

Počítačová komunikácia a siete – 2. projekt
Sniffer paketov

23. apríla 2021

Peter Zdravecký (xzdrav00)

Obsah

Úvod	2
Teória	2
1 Návrh	2
2 Implementácia	3
2.1 Použité knižnice	3
2.2 Spracovanie argumentov	3
2.3 Popis funkcií triedy <code>sniffer</code>	3
2.4 Popis funkcií triedy <code>packet</code>	4
3 Preklad a spustenie programu	6
4 Výstup a ukončenie programu	6
5 Testovanie	7
5.1 Odchytávanie paketov a porovnávanie	7
Záver	10
Literatúra	10

Úvod

Zadaním projektu je implementovať sniffer paketov, teda program, ktorý slúži na zachytávanie a filtrovanie paketov v sieti. Zachytávať pakety je možné na rôznych sieťových zariadeniach. Riešenie podporuje len zariadenia ethernetového typu. Program umožňuje aj filtrovať pakety na základe vybraných a dostupných filtrov. Filtrovať je možné podľa protokolu paketu (prípadne ich kombináciou) a zároveň aj podľa portu, na ktorom komunikácia prebieha. Výpis programu sa uskutočňuje na štandardný výstup `stdout`.

Teória

Sniffer

Odchyťovanie paketov (`packet sniffing`) je metóda, pomocou ktorej sa dajú čerpať dáta cestujúce v sieti. Využívajú sa najmä v rámci správy sietí na monitorovanie a kontrolu sieťovej prevádzky [3].

Pakety

Dátové pakety obsahujú vrstvy, každá vrstva má hlavičku a telo, telo zároveň obsahuje nižšiu vrstvu, teda ďalšiu hlavičku a telo. Postupným spracovaním paketu po vrstvách sa nakoniec vieme dostať k potrebným dátam [4].

1 Návrh

Program je napísaný v jazyku C++ a využíva sa v ňom objektovo orientované programovanie. Celý program je implementovaný v 3 moduloch: `main`, `sniffer`, `packet`.

V module `main` prebieha spracovávanie argumentov príkazového riadka a vytvorenie inštancie triedy `sniffer` a volanie metód tejto triedy.

Modul `sniffer` obsahuje implementáciu rovnomennej triedy. Trieda obsahuje dáta o konfigurácii zariadenia, ktoré bude pakety odchyťovať. Teda typ zariadenia, druh filtra, počet paketov, ktoré sa majú odchytiť, a pod. Inštancia tejto triedy potom vykonáva svoje metódy:

- pripojí sa na vybrané sieťové zariadenie,
- nakonfiguruje nastavenia, podľa ktorých sa odchyťávajú pakety,
- spúšťa zachytávanie paketov,
- vypíše všetky dostupné sieťové zariadenia.

Na spracovávanie a výpis zachytených paketov slúži trieda `packet` vo svojom module. Inštancia triedy `packet` obsahuje dáta o zachytenom pakete. Konkrétne sa jedná o dáta zachyteného paketu, dĺžku dát a čas zachytenia paketu. V metóde `parse()` sa zhromažďujú metadáta paketu (napr. adresa prijímateľa, odosielateľa, porty, ...). Potom sa podľa určeného druhu paketu volajú funkcie, ktoré „rozbalia“ paket po jednotlivých vrstvách. Trieda ešte obsahuje metódu `print_packet()` na výpis paketu.

2 Implementácia

2.1 Použité knižnice

Na implementáciu je použitá open source knižnica `libpcap`, konkrétne sa jedná o `pcap.h` [1], ktorá poskytuje možnosť monitorovať pohyb paketov po sieti.

Zoznam dôležitých knižníc použitých pri implementácii.

```
1 <pcap/pcap.h>
2 <iostream>
3 <stdio.h>
4 <stdlib.h>
5 <getopt.h>
6 <signal.h>
7 <string>

9 <netinet/udp.h>
10 <netinet/tcp.h>
11 <netinet/icmp6.h>
12 <netinet/ip_icmp.h>
13 <netinet/ip6.h>
14 <netinet/if_ether.h>
15 <netinet/igmp.h>
16 <net/ethernet.h>
```

2.2 Spracovanie argumentov

Pre spracovanie argumentov je využitá knižnica `getopt` [2]. Program prijíma krátke aj dlhé možnosti prepínačov. Dostupné prepínače pri spúšťaní programu:

```
1 [] = required argument, {} - optional argument
2 [ -i interface | --interface interface ]
3 {-p port}
4 {[--tcp | -t] [--udp | -u] [--arp] [--icmp]}
5 {-n count}
6 {-h | --help}
```

Pri zadaní prepínaču `-i` bez udania argumentu program vypíše na štandardný výstup dostupné sieťové zariadenia. Keďže knižnica `getopt` nepodporuje spracovanie tohto typu, v projekte sa tento problém ošetruje iným spôsobom [6].

