

IPK Projekt 2: varianta Zeta

Dokumentácia

Jakub Duda, xdudaj02

Obsah

1. Úvod	3
2. Funkcionalita	4
3. Implementácia	5
3.1. Všeobecné informácie	5
3.2. Globálne definície	5
3.2.1. Štruktúry hlavičiek	5
3.2.2. Ďalšie globálne definície	5
3.3. Činnosť programu	5
3.3.1. Vyhodnotenie možností príkazového riadku	5
3.3.2. Generovanie reťazca pre sniffovací filter	6
3.3.3. Založenie spojenia so sniffovaným zariadením	6
3.3.4. Sniffovacia slučka	6
3. Testovanie	8
4. Záver	9
5. Bibliografia	10

1. Úvod

Sniffer pakiet je program slúžiaci na monitorovanie sieťovej premávky. Tento program zachytáva pakety v drôtových aj bezdrôtových sieťach a následne skúma ich obsah. Sniffer v promiskuitnom móde je schopný zachytávať aj pakety, ktoré nie sú určené pre dané zariadenie. Takýto sniffer zachytáva obrovské množstvo pakiet, a preto je dôležité použitie tzv. filtrov. Filtre slúžia na odchytenie len takých paketov, o ktoré máme záujem.¹

Jedným z rozhraní, na ktorom je možné prevádzkovať sniffovanie paketov je Ethernet. Ethernet paket má na prvej vrstve Ethernet hlavičku a na ňom sa zvyčajne vyskytujú ďalšie protokoly s konkrétnou funkciou.

ARP je komunikačný protokol používaný na zistenie fyzickej adresy prijímateľa (adresa MAC), o ktorom je známa jeho adresa sieťovej vrstvy, zvyčajne IPv4 adresa.²

Internetový protokol (IP), je komunikačný protokol sieťovej vrstvy používaný na výmenu dát sieťou vo forme paketov. Nad IP protokolom sa väčšinou vyskytuje ďalší protokol s konkrétnou funkciou.³

TCP protokol, protokol riadenia prenosu, slúži na vytvorenie spojenia v sieti. Tento protokol zaručuje, že dáta odoslané z jedného konca spojenia budú prijaté na druhej strane spojenia v rovnakom poradí a bez chýbajúcich častí. Využíva sa v spojení s IP protokolom a vytvára strednú vrstvu medzi ním a aplikáciami nad ním. Tento protokol využíva porty.⁴

UDP protokol, používateľský datagramový protokol, sa taktiež zvyčajne nachádza nad IP protokolom. V porovnaní s TCP protokolom je menej spoľahlivý, nezaručuje, že prenášaný paket sa nestratí, že sa nezmení poradie paketov, ani že sa niektorý paket nedoručí viackrát. Vďaka tomu je však rýchlejší a preto je pri jednoduchších účeloch viac efektívny.⁵

ICMP protokol je dôležitým protokolom vo sfére Internetu. Používajú ho operačné systémy počítačov v sieti na odosielanie chybových správ, napríklad na oznámenie, že požadovaná služba nie je dostupná. Používa ho napríklad program *ping*.⁶

¹ What is a Packet Sniffer? (23.4.2021). www.kaspersky.com.

<https://www.kaspersky.com/resource-center/definitions/what-is-a-packet-sniffer>

² Address Resolution Protocol. (23.4.2021). Wikipedia.

https://en.wikipedia.org/wiki/Address_Resolution_Protocol

³ Internet Protocol. (23.4.2021). Wikipedia. https://en.wikipedia.org/wiki/Internet_Protocol

⁴ Transmission Control Protocol. (23.4.2021). Wikipedia.

https://en.wikipedia.org/wiki/Transmission_Control_Protocol

⁵ User Datagram Protocol. (23.4.2021). Wikipedia. https://en.wikipedia.org/wiki/User_Datagram_Protocol

⁶ Internet Control Message Protocol. (23.4.2021). Wikipedia.

https://en.wikipedia.org/wiki/Internet_Control_Message_Protocol

2. Funkcionalita

Program ipk-sniffer dokáže na rozhraní Ethernet sniffovať pakety. Odchytáva ľubovoľné kombinácie paketov typu TCP, UDP, ARP a ICMP a podporuje IP protokoly verzie 4 aj verzie 6. Program je potrebné spúšťať s právami superpoužívateľa.

