsou převedeny do hex formátu s počtem znaku dva a mezi kterých jsou přidané dvojtečky. Příklad: `"%02x:..."`
Typ příští hlavičky je prostě porovnán s konstantami typů a pomocí konstrukce `switch` je vybrán nutný typ.

3.5.2 Zpracování IPv4/IPv6 hlaviček

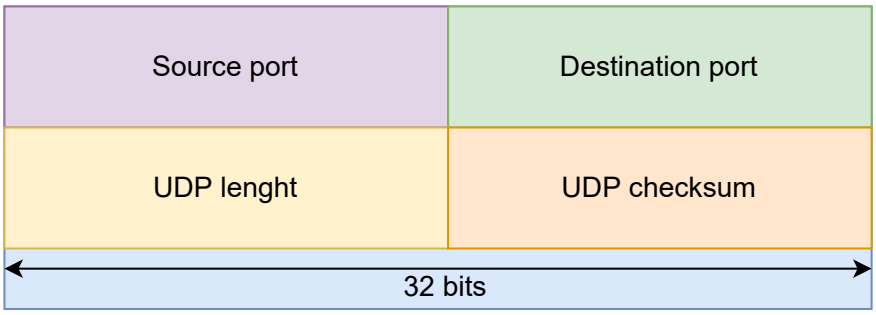
Pro získaní IP hlaviček také využijeme struktury a musíme dodat do ukazatelů délku `Ethernet` hlavičky. Jak je vidět na obrázku č:2 délka `Ethernet` hlavičky je vždy stejná a rovna se 14 bajtů. Hlavický IPv4 a IPv6 jsou různé, takže pro jejich zpracování budou využity různé struktury. Struktura pro IPv4 `iphdr` z knihovny `netinet/ip.h`, pro IPv6 `ip6_hdr` z knihovny `netinet/ip6.h`. Hlavičky budou zpracovány různými metodami ale typ zpracování pro data který potřebujeme je stejný.



Obrázek 3: IPv4 IPv6

Z obrázku č:3.4 jde vidět že hlavičky mají jak položky ze stejným názvem (označeny stejnou barvou), tak i položky z různým názvem na různých místech ale které reprezentují stejné informace (jsou označené modrou barvou). Pro účely projektu je nutné získat IP adresu. Pro její získání je využita funkce `inet_intop` která zapíše do bufferu IP adresu v obyčejném formátu. Ještě zapíšeme do bufferu `Time to Life` z IPv4 hlavičky a `Hop limit` z IPv6 hlavičky, reprezentují stejnou informaci o tom kolik paket můžou cestovat v síti. Také musíme získat délku IPv4 paketu, IPv6 má vždy stejnou v 40 bajtů, a typ příští hlavičky. Délka je reprezentována v poctu slov, každé slovo je 32 bitů. Délka je potřebná pro práci s příští hlavickou. Stejně jak je napsáno na začátku sekce. Typ příští hlavičky v IPv4 je v `Protokolu`, v IPv6 je v `Priste hlavičce`. A podle typu rozhodujeme která hlavická jde dál`UDP`,`TCP`,`ICMP`.