2.3 Popis funkcií triedy `sniffer`

Funkcia `init()`

Slúži na konfiguráciu základných hodnôt potrebných k spusteniu zachytávaniu paketov v sieti. Potrebné konfiguračné dáta:

- zariadenie na ktorom sa bude zachytávať,
- filter ktorý bude použitý pri spracovaní paketov,
- čas ukončenia programu pri chýbajúcej odpovedi od sieťového zariadenia,
- mód promiskuitného použitia,

- počet paketov ktoré sa majú zachytiť a vypísať na obrazovku.

Následne pomocou vybraného zariadenia sa zistí jeho IPV4 adresa a jeho maska pomocou funkcie `pcap_lookupnet()`. Po vykonaní tejto funkcie sa spustí funkcia `pcap_open_live()`, ktorá získa popisovač zachytávania paketov. Zadaný filter v textovej podobe sa preloží do filtrovacieho programu pomocou funkcie `pcap_compile()` a nastaví pomocou `pcap_setfilter()`. Na zistenie typu datalinku zariadenie sa volá funkcia `pcap_datalink()`, ktorá uloží typ zariadenie do premennej kontrolovanej z funkcie `main()`.

Pri akejkoľvek chybe sa vypíše chybové hlásenie a program skončí s chybovou návratovou hodnotou 1.

Funkcia `print_interfaces()`

Slúži na výpis všetkých dostupných sieťových zariadení. Pri chybe sa vypíše chybové hlásenie a program skončí s návratovou hodnotou 1.

Funkcia `capture_packets()`

V tejto funkcii sa začína zachytávanie paketov funkciou `pcap_loop()` a posielaním ich do funkcie `handle_packet()`, ktorá pakety spracúva.

Funkcia `handle_packet()`

Vytvorí a inicializuje nový objekt `packet` dátami prijatými funkciou `pcap_loop()`. Následne sa vytvorený paket spracuje funkciou `parse()`. Po úspešnom spracovaní sa paket vypíše funkciou `print_packet()`.

Funkcia `exit_sniffer()`

Korektne ukončí beh snifferu, ak stále sa nachádza vo funkcii `pcap_loop()`. Beh tejto funkcie indikuje premenná `running` a volá sa `loop_running()`. K vypnutiu behu tejto funkcie slúži funkcia s názvom `pcap_breakloop()` a následne sa aj uvoľní pamäť popisovača.

2.4 Popis funkcií triedy `packet`

Konstruktör

Pri vytváraní paketu alokuje pamäť, do ktorej sa ukladá časová stopa zachytenia paketu a dĺžka prijatého paketu. Ukladá sa aj dátová časť paketu, ktorá bude slúžiť na zistenie metadáta prijatého paketu.

Deštruktör

Slúži na uvoľnenie alokovaných zdrojov pri vytváraní paketu.

Funkcia `print_packet()`

Slúži na výpis celého paketu. Volá špecifické funkcie na výpis jednotlivých častí paketu. Výpis paketu je podmienený jeho správnym spracovaním.

Funkcia `print_header()`

Slúži na výpis hlavičky paketu. Hlavička paketu obsahuje čas prijatia paketu, zdroj a cieľ, kam sa paket posielal a dĺžku paketu v bajtoch.

Funkcia `print_data()`

Slúži na výpis formátovaných dát paketu. Dáta sa po bajtoch zapisujú ako v hexadecimálnej forme, tak aj v ASCII formáte (iba v prípade, že daný znak možno vypísať ASCII znakom, inak sa vypíše bodka).