Sniffer ponúka možnosť pomocou prepínačov na príkazovom riadku špecifikovať rozhranie, počet paketov, číslo portu alebo typ paketov, ktoré sa budú sniffovať. Pomocou prepínaču -i alebo --interface sa nastavuje rozhranie, na ktorom bude prebiehať sniffovanie. Pre absenciú tohto prepínaču alebo argumentu pre tento prepínač, program vypíše zoznam dostupných rozhraní. Pomocou prepínaču -p je možné špecifikovať číslo portu, ktorý môže byť na strane odosielateľa alebo na strane prijímateľa. Pri absencií sú sniffované pakety bez ohľadu na číslo portu. Prepínač -n slúži na nastavenie počtu paketov, ktoré budú vysniffované, pri absencii je sniffovaný jeden paket. Prepínače --tcp, resp. -t, --udp, resp. -u, --arp a --icmp slúžia na určenie typu sniffovaných paketov. Pri použití takéhoto prepínača bude sniffovaný len ten typ paketov, ktorý bol špecifikovaný prepínačom. Je možné použiť nula až všetky z týchto prepínačov.

Výpis obsahu paketu prebieha v nasledujúcom formáte:

hlavička:

čas vo formáte RFC3339 IP (resp. MAC) : port > IP (resp. MAC) : port, length dĺžka v bytoch

obsah:

offset	hexadecimálne dáta	ASCII dáta
0x0000	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0010	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

3. Implementácia

3.1. Všeobecné informácie

Program *ipk-sniffer* je implementovaný v jazyku C++. Využíva ale len niekoľko vymožeností tohto jazyka a zvyšný kód je napísaný v jazyku C.

Program využíva viacero importovaných knižníc. Medzi najzaujímavejšie patrí knižnica `getopt.h` využívaná na spracovanie argumentov, knižnice `sstream` a `iomanip` využívané pri práci s reťazcami, knižnica `csignal` využitá na ošetrovanie prerušenia činnosti programu signálom `interrupt`, knižnice `netinet/in.h`, `netinet/ether.h` a `arpa/inet.h` obašujúce konštanty a funkcie využívané pri spracovávaní dát jednotlivých paketov a v neposlednom rade knižnica `pcap.h` obsahujúca funkcie potrebné pre manipuláciu so sieťovými rozhraniami, pre vytváranie sniffovacích filtrov a pre samotné sniffovanie.

3.2. Globálne definície

3.2.1. Štruktúry hlavičiek

Program obsahuje definície špeciálnych štruktúr využívaných na reprezentáciu hlavičiek jednotlivých protokolov. Týmto spôsobom je definovaný formát Ethernet hlavičky, IPv4 hlavičky, IPv6 hlavičky, ARP hlavičky, TCP hlavičky, UDP hlavičky a ICMP hlavičky.

2.2.2. Ďalšie globálne definície

V programe sú využité aj globálne konštanty. Takto je definovaná veľkosť Ethernet hlavičky a veľkosť IPv6 hlavičky, ktoré majú pevnú veľkosť 14, resp. 40 bytov. Počet vypísaných bytov na jeden riadok pri výpise paketu je definovaný na 16 a výstupné chybové kódy sú, 1, pri chybe v argumentoch programu a 2, pri chybe nastávajúcej počas činnosti snifferu.

Premenná typu `pcap_t* handle`, ktorá slúži na manipuláciu so sieťovým rozhraním je definovaná ako globálna premenná z dôvodu potrebnej dostupnosti pri uvoľnení pamäti pri prerušení činnosti programu.

2.3. Činnosť programu

Činnosť programu spočíva vo vyhodnotení použitých možností príkazového riadku, vytvorení sniffovacieho filtru, založení spojenia so sieťovým zariadením využitím na sniffovanie a samotnej sniffovacej slučky.

2.3.1. Vyhodnotenie možností príkazového riadku

Vďaka využitiu knižnice `getopt` a funkcie `getopt_long()` je spracovanie možností/argumentov príkazového riadku pomerne jednoduché a priamočiare.

Pre normálnu funkcionálnosť snifferu je potrebné použiť možnosť *-i/--interface* s argumentom špecifikujúcim rozhranie. Pri nevyužití tejto možnosti, resp. nedodaní argumentu, program vypíše

všetky dostupné rozhranie a úspešne sa ukončí. Táto funkcionálna je definovaná vo funkcii `print_all_devs()`, ktorá volá funkciu `pcap_findalldevs()` z knižnice `pcap.h` a vypíše všetky vrátené rozhrania.

