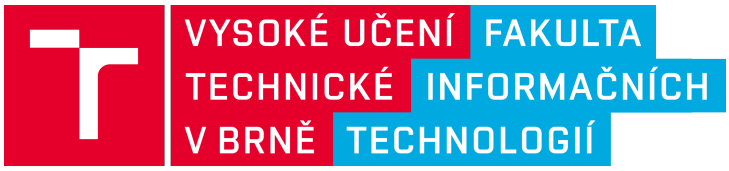


IPK – 2. Projekt, varianta ZETA
Sniffer paketů

Dominik Vágner,
xvagne10

23. dubna 2022

Obsah

1 Úvod

Zadáním projektu bylo navrhnout a implementovat síťový analyzátor, který bude schoný na zvoleném síťovém rozhraní zachytávat a filtrovat pakety určitých typů.

Paketový sniffer umožnuje zachytávat pouze na zařízeních ethernetového typu. Podporovanými protokoly komunikace, které můžeme filtrovat, jsou `ARP`, `TCP`, `UDP` a `ICMP`. Pro protokoly které se nachází v síťové nebo transportní vrstvě je podpora pro přenos (nebo jejich respektivní verze) pomocí `IPv4` nebo `IPv6`. Je také možné filtrovat dle portů transportní vrstvy nebo zadat počet odchytávaných paketů.

Informace o zachycených paketech jsou poté vypisovány na standardní výstup `stdout`. Mezi vypisované informace patří třeba příchozí a odchozí MAC adresy, IP adresy, porty, časové razítko, specifické informace pro některé z protokolů a hexadecimální výpis všech dat v paketu.

2 Problematika

Problematiku projektu můžeme rozdělit na dvě hlavní části. První je připojení na síťový rozhraní a jeho nastavení pro zachytávaní. V druhé části poté analyzujeme přijmutý paket a podle získaných informací vypysujeme data v korektním zápisu.

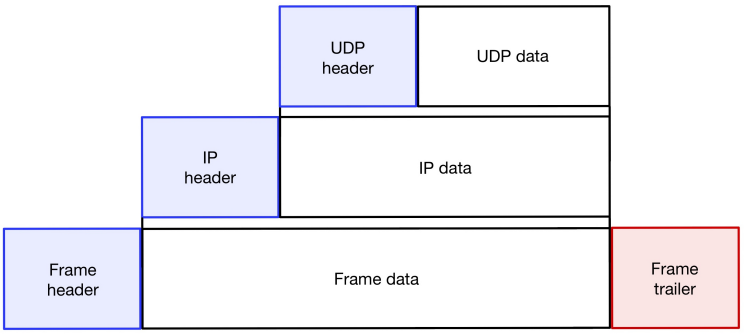
2.1 Zahájení odchytu

Před zahájením naslouchání musíme zpracovat argumenty programu pro informace o tom jak nastavit filtr pro zachycené pakety a na jakém rozhraní naslouchat. Poté ověříme zda rozhraní je ethernetového typu. Následně sestavíme filtr paketů z argumentů programu a zahajíme zachytávání paketů na určeném rozhraní pro specifikovaný počet paketů.

Abychom mohli na rozhraní naslouchat tak musí být síťová karta použita v promiskuitním módu. Tento mód znamená že všechny síťové adaptéry můžou vidět přenášené pakety [1].

