VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Projektová dokumentace k projektu IPK

Varianta ZETA: Sniffer paketů

25. dubna 2021

Serhii Salatskyi (xsalat01)

Obsah

1 Úvod **2**

2 Spuštění programu **2**

3 Implementace **3**
3.1 Sestavení síťového adaptéru 3
3.2 Prace a analýza s pakety 3
3.2.1 Zpracování ARP paketů 3
3.2.2 Zpracování IP paketů 3
3.2.3 Přesnější zpracování zachýcenych paketů 4
3.3 Výpis výstupu 4

4 Testování **5**

1 Úvod

Zadáním projektu bylo navrhnout a naimplementovat síťový analyzátor v C, který bude schopný na určitém síťovém rozhraním zachytávat a filtrovat pakety podle protokolů `TCP`, `UDP`, `ICMP` či `ARP`.

Packet sniffer, nazývaný také paketový analyzátor sestává ze dvou hlavních částí. První je síťový adaptér, který připojuje program k existující síti. Druhá část uskutečňuje analýzu samotných zachycených paketů, v našem případě zjišťuje, kdy byl paket zachycen, IP adresu zdroje a cílů a také port zdroje a cílů.

2 Spuštění programu

Program je kompatibilní s linuxovými systémy (aplikace byla testována na systémech Fedora 32 a Ubuntu 20.04). K správné kompilací je vhodné disponovat překladačů `gcc 7.5.0` a vyšší. Rovněž je potřebný program `make`, testované na verzí `GNU Make 4.1`.

V složce projektu se nachází Makefile, který umožní projekt sestavit použitím:

```
$ make
```

Aplikace se spustí pomocí příkazu:

```
$ sudo ./ipk-sniffer {-h} [-i rozhraní|--interface rozhraní] {-p port}
                     {[--tcp|-t] [--udp|-u] [--arp] [--icmp]} {-n počet}
```

- `-h` (vypíše napovědu na standartní výstup [vypíše se i při zadání neplatného argumentu])
- `-i` nebo `--interface <rozhraní>` (právě jedno rozhraní, na kterém se bude poslouchat. Nebude-li tento parametr uveden, či bude-li uvedené jen -i bez hodnoty, vypíše se seznam aktivních rozhraní)
- `-p <port>` (bude filtrování paketů na daném rozhraní podle portu; nebude-li tento parametr uveden, uvažují se všechny porty; pokud je parametr uveden, může se daný port vyskytnout jak v source, tak v destination části)
- `-t` nebo `--tcp` (bude zobrazovat pouze TCP pakety)
- `-u` nebo `--udp` (bude zobrazovat pouze UDP pakety)
- `--icmp` (bude zobrazovat pouze ICMPv4 a ICMPv6 pakety)
- `--arp` (bude zobrazovat pouze ARP rámce).
- `-n <počet>` (určuje počet paketů, které se mají zobrazit; pokud není uvedeno, uvažujte zobrazení pouze jednoho paketu) argumenty mohou být v libovolném pořadí

Pokud nebudou konkrétní protokoly specifikovány, uvažují se k tisknutí všechny (tj. veškerý obsah, nehledě na protokol). Program je možné kdykoliv ukončit pomocí klaves CTRL + C.

Aplikace:

- V případě úspěchu vrátí hodnotu 0
- V případě chybných argumentů nebo selhání alokace paměti, skončí s návratovou hodnotou 1
- V případě selhání soucástí knihovny PCAP vrátí hodnotu -1.

3 Implementace

Program je implementován v jazyce C v souboru `ipk-sniffer.c`. Na začátku se do proměnných načítají vstupní argumenty uvedené v sekcí 2 pomocí funkce `parse_args()`.

3.1 Sestavení síťového adaptéru

Implementace síťového adaptéru připojující se na existující síť je v funkci `main()` využívají funkce knihovny `pcap.h`. Na začátku je třeba nastavit interface, ten je poskytnut uživatelům diky argumentu `-i` a je uložen do proměnné `char *interface`, případně je možné zobrazit seznam dostupných zařízení pokud tento parametr vynecháme.