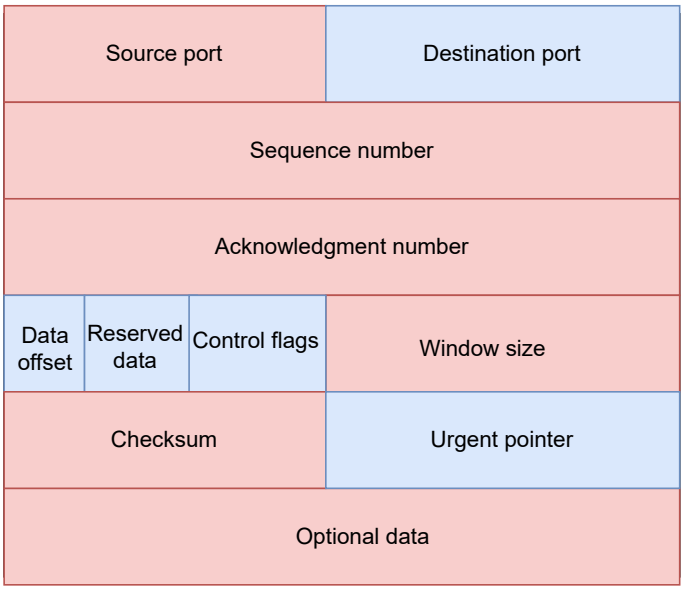
3.5.3 Zpracování UDP hlavičky



Obrázek 4: UDP hlavička

`UDP` určen k zasílaní zprav jiným počítačům viz [11]. Pro `UDP` hlavičku také využijeme strukturu `udphdr` z knihovny `netinet/udp.h`. A získám jen cílový port a zdrojový port. Při zpracování portu je nutné přehodit bity a na to je využita funkce `ntohs`.

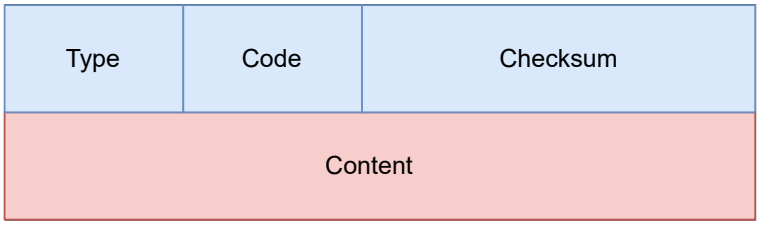
3.5.4 Zpracování TCP hlavičky



Obrázek 5: TCP hlavička

`TCP` transportní protokol přes který počítače mohou přenášet data viz [13]. Pro zpracování `TCP` hlavičky také využijeme strukturu `tcphdr` z knihovny `netinet/tcp.h` A jako u UDP získáme porty.

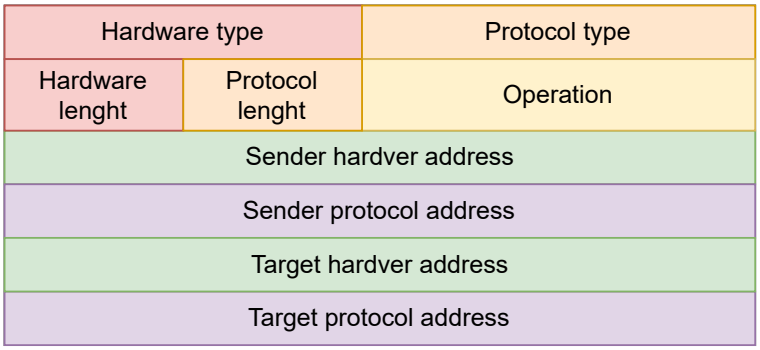
3.5.5 Zpracování ICMP hlaviček



Obrázek 6: ICMP hlavička

Protokol ICMP se používá k hlášení chyb a dalších výjimek, ke kterým dochází při přenosu dat viz [12]. Také využijeme strukturu z knihovny . Program musí podporovat dva ICMP protokoly `ICMP, ICMPv6` ale pro naši účely nemají žádný rozdíl. Pro `ICMP` využijeme strukturu `icmphdr` z knihovny `netinet/ip_icmp.h`, pro `ICMPv6` využijeme strukturu `icmp6_hdr` z knihovny `netinet/icmp6.h`. Z hlavičky zpracované `Typ` reprezentuje typ zprávy, a `kod` specifikuje účel zprávy.