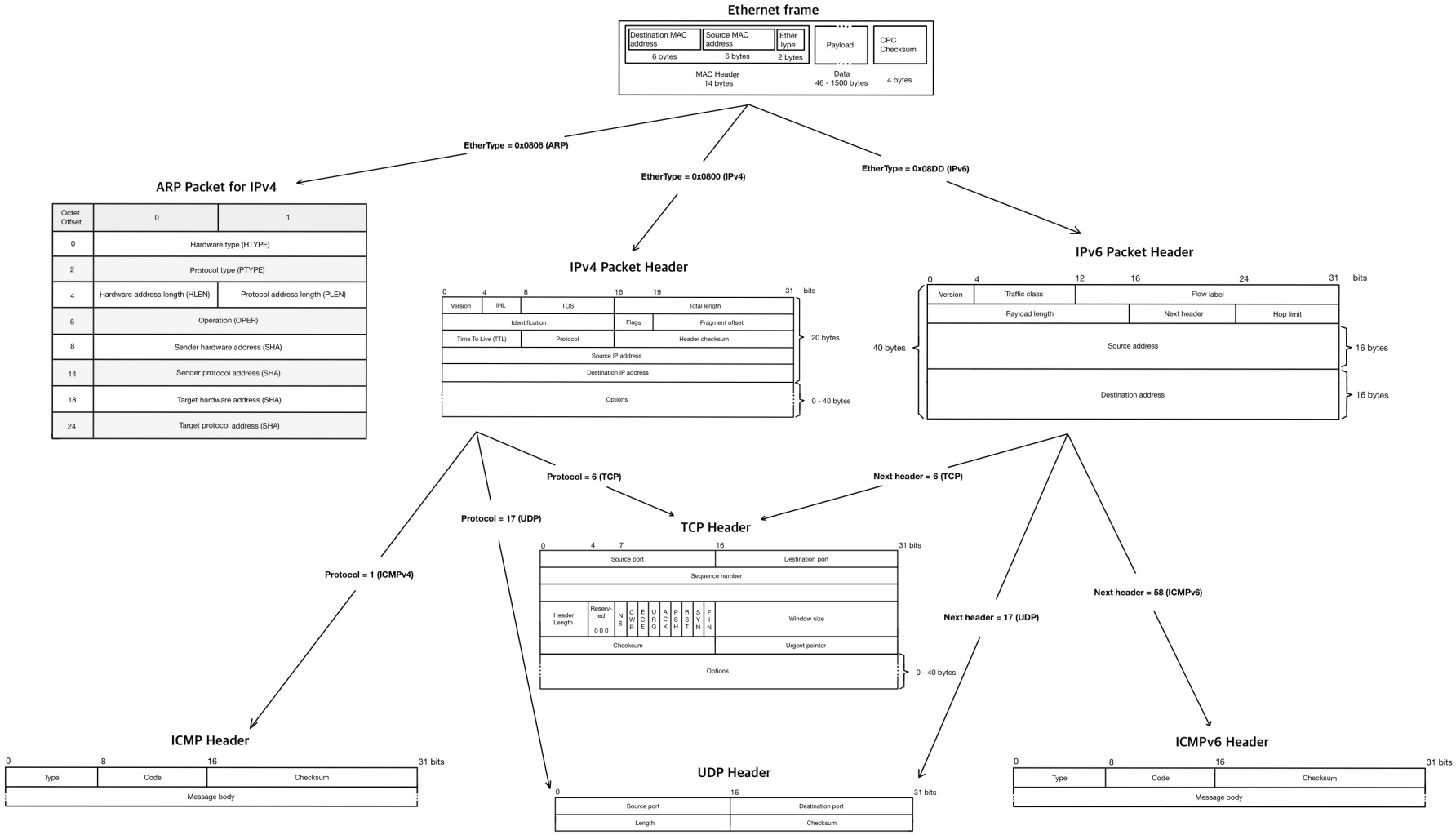
2.2 Analýza přijmutých informací

Při zpracovávání informací nejvíce narážíme na zapouzdřování datagramů. Zachycený ethernetový rámec má za hlavičkou payload s typem dat určeným v hlavičce. Podobně je to tak i s protokoly na síťové vrstvě a trasnportní vrstvě. Postup pro určení správné hlavičky protokolu (z podporovaných) pro výpis je zobrazen obrázkem č. 2.



Obrázek 1: Příklad zapouzdření ukázán na UDP datagramu [2].

Dalším problémem jsou také protokoly s různou délkou, např. IPv4. U takových protokolů musíme vypočítat aktualní délku podle informací uložených v jejich hlavičkách.



Obrázek 2: Diagram postupu určování protokolu [3–11]

.