Pri využití možnosti `-p` a `-n` musia byť zadané argumenty a tieto musia byť platné celočíselné hodnoty, čo je ošetrené využitím funkcie `std::stoi()` a try-catch bloku.

Využitie možností špecifikujúcich typ pakiet je ukladané do príslušných premenných na základe, ktorých sa neskôr vytvára reťazec definujúci sniffovací filter.

2.3.2. Generovanie reťazca pre sniffovací filter

Reťazec je generovaný na základe využitia jednej až štyroch možností príkazového riadku špecifikujúcich typ pakiet a taktiež na základe využitia možnosti špecifikujúcej konkrétne číslo portu. Generácia reťazca je realizovaná pomocou viacerých konštrukcií if-else. Pre typy ARP a ICMP sa pridáva reťazec *"arp or "*, resp. *"icmp or "* a pre typy TCP a UDP podľa využitia možnosti špecifikácie portu buď reťazec *"tcp port x or "*, resp. *"udp port x or "* alebo reťazec *"tcp or "*, resp. *"udp or "*. Na koniec je ešte odstránené prebytočné *"or"*.

Pri nešpecifikovaní ani jedného typu pakety je daný využitý nasledovný reťazec, *"tcp or udp or arp or icmp"*, resp. *"tcp port x or udp port x or arp or icmp"*.

2.3.3. Založenie spojenia so sniffovaným zariadením

Táto časť programu sa skladá najmä z využívania funkcií knižnice `pcap.h`. Výstup každej z týchto funkcií je kontrolovaný a možnosť zlyhania je ošetrená prípadným uvoľnením využitých pamäťových zdrojov, výpisom príslušnej chyby a ukončením s príslušným chybovým kódom.

Najprv je pomocou funkcie `pcap_create()` inicializovaná už spomínaná premenná `handle`. Ďalej je využitá funkcia `pcap_set_promisc()` za účelom nastavenia promiskuitného módu sieťovej karty. Ďalšia využitá funkcia je `pcap_activate()` slúžiaca na aktivovanie spojenia.

Nasleduje kontrola typu využívaného rozhrania pomocou funkcie `pcap_data_link()`, ktorá vracia typ rozhrania. V prípade, že rozhranie nie je typu Ethernet je program ukončený, nakoľko je tento jediný podporovaný typ rozhrania.

Na záver je ešte potrebné aplikovať filter. Ten sa najprv na základe reťazca vygenerovaného na základe možností príkazového riadku skompiluje pomocou funkcie `pcap_compile()` a uloží sa do pripravenej štruktúry typu `bpf_program`. Následne môže byť filter aplikovaný pomocou funkcie `pcap_setfilter()`.

Príprava na sniffovanie je ukončená uvoľnením už nepotrebných štruktúr, v ktorej je uložený filter, ktorý bol práve aplikovaný. Na toto je využitá funkcia `pcap_freecode()`.

2.3.4. Sniffovacia slučka

Sniffovacia slučka je realizovaná pomocou funkcie `pcap_loop()` s použitými argumentmi `handle`, `num_packets`, `handle_ether`, `nullptr`. Táto funkcia volá `num_packets`-krát callback funkciu `handle_ether()` nad rozhraním v premennej `handle` s tým, že callback funkciou neposiela žiadne argumenty (iba `null`). Hodnota uložená v `num_packets` je špecifikovaná pomocou možnosti príkazového riadku `-n` alebo je využitá základná hodnota, 1.

Program obsahuje desať definícií funkcií, ktoré slúžia na spracovanie jednotlivých typov hlavičiek, ktoré sa môžu vyskytovať v odchytených paketoch a sú pre sniffer zaujímavé. Pre sniffer sú zaujímavé pakety typu ARP, ICMP, UDP a TCP, a preto potrebuje byť schopný spracovať hlavičky týchto protokolov a hlavičky všetkých protokolov nachádzajúcich sa na vyšších vrstvách. Sniffer teda spracováva hlavičky na prvej, druhej a prípadne tretej vrstve.