3.5.6 Zpracování ARP hlavičky



Obrázek 7: ARP hlavička

ARP je určen k určení MAC adresy jiného počítače ze známé IP adresy viz [9]. Také využijeme struktury `arphdr` a `ether_arp` z knihovny `if_arp.h`. Zpracujeme `opcod`, který reprezentuje typ operace: odpověď nebo žádost. Stejně jako z `Ethernet` zpracujeme `MAC` adresy: zdrojovou a cílovou. Pak zpracujeme IP adresu a pomocí funkce `sprintf` dodáme ji do bufferu v formátu s tečkami, můžeme to udělat protože ARP pracuje jen s IPv4.

4 Zajímavý pasáží implementace

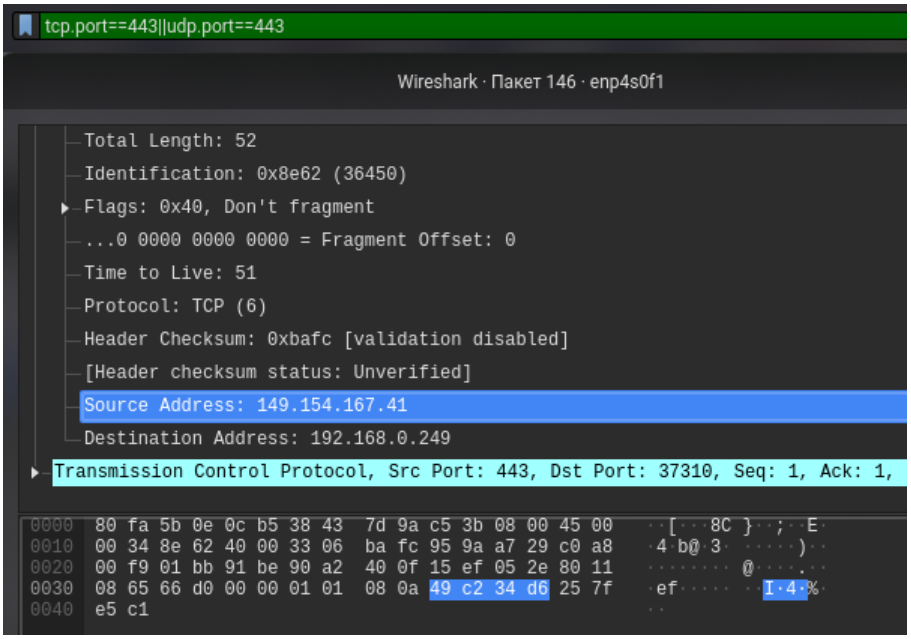
4.1 Getopt

Při implementaci analýzy argumentů pomocí `getopt_long` byl vykřesán problém s argumenty `-i/--interface` protože mají volitelný parametr. Podle manuálu viz [2] takový parametr musí byt zadán podle `POSIX` pravidel: buď za argumentem nebo přes znak =. Kvůli žádné zkušeností s `getopt` a žádné informaci na internetu bylo to vyřešeno pomocí nastavení parametru povinným pro argumenty `-i/--interface` a vypínání chybového hlášení `getopt`. A když dojde k takové situaci, `getopt` vstoupí do `swith: case '?'` kde ukončí metodu analýzy argumentů. Pak budou rozhraní vypsány na standardní vystup.

5 Testování

Testování proběhlo v několik etapách. Program byl testován při návrhu, testovaní probíhalo na vlastním systému[1] bez virtualizace. Pro testování byl využít `Wireshark` při testování skoro vždy mimo testovaní `ARP`, `ICMP` byl ukázán port 443 aby bylo možné stihnout zjistit který paket byl zachován programem a porovnat. V `Wiresharku` byl nastaven stejný filtr.