3 Implementace

Projekt byl implementován v jazyce C++. Je složen ze dvou souborů, `ipk-sniffer.c` s implementacemi funkcí ze kterých se program skládá a `ipk-sniffer.h` obsahující seznam použitých knihoven, definice vlastních struktur, deklarace funkcí a komentáře pro dané funkce. Zdroje ze kterých byly získány informace o tom jaka implementovat některé části tohoto projektu jsou ozdrojovány v kódu.

3.1 Časti programu

Hlavní funkce programu

Ve vstupní/hlavní funkci programu `main()` se nejdříve zavolá funkce `parse_arguments()` pro zpracování argumentů. Ve struktuře pro argumenty zkontolujeme zda je specifikované rozhraní na kterém máme naslouchat a pokud není tak zjistíme vše dostupné rozhraní a vypíšeme je. Máme-li zadané rozhraní tak ověříme jeho existenci. Následně se pokusíme otevřít rozhraní pro naslouchání a otestujeme je-li ethernetového typu. Poté pomocí funkce `fill_filter()` naplníme string podle argumentů programu a tento řetězec aplikujeme na otevřené rozhraní. Nakonec zahájíme odchyt specifikovaného množství paketů. Každý zachycený paket pošleme do funkce pro analýzu paketů `callback_handler()`.

Při každém neúspěšném ověření vypýšeme error vrácený funkcemi z knihovy `libpcap` pro daný úkon. Tyto hlášky vypíšeme na `stdout` a ukončíme program.

Většina logiky spojené s prací s rozhraními a zachycenými pakety je implementována pomocí knihovny `libpcap`. Od správců této knihovny je vytvořen také velice důkladný a srozumitelný tutoriál [12].

Zpracování argumentů

Pro agrumenty máme v programu vytvořenou strukturu `ARG_VALUES` s příznaky pro všechny argumenty a jejich hodnoty. V funkci pro zpracování argumentů `parse_arguments()` tuto strukturu naplníme pomocí knihovny `getopt` [13] a následně ji vratíme. Tento výsledek si poté uložíme ve funkci `main()` pro budoucí práci s argumenty.

Analýza paketu

Ve funkci pro analýzu paketů postupujeme podle obrázku č. 2. Paket dostaneme pouze jako pole unsigned charakterů a od začátku poté zjišťujeme jaké hlavičky protokolů jsou za sebou. Jako první máme vždy ethernet frame. Poté postupujeme pomocí informací v daných protokolech o následujících headerů. Pokud narazíme na protokol, který už za sebou nemá další informace tak zavoláme funkci pro výpis daného protokolu.

Výpis

Vetšina funkcí v tomto projektu slouží k výpisu získaných informací. Máme funkce pro výpis základních infromací o paketu, MAC/IPv4/IPv6 adres, informací specifických pro každý podporovaný protokol a pro výpis celého paketu v hexadecimálním formátu. Ukázky o tom jaké informace z různých protokolů vypysujeme můžeme najít v sekci o testování 4.

- `print_base()` – Výpis časového razítka, zdrojové a cílové MAC adresy a délky rámce.
- `print_arp()` – Výpis specifických informací z ARP headeru.
- `print_ip4_addr()` – Výpis zdrojové a cílové IPv4 adresy.
- `print_ip6_addr()` – Výpis zdrojové a cílové IPv6 adresy.
- `print_icmp()` – Výpis specifických informací z ICMPv4 headeru.
- `print_icmpv6()` – Výpis specifických informací z ICMPv6 headeru.
- `print_tcp()` – Výpis specifických informací z TCP headeru.

- `print_udp()` – Výpis specifických informací z UDP headeru.
- `print_payload()` – Výpis celého paketu v hexadecimálním zápisu.