Keďže sniffovanie prebieha na rozhraní Ethernet, očakávaná prvá hlavička je Ethernet hlavička. Táto je spracovávaná funkciou `handle_ether()`, ktorá je použitá ako callback funkcia pri volaní `pcap_loop()`. Do premennej je uložený odchytený paket, pretypovaný na štruktúru Ethernet hlavičky `struct ethernet_header*`. Následne na základe údajov o nasledujúcej hlavičke je volaná príslušná funkcia na spracovanie tejto hlavičky.

V prípade, že nasleduje ARP hlavička, volá sa funkcia `handle_arp()`. Do premennej je uložený paket s offsetom o veľkosti Ethernet hlavičky, pretypovaný na štruktúru ARP hlavičky `struct arp_header*`. Následne program prejde na výpis paketu.

Podobne po Ethernet hlavičke môže nasledovať IPv4 alebo IPv6 hlavička. Pre tieto sa volá funkcia `handle_ipv4()`, resp. `handle_ipv6()`. Pre IPv4 hlavičku prebehne kontrola jej dĺžky, minimálne 20 bytov. IPv6 hlavička má stabilnú veľkosť 40 bytov, preto neprebíha žiadna kontrola. Následne sa pomocou konštrukcie `switch` na základe údajov o type hlavičky na ďalšej vrstve volá funkcia na jej spracovanie.

Keďže je filter nastavený maximálne na prijímanie UDP, TCP, ICMP a ARP paketov a ARP hlavička sa nachádza na druhej vrstve, po IPv4 alebo IPv6 hlavičke môžu nasledovať len hlavičky TCP, UDP alebo ICMP. Na ich spracovanie program využíva lokálne funkcie `handle_tcp()`, `handle_tcpv6()`, `handle_udp()`, `handle_udpv6()`, `handle_icmp()`, `handle_icmpv6()`. Hlavičky nasledujúce po IPv6 hlavičke majú rovnaký formát ako tieto isté hlavičky nasledujúce po IPv4 hlavičkách, no pre rozdielnosť argumentov, ktoré je potreba dať týmto funkciám a tiež s dôvodu prehľadnosti sú rozdelené do samostatných funkcií. Pri hlavičke na nasledujúcej vrstve je potrebné vedieť veľkosť tej predošlej, aby bolo možné vypočítať offset. Kdežto pri IPv6 hlavičkách je ich veľkosť daná, pri IPv4 hlavičkách je táto informácia zistená až za behu programu a teda pri volaní funkcií pre spracovanie nasledujúcich hlavičiek je odovzdávaná ako parameter. Taktiež, keďže výpis paketu sa vykonáva až pri spracovaní poslednej spracovanej hlavičky (nachádzajú sa tam niektoré potrebné informácie, napr. číslo portu), IP adresy odosielateľa a prijímateľa sú taktiež predávané ako parametre a logicky sa pri IPv6 a IPv4 adresách ich veľkosť líši. IPv4 adresy sú uložené v dátovom type `in_addr` a IPv6 adresy, vo formáte `in6_addr`.