Pak můžeme přiřadit zařízení masku podsítě `bpf_u_int32 pMask` a ip adresu `bpf_u_int32 pNet` prostřednictvím funkce `pcap_lookupnet()`. Spolecne se jí prodává i zařízení, na kterém pracujeme. V případě chyby se vypíše chybová hláška uložena v `errbuf`. Následně je možné otevřít zařízení k zachytávání paketů, k tomu slouží funkce `pcap_open_live()` a hodnota z funkce se uloží do `pcap_t *sniffer`. Hodnota 0 vypíná promiskuitní režim, ovšem i tak se může stát, že v konkrétních případech zůstane zapnutý (závisí i od platformy, na které se program používá).

Nyní je možné sestavit filtr `char filter` díky funkce `create_filter()`, která vrátí filtr podle uživatelem zadaných argumentů. V případě, že nebyl explicitně zadán požadavek na filtraci, implicitně se nastavuje filtr pro všechny pakety, ktere může zpracovat aplikace. Pomocí funkce `pcap_compile()` zkompilujeme náš adaptér a následně jej můžeme aplikovat pomocí `pcap_setfilter()`. Takto zostavaný adaptér nyní můžeme pomocí `pcap_loop()` uvést do "nekonečného cyklu", kde se zachytávavá pakety v počtu `pnum` a při každém zachyceném paketu se volá funkce `callback`.

3.2 Prace a analýza s pakety

Funkce `callback` zpracovává každý zachycený paket. Nejdřív se ukládá informace o timestampu paketu - tedy času, kdy byl paket zachycen, jehož hodnota je uložena v `pkthdr->ts.tv_sec` a mikro sekundy v `pkthdr->ts.tv_usec`; pomocí funkce localtime() se správně od Unixové epochy vypočítá cas.

Potom pomocí struktury `ether_header *p` se zjišťuje či daný paket používá IPv4, IPv6 nebo ARP. Podle typu etheru se výbere metoda, která bude zpracovavat zachycený paket (`process_arp_ether()` anebo `process_ip_ether()`). Při zpracování paketů je nutné někam ukládat data o něm. Pro tento účel byla vytvořena struktura `pckt_info`

```
struct pckt_info
{
    char src_addr[1025];   // obsahuje IP adresu zdroje
    char dest_addr[1025];  // obsahuje IP adresu cíle
    unsigned src_port;     // obsahuje zdrojový port
    unsigned dest_port;    // obsahuje zdrojový port
    unsigned proto_type;   // určuje číslo protokolu
};
```

3.2.1 Zpracování ARP paketů

Funkce `process_arp_ether()` pracuje z hlavičkou paketu podle Address Resolution protocolu. Pro zjištění IP adres zdroje a cíli se používá `struct ether_arp`.

```
struct ether_arp *arph = (struct ether_arp *)(buffer + 14);
```

Na základě získané hlavičky, je možné získat IP zdroje (`arph->arp_spa`) a cíle (`arph->arp_tpa`).

3.2.2 Zpracování IP paketů

Funkce `process_arp_ether()` pracuje z hlavičkou paketu podle Internet protocolu. Pomocí struktur `struct ip` | `struct ip6_hdr` resp. pro IPv4 a IPv6 pakety je možné zjistit IP adres zdroje a cíli.

```
struct ip* iph = (struct ip *)(buffer + sizeof(struct ether_header));
```

nebo

```
struct ip6_hdr* iph = (struct ip6_hdr *)(buffer + sizeof(struct ether_header));
```

Tyto hodnoty se dále posílají do funkce `host_name` příp. `host_nameIPv6`, které přetvorí je na smyslúplné řetězce, jejichž výstup se ukládá do `char *temp_src` a `char *temp_dest`. Po té operaci se jejich hodnoty ukládají do resp. `header.src_addr` a `header.dest_addr`. Dále lze ze struktury `struct ip` zjistit velikost IPv4 (protoze může být v rozmezí od 20 do 60 bajtů). Tato hodnota se nachází v `iph->ip_hl*4` (abychom získali pocet bajtů) a ukládá se do `int iphdr_len`. Při IPv6 je udávaná velikost stálých 40 bajtů.

Také jde upřesnit číslo protokolu paketu, které určuje jeho druh (bud' TCP, UDP, ICMP nebo ICMPv6). To číslo se ukládá do proměnné `header.proto_type`.