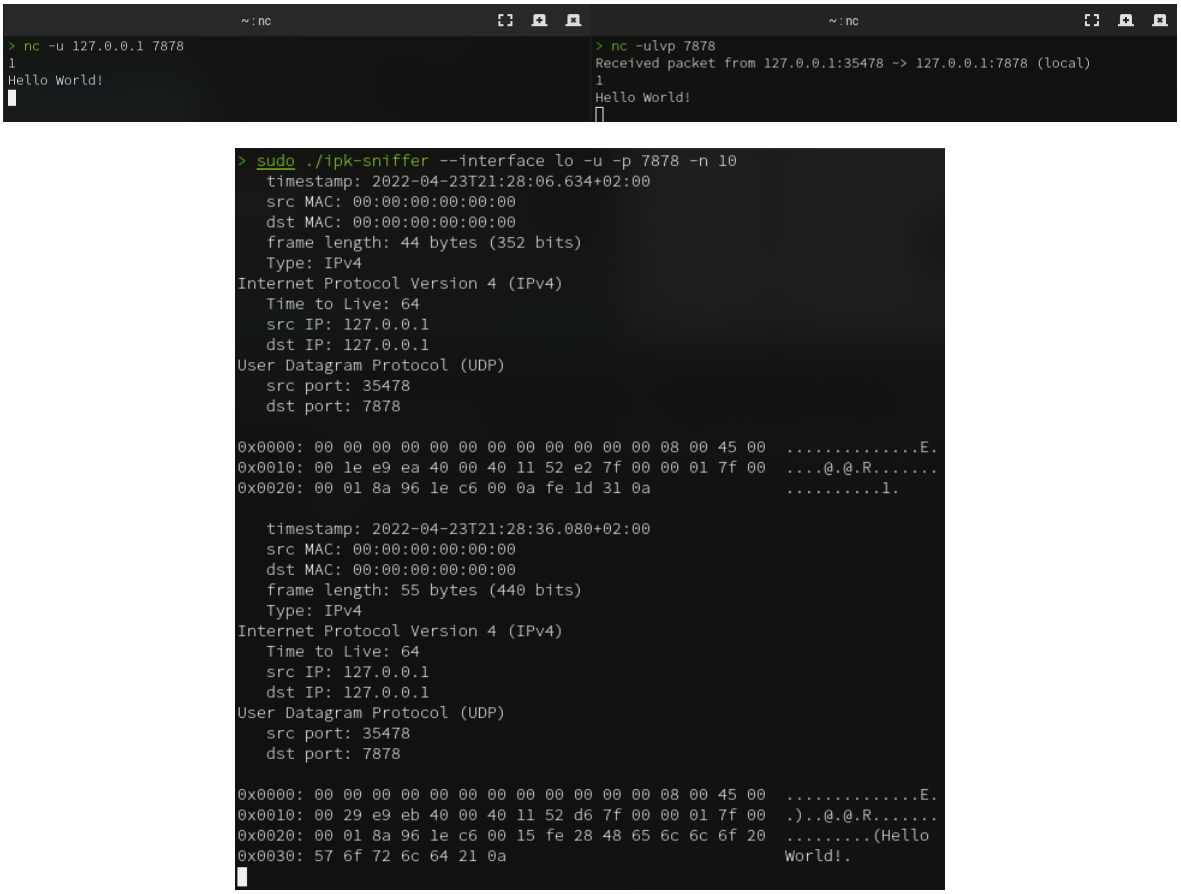
```
> sudo ./ipk-sniffer --interface enp4s0f1 -t -p 443
   timestamp: 2022-04-23T21:18:41.871+02:00
   src MAC: 38:43:7d:9a:c5:3b
   dst MAC: 80:fa:5b:0e:0c:b5
   frame length: 66 bytes (528 bits)
   Type: IPv4
Internet Protocol Version 4 (IPv4)
   Time to Live: 51
   src IP: 149.154.167.41
   dst IP: 192.168.0.249
Transmission Control Protocol (TCP)
   src port: 443
   dst port: 37310
   Sequence number: 2426552335
   Acknowledgment number: 367985966

0x0000: 80 fa 5b 0e 0c b5 38 43 7d 9a c5 3b 08 00 45 00  ..[...8C}..;..E.
0x0010: 00 34 8e 62 40 00 33 06 ba fc 95 9a a7 29 c0 a8  .4.b@.3......)..
0x0020: 00 f9 01 bb 91 be 90 a2 40 0f 15 ef 05 2e 80 11  ........@.......
0x0030: 08 65 66 d0 00 00 01 01 08 0a 49 c2 34 d6 25 7f  .ef.......I.4.%.
0x0040: e5 c1                                            ..
```