3.2 Knihovny

- `pcap.h` – Hlavní knihovna pro zachytávání paketů a pro práci s rozhraními.
- `arpa/inet.h` – Funkce pro obrácení unsigned short a long proměnných.
- `netinet/ether.h` – Struktura pro ethernet header a makra k ní vázané.
- `netinet/tcp.h` – Struktura pro TCP header a makra k ní vázané.
- `netinet/udp.h` – Struktura pro UDP header a makra k ní vázané.
- `netinet/ip_icmp.h` – Struktura pro ICMPv4 header a makra k ní vázané.
- `netinet/ip.h` – Struktura pro IPv4 header a makra k ní vázané.
- `netinet/ip6.h` – Struktura pro IPv6 header a makra k ní vázané.
- `netinet/icmp6.h` – Struktura pro ICMPv6 header a makra k ní vázané.
- `iostream`, `iomanip` – C++ knihovny pro výpis pomocí streamů a jejich formátování.
- `getopt.h` – Funkce k zpracování argumentů programu.
- `stdio.h` – Pro `printf()` funkci.
- `string.h` – Přidání string typu.
- `signal.h` – Zpracování interrupt signalů pro korektní ukončení.
- `ctime`, `locale` – Pro výpis časového razítka.

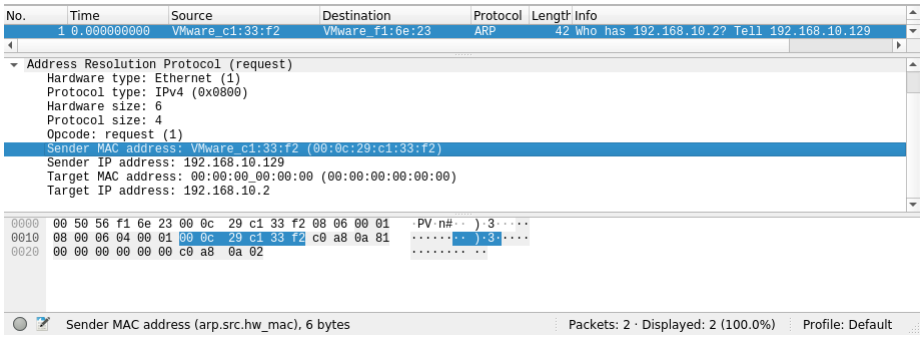
4 Testování

Projekt byl testován na referenční m virtuálním stroji s `Ubuntu 20.04 LTS` dodané k projektu. Testování spočívalo v porovnávání výstupu projektu a open source programu `Wireshark`[1]. Otestované byly všechny podporované protokoly. Pro simulování/generování vlastní komunikace byly použity programy `netcat`[2] (přesněji jeho `OpenBSD` přepis, který podporuje IPv6), `ping` a `ping6`.

[1] Jeden z nejznámějších síťových analyzátorů.: `https://www.wireshark.org/`

[2] Netcat je UNIX program, který čte a zapisuje data přes síťové spojení, pomocí TCP nebo UDP protokolu.: `https://salsa.debian.org/debian/netcat-openbsd`

ARP



Obrázek 3: ARP paket zachycený ve `Wiresharku`

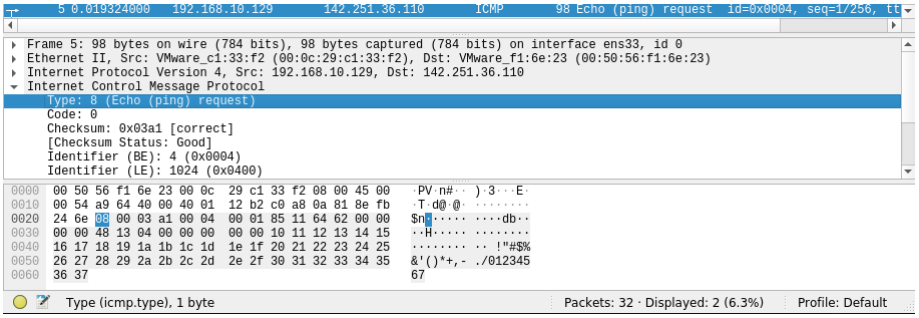
```
student@student-vm:/mnt/hgfs/code$ sudo ./ipk-sniffer -i ens33 --arp
timestamp: 2022-04-23T17:05:14.625+02:00
src MAC: 00:0c:29:c1:33:f2
dst MAC: 00:50:56:f1:6e:23
frame length: 42 bytes
src IP: 192.168.10.129
dst IP: 192.168.10.2
protocol: ARP

ARP Specifics:
sender MAC: 00:0c:29:c1:33:f2
target MAC: 00:00:00:00:00:00
operation: 1 (request)

0x0000:  00 50 56 f1 6e 23 00 0c  29 c1 33 f2 08 06 00 01  .PV.n#.. ).3.....
0x0010:  08 00 06 04 00 01 00 0c  29 c1 33 f2 c0 a8 0a 81  ........ ).3.....
0x0020:  00 00 00 00 00 00 c0 a8  0a 02                    ........ ..
```