3.2.3 Přesnější zpracování zachýcenych paketů

Podle čísla `header.proto_type` výbíráme, jakým způsobem je třeba zpracovavat hlavičku paketu.
Pokud jde řeč o TCP paketu, tak můžeme pomocí struktury `struct tcphdr *tcph` zjistit port zdroje a cíli:

```
packet.src_port = ntohs(tcph->th_sport);
packet.dest_port = ntohs(tcph->th_dport);
```

Jde to podobně udělat i pro UDP paket pomocí struktury `struct udphdr *udph`:

```
packet.src_port = ntohs(udph->uh_sport);
packet.dest_port = ntohs(udph->uh_dport);
```

ICMP a ICMPv6 hlavičky nedefinují porty, proto není třeba zpracovavat je.

3.3 Výpis výstupu

Výstup dat zajistuje funkce `print_data()`, která je volána z funkce callback. Data vypisuje na standardní výstup podobně jako je v zadání a obdobně s opensource softwarem `Wireshark`. Výpis je uskutečňován po řádcích v cyklu `for`.

```
2021-04-25T21:14:12.278778+02:00 192.168.0.108:44282 > 147.229.9.21:443, length: 66
------------------------------------------------------------------------
0x0000 74 da 88 23 a9 92 48 89  e7 ff a8 ca 08 00 45 00  t..#..H. ......E.
0x0010 00 34 97 6f 40 00 40 06  45 46 c0 a8 00 6c 93 e5  .4.o@.@. EF...l..
0x0020 09 15 ac fa 01 bb c1 0a  fc 3b 42 b7 8e 98 80 11  ........ .;B.....
0x0030 01 f5 57 23 00 00 01 01  08 0a 11 f5 8f c7 a8 13  ..W#.... ........
0x0040 38 79                                             8y
------------------------------------------------------------------------
```

Na prvním řádku je timestamp paketu ve formátu dle RFC3339, následuje IP adresa a port zdroje, za znakem > se nachází IP adresa a port cíle. Dále je ukazána délka paketu v bajtech.

Následující řádek slouží jako oddělení mezi informaci o zdroje a cíli a obsahem paketu.

První sloupec značí offset vypsaných bajtů. Ve druhém sloupci se nachází maximálně 16 bajtů, které se píší po dvojicích a pro lepší přehlednost je sloupec po osmi bajtech obohacen o jednu mezeru navíc. Ve třetím sloupci je výpis stejných bajtů, což ve druhém sloupci avšak jsou vypsány tisknutelnou znaky ASCII, netisknutelné jsou pomocí funkce `isprint()` nahrazeny tečkou.

Hlavička paketu je od zbývajících dat odděleních mezerou. Poslední řádek slouží opět jako orámování a oddělení od ostatních zachycených paketů.