Funkcia `parse()`

Obslužná funkcia pre spracovanie paketu. V tejto funkcii sa zisťuje z ethernetovej hlavičky, o aký typ paketu sa jedná (IPv4, IPv6 alebo ARP). Podľa toho sa rozhodne, ktorá funkcia na spracovanie ďalšej vrstvy paketu sa zavolá.

Funkcia `handle_ip_packet()`

Spracúva pakety typu IPv4. Z IPv4 hlavičky sa zistí, o aký protokol sa jedná TCP, UDP, ICMP, IGMP. Po spracovaní sa metadáta sa zapíšu do hlavičky.

Funkcia `handle_ip6_packet()`

Spracúva pakety typu IPv6. Z IPv6 hlavičky sa zistí, o aký protokol sa jedná TCP, UDP, ICMP. Po spracovaní sa metadáta zapíšu do hlavičky.

Funkcia `handle_arp_packet()`

Spracúva pakety typu ARP. Z ARP hlavičky sa zistí zdrojová a cieľová adresa a zapíše sa do hlavičky.

Funkcia `get_packet_time()`

Slúži na formátovanie časovej stopy prijatia paketu. Použitý časový formát je RFC3339 [5].

3 Preklad a spustenie programu

Program sa prekladá pomocou príkazu `make`, ktorý vytvorí spustiteľný súbor v koreňovom priečinku. Príkaz `make clean` vymaže dočasné súbory a vytvorené spustiteľné súbory.

Možné príklady spustenia:

```
1 $ ./ipk-sniffer -i
2 $ ./ipk-sniffer -i eth0 -t -n 2
3 $ ./ipk-sniffer -i eth0 --udp
4 $ ./ipk-sniffer -i lo -u -t -p 80
5 $ ./ipk-sniffer -h
```

4 Výstup a ukončenie programu

Program spracuje zachytené pakety a vypisuje ich na štandardný výstup. Výpis obsahuje hlavičku, v ktorej sú uložené metadáta paketu. Zvyšok výpisu tvorí celá dátová časť paketu vo formátovanom tvare.

```
1 2021-04-12T21:16:24.219+02:00 127.0.0.1 > 127.0.0.1, length 98 bytes
3 0x0000:  00 00 00 00 00 00 00 00 00 00 00 00 08 00 45 00  .....E.
4 0x0010:  00 54 3B 89 40 00 40 01 01 1E 7F 00 00 01 7F 00  .T;.@.@. ....
5 0x0020:  00 01 08 00 C3 9F 00 37 00 01 88 9C 74 60 00 00  .....7 ....t\..
6 0x0030:  00 00 75 58 03 00 00 00 00 00 10 11 12 13 14 15  ..uX.....
7 0x0040:  16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23 24 25  ..... !"#$$%
8 0x0050:  26 27 28 29 2A 2B 2C 2D 2E 2F 30 31 32 33 34 35  &'()*+,-./012345
9 0x0060:  36 37                                           67
```


Ukážka 1: Vzorový výstup zachyteného paketu ICMP

```
1 2021-04-13T14:10:06.126+02:00 fe80::215:5dff:feab:2ac7 : 58817 > fe80::215:5dff:
   feab:2ac7 : 80, length 74 bytes
3 0x0000:  00 00 00 00 00 00 00 00 00 00 00 00 86 DD 60 01  .....`.
4 0x0010:  EE B9 00 14 06 40 FE 80 00 00 00 00 00 00 02 15  .....@..
5 0x0020:  5D FF FE AB 2A C7 FE 80 00 00 00 00 00 00 02 15  ]...*...
6 0x0030:  5D FF FE AB 2A C7 E5 C1 00 50 A1 E3 8E 5F 00 00  ]...*... .P..._..
7 0x0040:  00 00 50 02 05 C8 83 B5 00 00                ..P..... ..
```