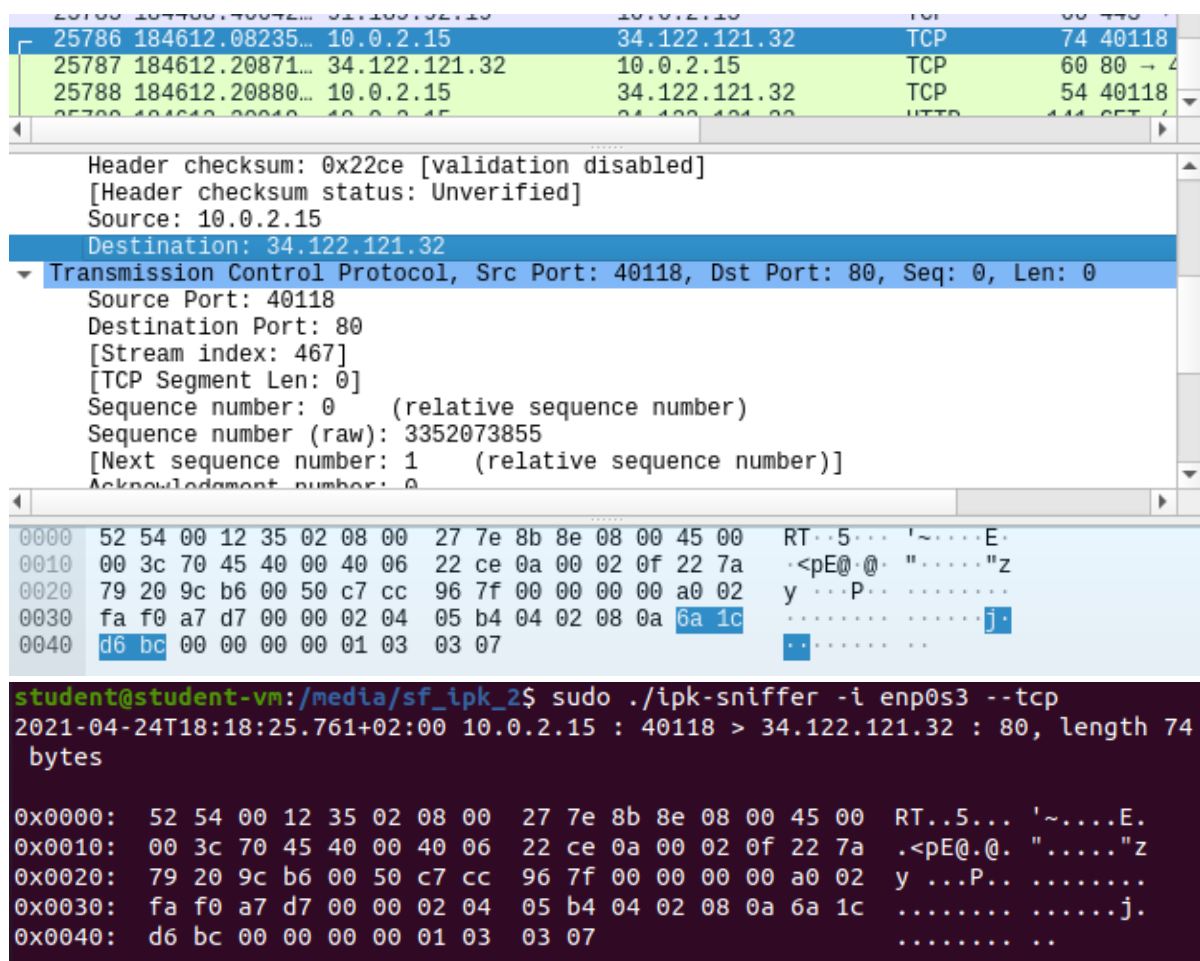
Vo funkciách na spracovanie ARP, ICMP, TCP a UDP hlavičiek na koniec prebieha výpis daného paketu. Najprv sa vypíše hlavička výpisu, skladajúca sa z časového údajov vo formáte RFC3339, IP adresy odosielateľa a prijímateľa a prípadne čísiel portov a veľkosti pakety v bytoch. Pri výpise ARP paketov sú miesto IP adresy vypisované MAC adresy. Na sformátovanie časového údajov je volaná lokálna funkcia `format_time()`. V tejto funkcii sa z údajov o sekundách od epochy pomocou funkcie `strftime()` vypočíta a sformátuje dátum a čas, ďalej sa údaj o mikrosekundách zaokrúhli na milisekundy a na záver sa pridá údaj o časovom pásme.

Následne prebieha formátovaný výpis celého paketu (hlavičky aj dáta). Pre tento účel sa volá funkcia `print_packet_content()`. V tejto funkcii sa pomocou konštrukcie `for` prejde každý byte daného paketu a po 16 bytoch na riadok sa najprv vypíše offset daného riadku a následne byty v hexadecimálnej reprezentácii a potom tie isté byty v ASCII reprezentácii. Nevypísateľné znaky sú nahradené znakom '.' a v polovici riadku je vždy vypísaná medzera pre lepšiu vizuálnu orientáciu.

3. Testovanie

Na testovanie funkcionality programu boli využité najmä programy *Wireshark*, *tcpreplay* a *ping*.

Pomocou programu *Wireshark* som sledoval jednotlivé pakety, ktoré odchyťoval aj môj program a porovnával som si výstup môjho programu s obsahom pakiet vypísaným týmto programom. V programe *Wireshark* som tiež využil možnosť nastavenia filtrov a otestoval som si tu pravidlá vytvárania reťazca pre tieto filtre. Využil som aj možnosť uloženia si odchytených paketov, ktoré som si znovu prehrával pomocou programu *tcpreplay*. Toto som použil najmä pri testovaní funkcionality IPv6 paketov, nakoľko nemám podporu IPv6 adres, a teda som ani nebol schopný takéto pakety vytvoriť, no z nejakého dôvodu sa mi podarilo využitím *Wiresharku* niekoľko zachytiť. Na vygenerovanie paketov podporovaných typov, ako napríklad ICMP, som využíval aj program *ping*.