Obrázek 4: ARP paket zachycený přes `ipk-sniffer`

ICMPv4



Obrázek 5: ICMPv4 paket zachycený ve `Wiresharku`

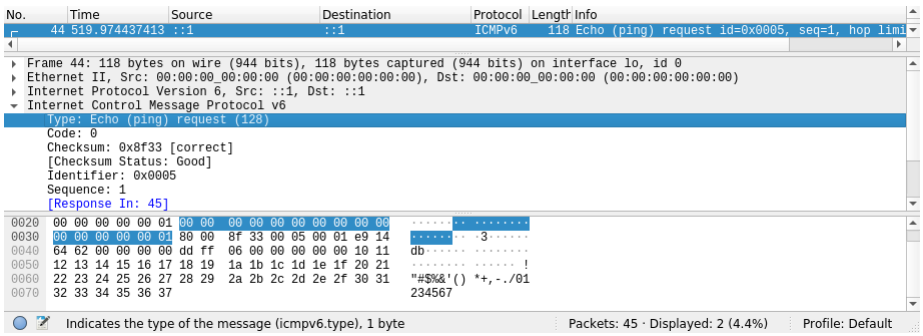
```
student@student-vm:/mnt/hgfs/code$ sudo ./ipk-sniffer -i ens33 --icmp
timestamp: 2022-04-23T16:47:33.267+02:00
src MAC: 00:0c:29:c1:33:f2
dst MAC: 00:50:56:f1:6e:23
frame length: 98 bytes
src IP: 192.168.10.129
dst IP: 142.251.36.110
protocol: ICMP

ICMP Specifics:
Type: 8
Code: 0

0x0000:  00 50 56 f1 6e 23 00 0c  29 c1 33 f2 08 00 45 00  .PV.n#.. ).3...E.
0x0010:  00 54 a9 64 40 00 40 01  12 b2 c0 a8 0a 81 8e fb  .T.d@.@. ........
0x0020:  24 6e 08 00 03 a1 00 04  00 01 85 11 64 62 00 00  $n...... ....db..
0x0030:  00 00 48 13 04 00 00 00  00 00 10 11 12 13 14 15  ..H..... ........
0x0040:  16 17 18 19 1a 1b 1c 1d  1e 1f 20 21 22 23 24 25  ........ .. !"#$%
0x0050:  26 27 28 29 2a 2b 2c 2d  2e 2f 30 31 32 33 34 35  &'()*+,- ./012345
0x0060:  36 37                                            67
```