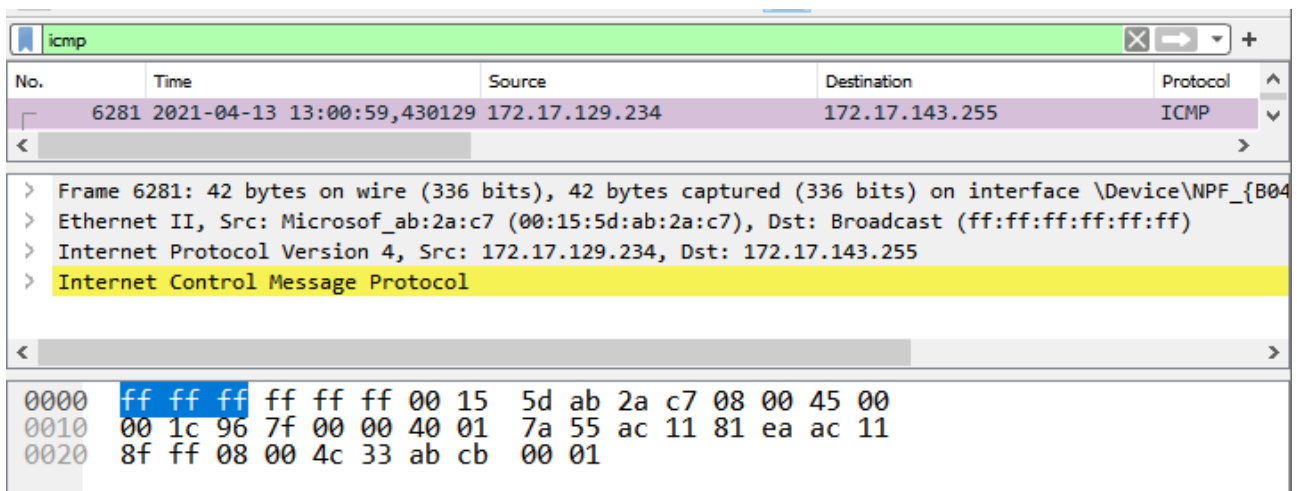
Ukážka 2: Vzorový výstup zachyteného paketu TCP IPv6

Program by sa nemal v žiadnom prípade nemal ukončiť neočakávanou chybou (`segmentation fault` a podobne). Možné chybové stavy sú chytené a ošetrené počas behu programu s príslušnými chybovými hláškami. Pri akejkolvek chybe sa vypíše chybové hlásenie na štandardný chybový výstup `stderr` a program skončí s chybovou návratovou hodnotou 1. Okrem toho sa v programe odchyťava aj signál `Keyboard interrupt`, po ktorom sa program korektné ukončí volaním funkcie `exit_sniffer()`.

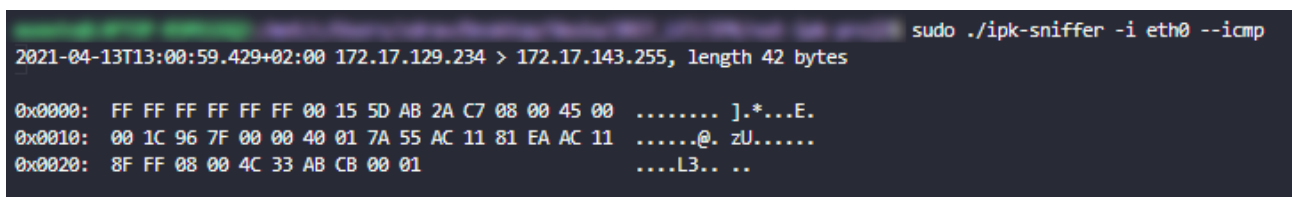
5 Testovanie

Testovanie prebiehalo pomocou zachytávania paketov a porovnávania výstupov na referenčnom stroji PDS-VM a stroji s operačným systémom Ubuntu 20.04.2.0 LTS. Program použitý na kontrolu bol  wireshark¹. Testované boli všetky programom podporované typy paketov. Pár vybraných testov sa nachádzajú v sekcii 5.1. Všetky testy sú zaznamenané v sekcii 5.1. Na simulovanie paketov bol použitý program ping a nping².