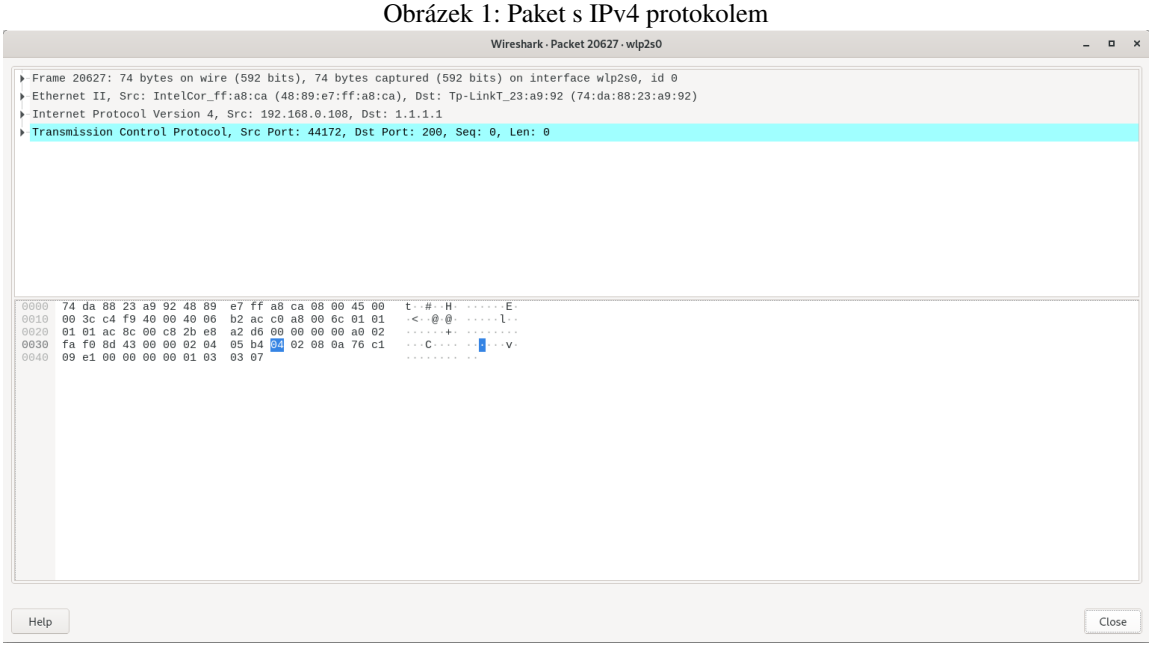
4 Testování

Testování probíhalo na na systémech Fedora32 a Ubuntu 20.4, pomocí souběžně spuštěného open source softwaru `Wireshark` a aplikací `ipk-sniffer`.

TCP paket odeslaný pomocí příkazu curl:

```
[ssalatsk@localhost proj2]$ curl 1.1.1.1:200
^C
[ssalatsk@localhost proj2]$
```

Obrázek 1: Paket s IPv4 protokolem



Obrázek 2: IPv4 paket zachycený v Wireshark

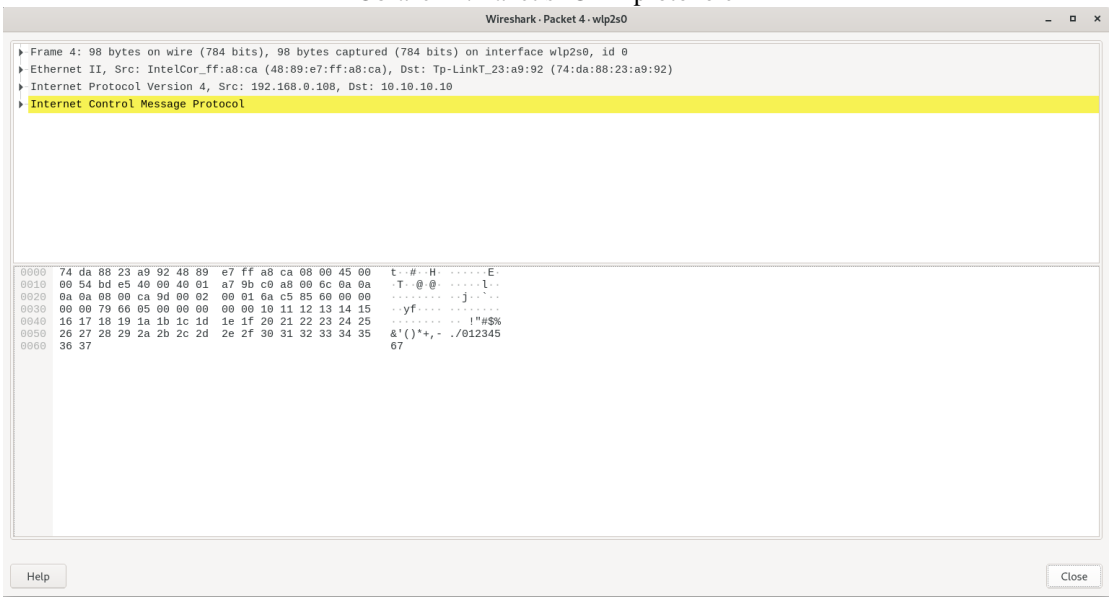
```
[ssalatsk@localhost proj2]$ sudo ./ipk-sniffer -i wlp2s0 --tcp -p 200
filter: tcp port 200
2021-04-25T21:56:29.175032+02:00 192.168.0.108:44172 > 1.1.1.1:200, length: 74
-------------------------------------------------------------------------
0x0000 74 da 88 23 a9 92 48 89  e7 ff a8 ca 08 00 45 00  t..#..H. ......E.
0x0010 00 3c c4 f9 40 00 40 06  b2 ac c0 a8 00 6c 01 01  .<..@.@. .....l..
0x0020 01 01 ac 8c 00 c8 2b e8  a2 d6 00 00 00 00 a0 02  ......+. ........
0x0030 fa f0 8d 43 00 00 02 04  05 b4 04 02 08 0a 76 c1  ...C.... ......v.
0x0040 09 e1 00 00 00 00 01 03  03 07                    ........ ..
-------------------------------------------------------------------------
```