Obrázek 8: Test 1

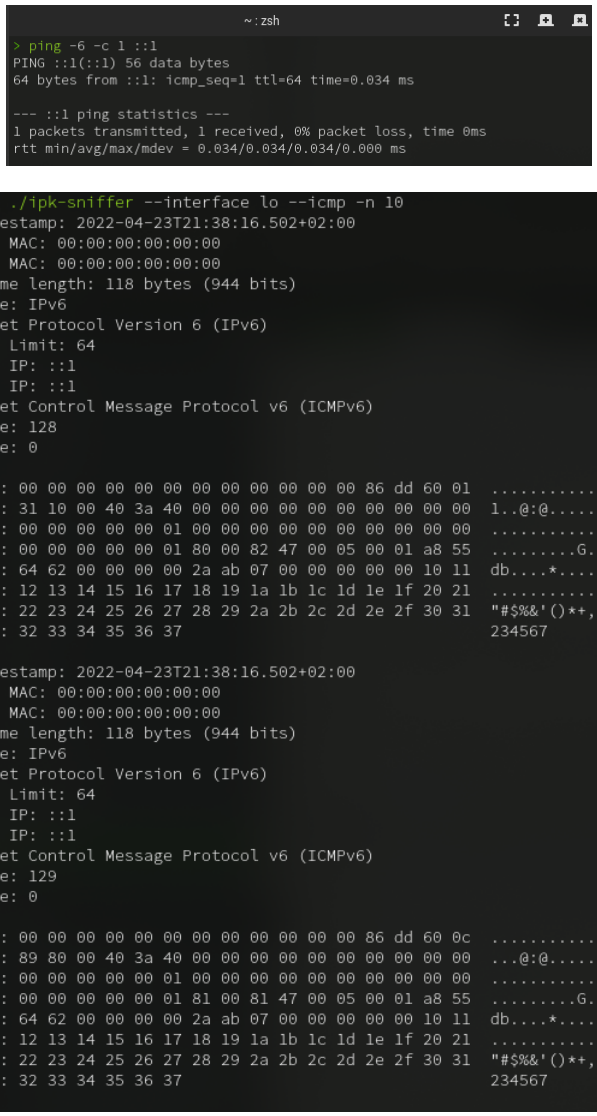
Podle obrázku č:8 jde vidět že data se shodují, takže test je úspěšný.
Také pro testování byl využít `netcat` který byl spuštěn ve dvou příkazových řádkách, v první pro odesílaní sprav na `localhost:7878`, ve druhé na přijaté. A vytvářený program byl spuštěn s argumenty –interface lo -u -p 7878, `Wireshark` byl nastaven podobně.