5.1 Odchytyvanie paketov a porovnávanie



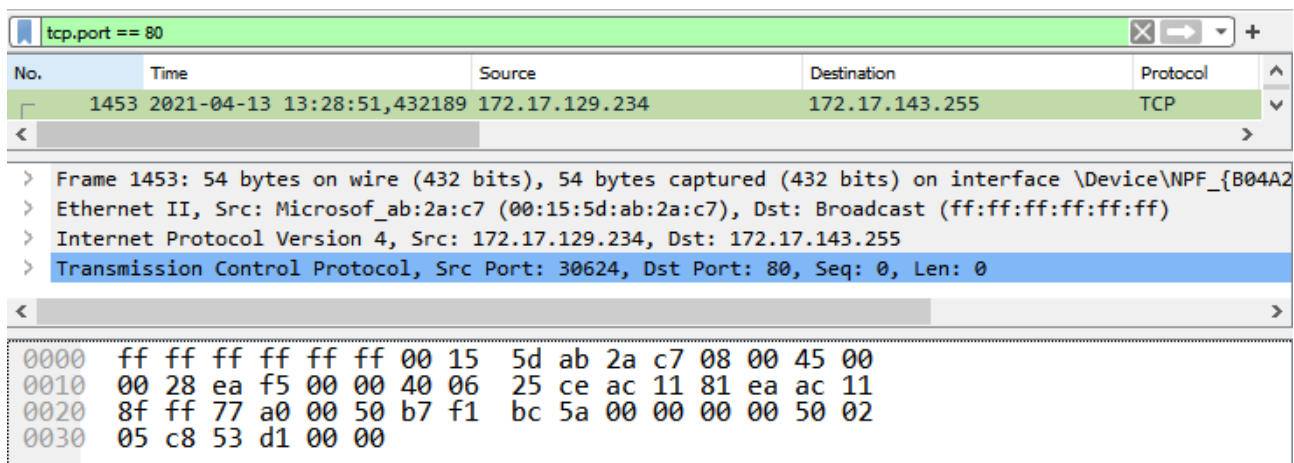
Obr. 1: Zachytenie ICMP paketu programom Wireshark



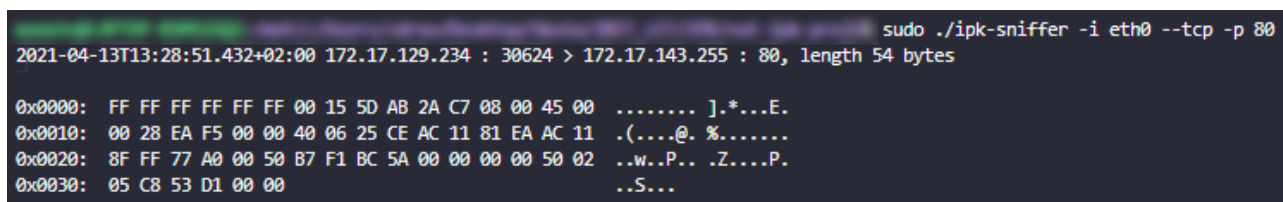
Obr. 2: Zachytenie ICMP paketu pomocou implementovaného programu

¹Open source program, ktorý slúži na zachytávanie paketov v sieti: <https://www.wireshark.org/>

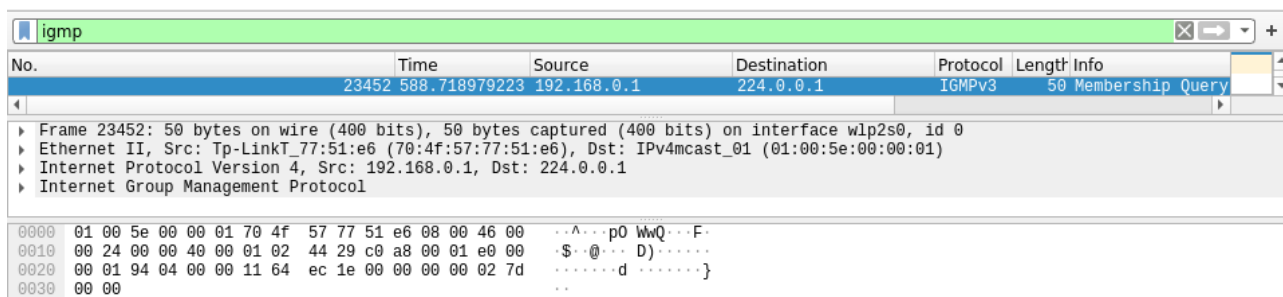
²Open source program na generovanie paketov v sieti: <https://nmap.org/nping/>