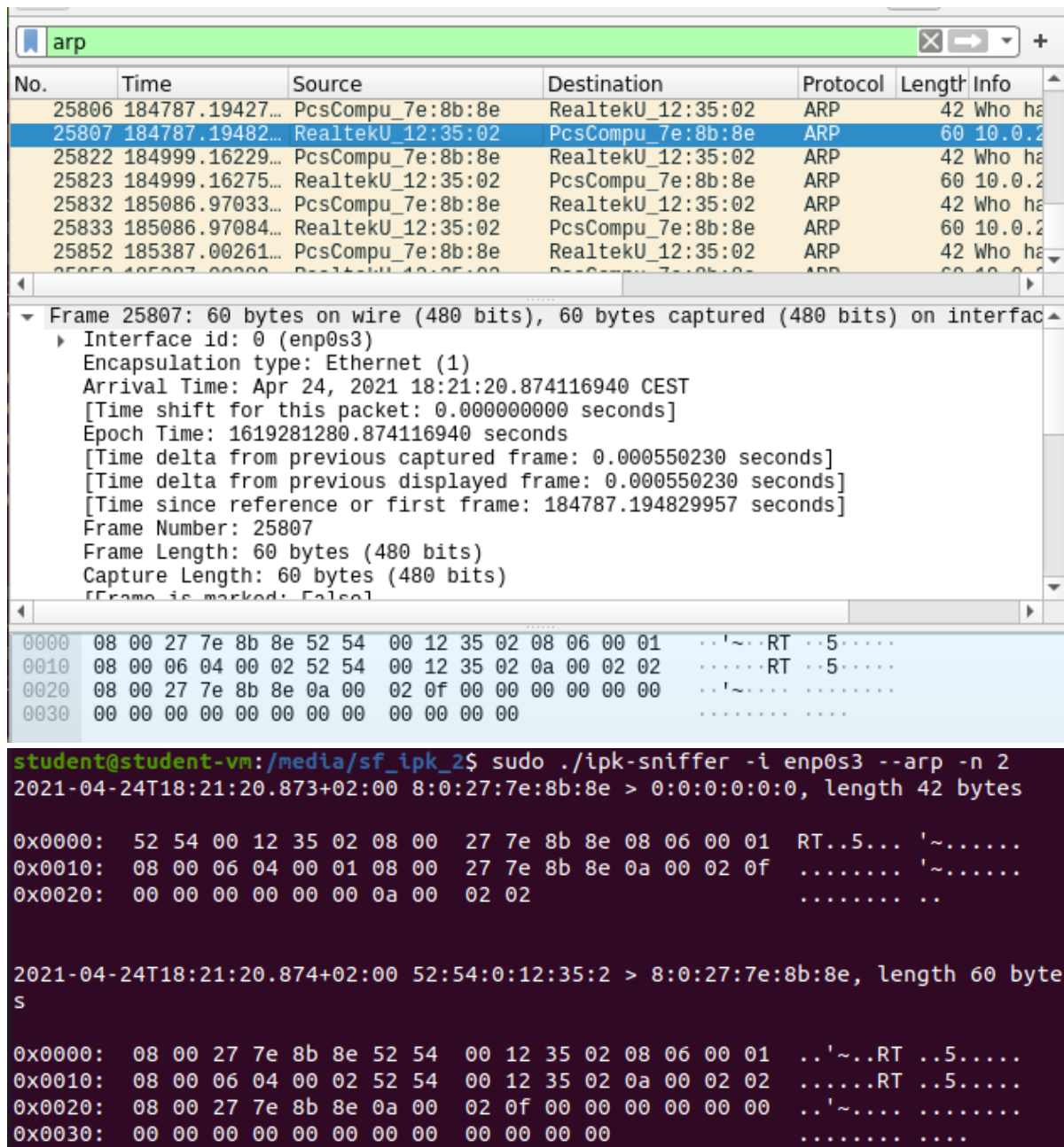
The screenshot displays the Wireshark network protocol analyzer interface. The top pane shows a list of captured packets, with packet 25786 selected. The middle pane shows the details of this packet, identifying it as a TCP segment from source 10.0.2.15 to destination 34.122.121.32, with source port 40118 and destination port 80. The bottom pane shows the raw packet data in hexadecimal and ASCII. Below the Wireshark interface, a terminal window shows the output of the 'ipk-sniffer' program, which has captured the same packet and displayed its details in a similar format to the Wireshark packet details pane.

```

student@student-vm:/media/sf_ipk_2$ sudo ./ipk-sniffer -i enp0s3 --tcp
2021-04-24T18:18:25.761+02:00 10.0.2.15 : 40118 > 34.122.121.32 : 80, length 74
bytes

0x0000:  52 54 00 12 35 02 08 00 27 7e 8b 8e 08 00 45 00  RT..5... '~...E.
0x0010:  00 3c 70 45 40 00 40 06 22 ce 0a 00 02 0f 22 7a  .<pE@.@. "...."z
0x0020:  79 20 9c b6 00 50 c7 cc 96 7f 00 00 00 00 a0 02  y ...P.. ....
0x0030:  fa f0 a7 d7 00 00 02 04 05 b4 04 02 08 0a 6a 1c  .....j.
0x0040:  d6 bc 00 00 00 00 01 03 03 07  .....