Obrázek 6: ICMPv4 paket zachycený přes `ipk-sniffer`

ICMPv6



Obrázek 7: ICMPv6 paket zachycený ve `Wiresharku`

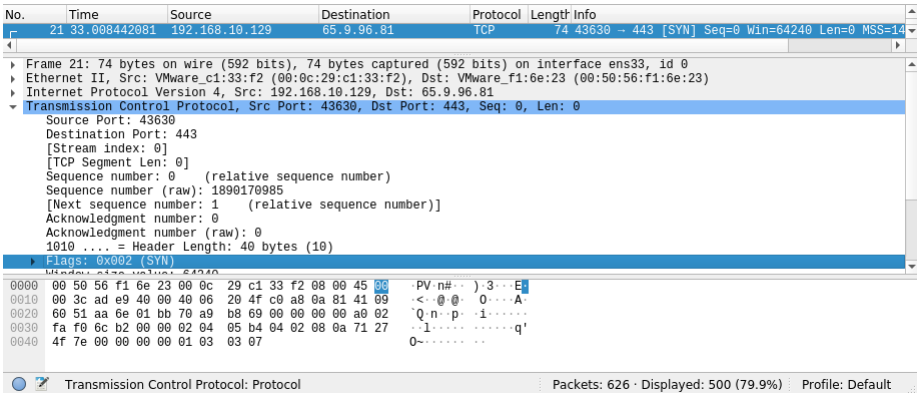
```
student@student-vm:/mnt/hgfs/code$ sudo ./ipk-sniffer -i lo --icmp
timestamp: 2022-04-23T17:02:01.466+02:00
src MAC: 00:00:00:00:00:00
dst MAC: 00:00:00:00:00:00
frame length: 118 bytes
src IP: 0000.0000.0000.0000.0000.0000.0000.0001
dst IP: 0000.0000.0000.0000.0000.0000.0000.0001
protocol: ICMPv6

ICMPv6 Specifics:
Type: 128
Code: 0

0x0000:  00 00 00 00 00 00 00 00  00 00 00 00 86 dd 60 0d  ........ ......`.
0x0010:  7d c6 00 40 3a 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 80 00  8f 33 00 05 00 01 e9 14  ........ .3......
0x0040:  64 62 00 00 00 00 dd ff  06 00 00 00 00 00 10 11  db...... ........
0x0050:  12 13 14 15 16 17 18 19  1a 1b 1c 1d 1e 1f 20 21  ........ ...... !
0x0060:  22 23 24 25 26 27 28 29  2a 2b 2c 2d 2e 2f 30 31  "#$%&'() *+,-./01
0x0070:  32 33 34 35 36 37                                234567
```

Obrázek 8: ICMPv6 paket zachycený přes `ipk-sniffer`

TCP



Obrázek 9: TCP paket zachycený ve `Wiresharku`

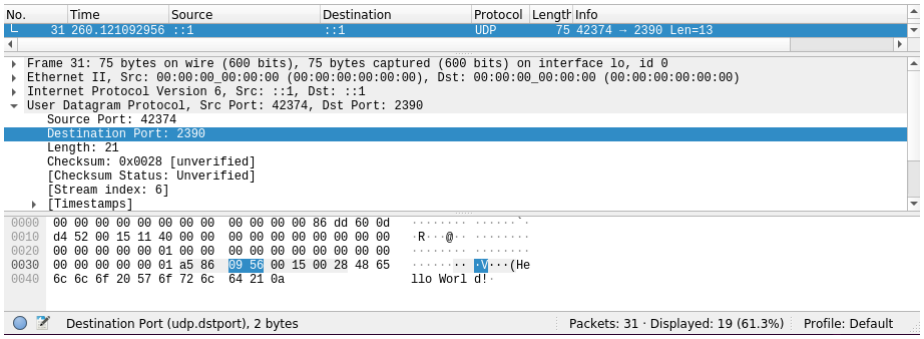
```
student@student-vm:/mnt/hgfs/code$ sudo ./ipk-sniffer -i ens33 -t
timestamp: 2022-04-23T16:50:18.644+02:00
src MAC: 00:0c:29:c1:33:f2
dst MAC: 00:50:56:f1:6e:23
frame length: 74 bytes
src IP: 192.168.10.129
dst IP: 65.9.96.81
src port: 43630
dst port: 443
protocol: TCP

TCP Specifics:
Sequence number: 1890170985
Acknowledgment number: 0
NS: 0
CWR: 0
ECE: 0
URG: 0
ACK: 0
PSH: 0
RST: 0
SYN: 1
FIN: 0

0x0000:  00 50 56 f1 6e 23 00 0c  29 c1 33 f2 08 00 45 00  .PV.n#.. ).3...E.
0x0010:  00 3c ad e9 40 00 40 06  20 4f c0 a8 0a 81 41 09  .<..@.@.  O....A.
0x0020:  60 51 aa 6e 01 bb 70 a9  b8 69 00 00 00 00 a0 02  `Q.n..p. .i......
0x0030:  fa f0 6c b2 00 00 02 04  05 b4 04 02 08 0a 71 27  ..l..... ......q'
0x0040:  4f 7e 00 00 00 00 01 03  03 07                    O~...... ..
```