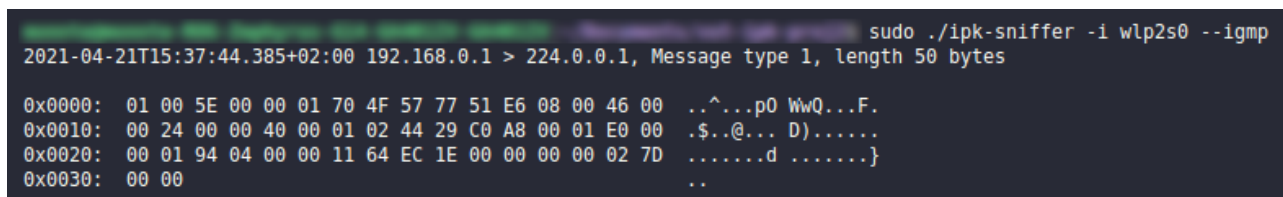
Obr. 3: Zachytenie TCP paketu (port 80) programom Wireshark



Obr. 4: Zachytenie TCP paketu (port 80) pomocou implementovaného programu



Obr. 5: Zachytenie IGMP paketu programom Wireshark



Obr. 6: Zachytenie IGMP paketu pomocou implementovaného programu

arp						
No.	Time	Source	Destination	Protocol	Length	Info
5	2021-04-23 10:16:35.006632	Microsof_a7:5e:1b	Broadcast	ARP	42	Who has 172.25.15.255? Tell 172.25.15.86
<div>> Frame 5: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface \Device\NPF_{3E9A116D-C3E4-4BE5-8D88-9C188E52881A}, id 0</div> <div>> Ethernet II, Src: Microsof_a7:5e:1b (00:15:5d:a7:5e:1b), Dst: Broadcast (ff:ff:ff:ff:ff:ff)</div> <div>> Address Resolution Protocol (request)</div>						
0000	ff ff ff ff ff ff	00 15 5d a7 5e 1b	08 06 00 01			
0010	08 00 06 04 00 01	00 15 5d a7 5e 1b	ac 19 0f 56			
0020	ff ff ff ff ff ff	ac 19 0f ff				

Obr. 7: Zachytenie ARP paketu programom Wireshark

```

sudo ./ipk-sniffer -i eth0 --arp
2021-04-23T10:16:34.806+02:00 00:15:5d:a7:5e:1b > ff:ff:ff:ff:ff:ff, length 42 bytes

0x0000: FF FF FF FF FF FF 00 15 5D A7 5E 1B 08 06 00 01 ..... ].^.....
0x0010: 08 00 06 04 00 01 00 15 5D A7 5E 1B AC 19 0F 56 ..... ].^....V
0x0020: FF FF FF FF FF FF AC 19 0F FF ..... ..

```

Obr. 8: Zachytenie ARP paketu pomocou implementovaného programu

Záver

Projekt spĺňa požiadavky zadania, nad rámec požiadaviek je implementovaná podpora IGMP paketov.

Literatúra

- [1] The Tcpdump Group, *Man page of PCAP* [online]. Posledná modifikácia: 9.9.2020 [cit. 7.4.2021]. Dostupné z: <https://www.tcpdump.org/manpages/pcap.3pcap.html>.
- [2] *Parsing program options using getopt* [online]. [cit. 8.4.2020]. Dostupné z: https://www.gnu.org/software/libc/manual/html_node/Getopt.html.
- [3] Ansari, S., Rajeev, S., Chandrashekar, H. Packet sniffing: a brief introduction. *IEEE Potentials*, 2003, vol. 21, č. 5: s. 17–19. doi:10.1109/MP.2002.1166620.
- [4] Chapman, D. B., Zwicky, E. D. *Building Internet Firewalls*. [online]. O'Reilly Associates, 1995. 517 s. ISBN 1-56592-124-0. Dostupné z: <https://www.cs.ait.ac.th/~on/O/oreilly/tcpip/firewall/index.htm>.
- [5] Klyne, e. a. *Date and Time on the Internet: Timestamps* [online]. Posledná modifikácia: júl 2002 [cit. 9.4.2021]. Dostupné z: <https://www.ietf.org/rfc/rfc3339.txt>.
- [6] Vandenberg, B. *getopt does not parse optional arguments to parameters* [online]. Posledná modifikácia: 21.4.2020 [cit. 8.4.2021]. Dostupné z: <https://stackoverflow.com/questions/1052746/getopt-does-not-parse-optional-arguments-to-parameters>.