```

The image displays a Wireshark packet capture of ARP traffic and a terminal window showing the output of the `ipk-sniffer` tool.

Wireshark Packet Capture:

No.	Time	Source	Destination	Protocol	Length	Info
25806	184787.19427...	PcsCompu_7e:8b:8e	RealtekU_12:35:02	ARP	42	Who ha
25807	184787.19482...	RealtekU_12:35:02	PcsCompu_7e:8b:8e	ARP	60	10.0.2
25822	184999.16229...	PcsCompu_7e:8b:8e	RealtekU_12:35:02	ARP	42	Who ha
25823	184999.16275...	RealtekU_12:35:02	PcsCompu_7e:8b:8e	ARP	60	10.0.2
25832	185086.97033...	PcsCompu_7e:8b:8e	RealtekU_12:35:02	ARP	42	Who ha
25833	185086.97084...	RealtekU_12:35:02	PcsCompu_7e:8b:8e	ARP	60	10.0.2
25852	185387.00261...	PcsCompu_7e:8b:8e	RealtekU_12:35:02	ARP	42	Who ha

Frame 25807 details:

- Interface id: 0 (enp0s3)
- Encapsulation type: Ethernet (1)
- Arrival Time: Apr 24, 2021 18:21:20.874116940 CEST
- [Time shift for this packet: 0.000000000 seconds]
- Epoch Time: 1619281280.874116940 seconds
- [Time delta from previous captured frame: 0.000550230 seconds]
- [Time delta from previous displayed frame: 0.000550230 seconds]
- [Time since reference or first frame: 184787.194829957 seconds]
- Frame Number: 25807
- Frame Length: 60 bytes (480 bits)
- Capture Length: 60 bytes (480 bits)
- [Frame is marked: false]

Packet Bytes:

```

0000 08 00 27 7e 8b 8e 52 54 00 12 35 02 08 06 00 01  ..!~..RT ..5.....
0010 08 00 06 04 00 02 52 54 00 12 35 02 0a 00 02 02  ....RT ..5.....
0020 08 00 27 7e 8b 8e 0a 00 02 0f 00 00 00 00 00 00  ..!~....
0030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....

```

Terminal Output:

```

student@student-vm:/media/sf_ipk_2$ sudo ./ipk-sniffer -i enp0s3 --arp -n 2
2021-04-24T18:21:20.873+02:00 8:0:27:7e:8b:8e > 0:0:0:0:0:0, length 42 bytes

0x0000: 52 54 00 12 35 02 08 00 27 7e 8b 8e 08 06 00 01  RT..5... !~.....
0x0010: 08 00 06 04 00 01 08 00 27 7e 8b 8e 0a 00 02 0f  ....RT ..5.....
0x0020: 00 00 00 00 00 00 0a 00 02 02  ..!~....

2021-04-24T18:21:20.874+02:00 52:54:0:12:35:2 > 8:0:27:7e:8b:8e, length 60 bytes
s

0x0000: 08 00 27 7e 8b 8e 52 54 00 12 35 02 08 06 00 01  ..!~..RT ..5.....
0x0010: 08 00 06 04 00 02 52 54 00 12 35 02 0a 00 02 02  ....RT ..5.....
0x0020: 08 00 27 7e 8b 8e 0a 00 02 0f 00 00 00 00 00 00  ..!~....
0x0030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....