Obrázek 3: IPv4 paket zachycený aplikací ipk-sniffer

ICMP paket odeslaný pomocí příkazu ping:

```
[ssalatsk@localhost proj2]$ ping 10.10.10.10
PING 10.10.10.10 (10.10.10.10) 56(84) bytes of data.
^C
--- 10.10.10.10 ping statistics ---
22 packets transmitted, 0 received, 100% packet loss, time 21509ms
```

Obrázek 4: Paket s ICMP protokolem

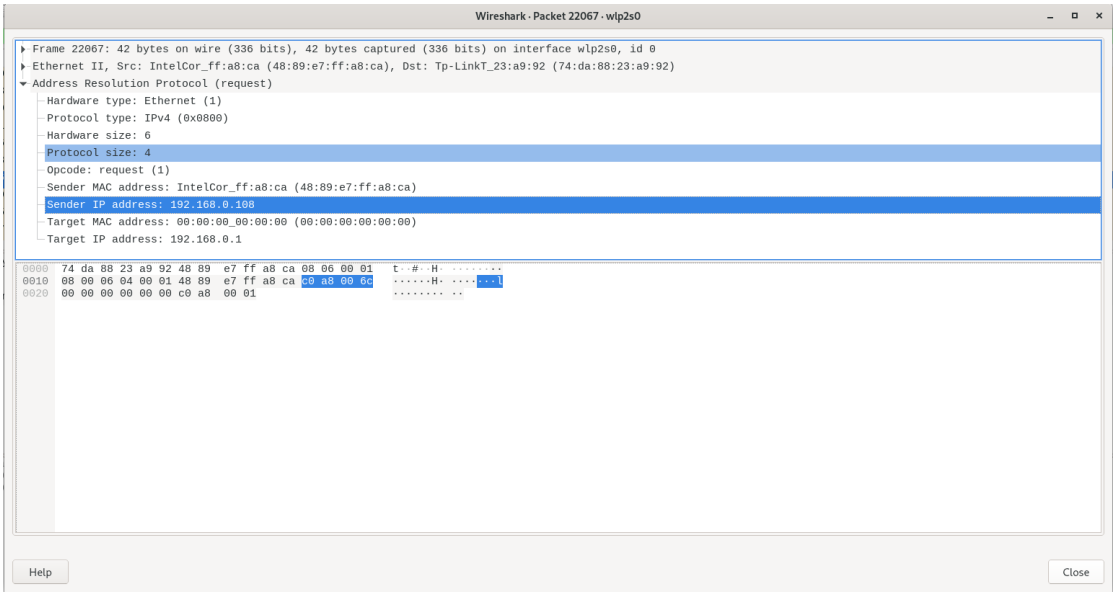


Obrázek 5: ICMP paket zachycený v Wireshark

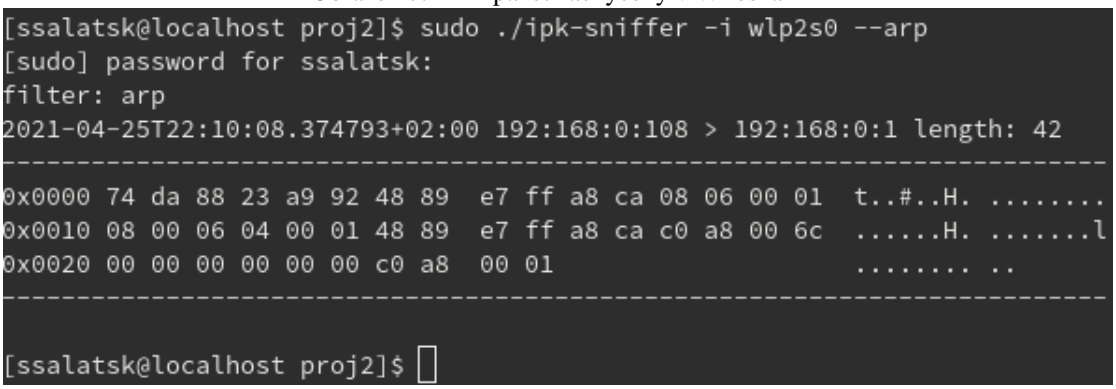
```
[ssalatsk@localhost proj2]$ sudo ./ipk-sniffer -i wlp2s0 --icmp
[sudo] password for ssalatsk:
filter: icmp or icmp6
2021-04-25T21:39:33.622934+02:00 192.168.0.108 > 10.10.10.10 length: 98
-------------------------------------------------------------------------
0x0000 74 da 88 23 a9 92 48 89  e7 ff a8 ca 08 00 45 00  t..#..H. ......E.
0x0010 00 54 d6 a6 40 00 40 01  8e da c0 a8 00 6c 0a 0a  .T..@.@. .....l..
0x0020 0a 0a 08 00 23 78 00 02  00 0c 75 c5 85 60 00 00  ....#x.. ..u..`..
0x0030 00 00 11 81 09 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: ICMP paket zachycený aplikací ipk-sniffer

Zachycení ARP paketu:



Obrázek 7: ARP paket zachycený v Wireshark