Obrázek 10: TCP paket zachycený přes `ipk-sniffer`

UDP



Obrázek 11: UDP paket zachycený ve `Wiresharku`

```
student@student-vm:/mnt/hgfs/code$ sudo ./ipk-sniffer -i lo -u -p 2390
timestamp: 2022-04-23T16:57:41.613+02:00
src MAC: 00:00:00:00:00:00
dst MAC: 00:00:00:00:00:00
frame length: 75 bytes
src IP: 0000.0000.0000.0000.0000.0000.0000.0001
dst IP: 0000.0000.0000.0000.0000.0000.0000.0001
src port: 42374
dst port: 2390
protocol: UDP

0x0000:  00 00 00 00 00 00 00 00  00 00 00 00 86 dd 60 0d  ........ ......`.
0x0010:  d4 52 00 15 11 40 00 00  00 00 00 00 00 00 00 00  .R...@.. ........
0x0020:  00 00 00 00 00 01 00 00  00 00 00 00 00 00 00 00  ........ ........
0x0030:  00 00 00 00 00 01 a5 86  09 56 00 15 00 28 48 65  ........ .V...(He
0x0040:  6c 6c 6f 20 57 6f 72 6c  64 21 0a                 llo Worl d!.
```

Obrázek 12: UDP paket zachycený přes `ipk-sniffer`

5 Závěr

V projektu se mi povedlo implementovat vše požadované a navíc i výpis některých specifických informací z jednotlivých protokolů. Vyzkoušel jsem si poprvé C++ a práci s knihovnou Libpcap. Více jsem se také naučil o zapouzdřování datagramů mezi ISO/OSI vrstvami a jednotlivých námi podporovaných protokolech. Hlavně jak jsou strukturované jejich hlavičky. Projekt mi přišel jako jeden z těch zajimavějších/zábavnějších tento semestr.

Literatura

[1] Wikipedia. Promiscuous mode — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Promiscuous%20mode&oldid=1072823480, 2022. [Online; accessed 22-April-2022].

[2] V. Veselý. Transportní vrstva. *[Univerzitní přednáška]*, 2022.

[3] Wikipedia. Address Resolution Protocol — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Address%20Resolution%20Protocol&oldid=1081211967, 2022. [Online; accessed 22-April-2022].

[4] Wikipedia. Ethernet frame — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Ethernet%20frame&oldid=1080405638, 2022. [Online; accessed 22-April-2022].

[5] Wikipedia. Internet Control Message Protocol — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Internet%20Control%20Message%20Protocol&oldid=1081447774, 2022. [Online; accessed 22-April-2022].

[6] Wikipedia. Internet Control Message Protocol for IPv6 — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Internet%20Control%20Message%20Protocol%20for%20IPv6&oldid=1062617837, 2022. [Online; accessed 22-April-2022].

[7] Wikipedia. IPv4 — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=IPv4&oldid=1081194704, 2022. [Online; accessed 22-April-2022].

[8] Wikipedia. IPv6 — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=IPv6&oldid=1080648470, 2022. [Online; accessed 22-April-2022].

[9] Wikipedia. Transmission Control Protocol — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Transmission%20Control%20Protocol&oldid=1083491738, 2022. [Online; accessed 22-April-2022].

[10] Wikipedia. User Datagram Protocol — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=User%20Datagram%20Protocol&oldid=1079061202, 2022. [Online; accessed 22-April-2022].

[11] Internet Assigned Numbers Authority. Protocol numbers. https://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml, Apr 2021.

[12] Tim Carstens. Programming with pcap. https://www.tcpdump.org/pcap.html, 2002. [Online; accessed 23-April-2022].

[13] Lei Mao. Parsing argument using getopt in c/c++. https://leimao.github.io/blog/Argument-Parser-Getopt-C/, 2019. [Online; accessed 23-April-2022].