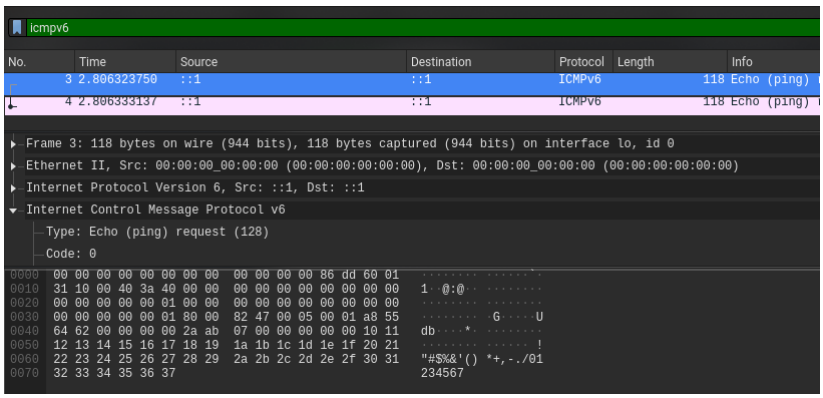
[1] Manjaro Linux



Obrázek 9: Test 2

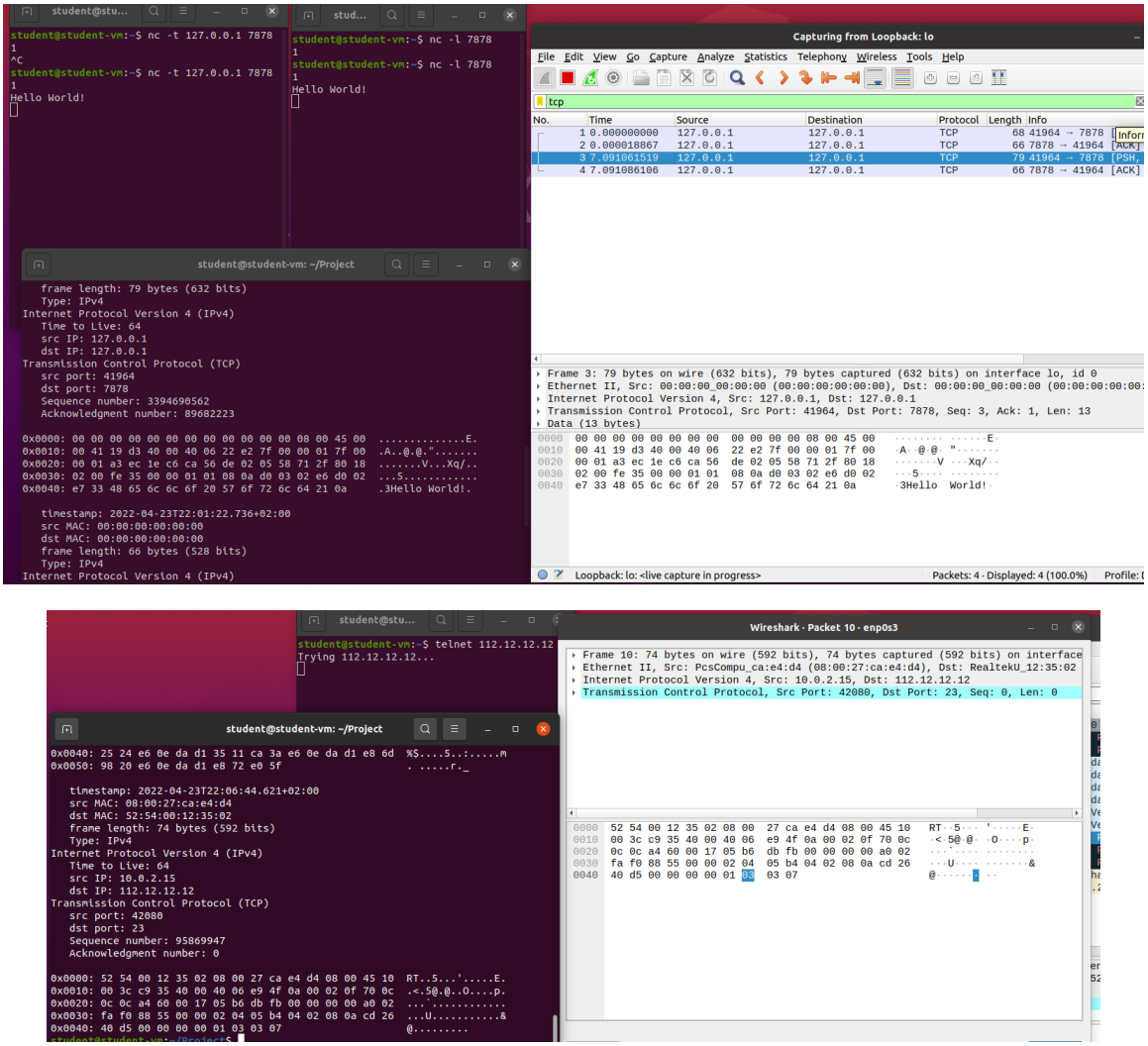
Podle obrázku č:9 jde vidět že data se shoduje, takže test je úspěšný.
Také byly otestovány `ICMP a ICMPv6` protokoly, s využitím nastoje `ping`