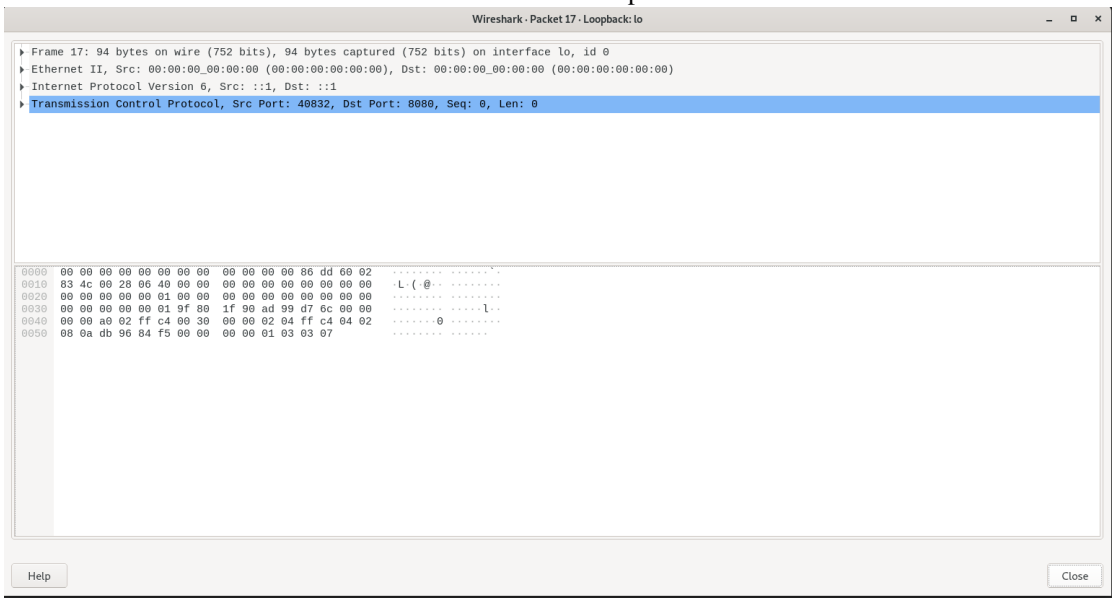


Obrázek 8: ARP paket zachycený aplikaci ipk-sniffer

Paket s IPv6 odeslaný pomocí příkazu curl:

```
[ssalatsk@localhost proj2]$ curl -g -6 "http://[::1]:8080/"
curl: (7) Failed to connect to ::1 port 8080: Connection refused
[ssalatsk@localhost proj2]$
```

Obrázek 9: Paket s IPv6 protokolem



Obrázek 10: Paket zachycený v Wireshark

```
[ssalatsk@localhost proj2]$ sudo ./ipk-sniffer -i lo --tcp -n 2
filter: tcp
2021-04-25T23:03:40.776552+02:00 ::1:40832 > ::1:8080, length: 94
------------------------------------------------------------------------------
0x0000 00 00 00 00 00 00 00 00  00 00 00 00 86 dd 60 02  ........ ......`.
0x0010 83 4c 00 28 06 40 00 00  00 00 00 00 00 00 00 00  .L.(.@.. ........
0x0020 00 00 00 00 00 01 00 00  00 00 00 00 00 00 00 00  ........ ........
0x0030 00 00 00 00 00 01 9f 80  1f 90 ad 99 d7 6c 00 00  ........ .....l..
0x0040 00 00 a0 02 ff c4 00 30  00 00 02 04 ff c4 04 02  .......0 ........
0x0050 08 0a db 96 84 f5 00 00  00 00 01 03 03 07        ........ ......
------------------------------------------------------------------------------
```

Obrázek 11: Paket zachycený aplikací ipk-sniffer