```

4. Záver

Program je funkčný a decentne otestovaný. Program sa vo viacerých prípadoch spolieha na korektnosť prijatého paketu a kontroluje minimum vecí, čo sa týka formátu paketu. Funkcionalita IPv6 paketov nebola dostatočne otestovaná z dôvodu neexistujúcej podpory od internetového dodávateľa.

5. Bibliografia

Základom môjho poznania o sniffovaní bola dokumentácia knižnice pcap, podľa ktorej som si vytvoril hlavnú konštrukciu celého programu. Taktiež som využil tu definované štruktúry pre Ethernet, IPv4 a TCP hlavičky, ktorými som sa tiež inšpiroval pri definovaní štruktúr pre hlavičky ostatných protokolov.

1. Programming with pcapTCPDUMP/LIBPCAP public repository. (21.4.2021). www.tcpdump.org.
<https://www.tcpdump.org/pcap.html>

Ďalej som využíval informácie z Linux man-pages. Predovšetkým spôsob volania funkcií, ich popis a ich návratové hodnoty.

Štruktúry hlavičiek jednotlivých protokolov.

2. Address Resolution Protocol (ARP) for the Identifier-Locator Network Protocol for IPv4 (ILNPv4). (21.4.2021). [trac.tools.ietf.org. https://trac.tools.ietf.org/html/rfc6747](http://tools.ietf.org/html/rfc6747)
3. Transmission Control Protocol. (21.4.2021). [trac.tools.ietf.org. https://tools.ietf.org/html/rfc793](http://tools.ietf.org/html/rfc793)
4. User Datagram Protocol. (21.4.2021). [trac.tools.ietf.org. https://tools.ietf.org/html/rfc768](http://tools.ietf.org/html/rfc768)
5. Internet Control Message Protocol. (21.4.2021). [trac.tools.ietf.org. https://tools.ietf.org/html/rfc792](http://tools.ietf.org/html/rfc792)
6. Internet Protocol, Version 6 (IPv6) Specification. (21.4.2021). [trac.tools.ietf.org. https://tools.ietf.org/html/rfc8200](http://tools.ietf.org/html/rfc8200)
7. Internet Protocol. (21.4.2021). [trac.tools.ietf.org. https://tools.ietf.org/html/rfc791](http://tools.ietf.org/html/rfc791)

Prevod časového údaju na formát RFC3339

8. C++ RFC3339 timestamp with milliseconds using std::chrono. Sebastian, <https://stackoverflow.com/users/1523730/sebastian>; Howard Hinnant, <https://stackoverflow.com/users/576911/howard-hinnant>. (20.4.2021). Stack Overflow.
<https://stackoverflow.com/questions/54325137/c-rfc3339-timestamp-with-milliseconds-using-stdchrono>

Ošetrenie vyvolania signálu interrupt (ctr+c)

9. How do I catch a Ctrl+C event in C++? (22.4.2021). Tutorials Point.
<https://www.tutorialspoint.com/how-do-i-catch-a-ctrlplusc-event-in-cplusplus>

Zdroje teórie.

10. What is a Packet Sniffer? (23.4.2021). www.kaspersky.com. <https://www.kaspersky.com/resource-center/definitions/what-is-a-packet-snifferr>
11. Address Resolution Protocol. (23.4.2021). Wikipedia.
https://en.wikipedia.org/wiki/Address_Resolution_Protocol
12. Internet Protocol. (23.4.2021). Wikipedia. https://en.wikipedia.org/wiki/Internet_Protocol
13. Transmission Control Protocol. (23.4.2021). Wikipedia.
https://en.wikipedia.org/wiki/Transmission_Control_Protocol
14. User Datagram Protocol. (23.4.2021). Wikipedia.
https://en.wikipedia.org/wiki/User_Datagram_Protocol
15. Internet Control Message Protocol. (23.4.2021). Wikipedia.
https://en.wikipedia.org/wiki/Internet_Control_Message_Protocol