Obrázek 10: Test 3

Na virtuálním stroji byla také otestována funkčnost programu pomocí zasílaní tcp paketu na lokálním rozhraní, a také pomocí využití nastroje telnet.



Obrázek 11: Test 4

Po testování lze říct že program funguje správně a zachovává pakety podle požadavku uživatele.

Literatura

[1] Kerrisk, M.: getopt(3) — Linux manual page. [online]. rev. 08.srpna.2021. [vid. 2022-04-20]. Dostupné z: https://www.man7.org/linux/man-pages/man3/getopt.3.html

[2] Repository, U. M.: getopt - parse command options. [online]. rev. 01.září.2019. [vid. 2022-04-20]. Dostupné z: http://manpages.ubuntu.com/manpages/bionic/man1/getopt.1.html

[3] Tcpdump, G.: LINK-LAYER HEADER TYPES. [online]. [vid. 2022-04-20]. Dostupné z: https://www.tcpdump.org/linktypes.html

[4] Tcpdump, G.: MAN PAGE OF PCAP_LOOP. [online]. rev. 05.března.2022. [vid. 2022-04-20]. Dostupné z: https://www.tcpdump.org/manpages/pcap_loop.3pcap.html

[5] Tcpdump, G.: MAN PAGE OF PCAP_OPEN_LIVE. [online]. rev. 06.prosince.2017. [vid. 2022-04-20]. Dostupné z: https://www.tcpdump.org/manpages/pcap_open_live.3pcap.html

[6] Tcpdump, G.: MAN PAGE OF PCAP_DATALINK. [online]. rev. 07.dubna.2014. [vid. 2022-04-20]. Dostupné z: https://www.tcpdump.org/manpages/pcap_datalink.3pcap.html

[7] Tcpdump, G.: MAN PAGE OF PCAP. [online]. rev. 09.září.2020. [vid. 2022-04-20]. Dostupné z: https://www.tcpdump.org/manpages/pcap.3pcap.html

[8] Tcpdump, G.: MAN PAGE OF PCAP_FINDALLDEVS. [online]. rev. 23.srpna.2018. [vid. 2022-04-20]. Dostupné z: https://www.tcpdump.org/manpages/pcap_findalldevs.3pcap.html

[9] Wikipedia, t. f. e.: Address Resolution Protocol. [online]. rev. 12.dubna.2022. [vid. 2022-04-21]. Dostupné z: https://cs.wikipedia.org/wiki/Address_Resolution_Protocol

[10] Wikipedia, t. f. e.: Unix time. [online]. rev. 15.dubna.2022. [vid. 2022-04-21]. Dostupné z: https://en.wikipedia.org/wiki/Unix_time

[11] Wikipedia, t. f. e.: User Datagram Protocol. [online]. rev. 24.srpna.2021. [vid. 2022-04-21]. Dostupné z: https://cs.wikipedia.org/wiki/User_Datagram_Protocol

[12] Wikipedia, t. f. e.: ICMP. [online]. rev. 29.července.2019. [vid. 2022-04-21]. Dostupné z: https://cs.wikipedia.org/wiki/ICMP

[13] Wikipedia, t. f. e.: Transmission Control Protocol. [online]. rev. 6.dubna.2022. [vid. 2022-04-21]. Dostupné z: https://cs.wikipedia.org/wiki/Transmission_Control_Protocol