

Počítačová komunikácia a siete – 2. projekt
Sniffer paketov

Obsah

Úvod	2
1 Implementácia	2
1.1 Základne štruktúra	2
1.2 Spracovanie argumentov	2
1.3 Trieda sniffer	2
1.3.1 Zoznam funckii	2
1.3.2 Funkcia <code>init</code>	3
1.3.3 Funkcia <code>print_interfaces</code>	3
1.3.4 Funkcia <code>capture_packets</code>	3
1.3.5 Funkcia <code>handle_packet</code>	3
1.3.6 Funkcia <code>exit_sniffer</code>	3
1.4 Trieda <code>packet</code>	3
1.4.1 Zoznam funckii	4
1.4.2 Konštruktor	4
1.4.3 Deštruktor	4
1.4.4 Funkcia <code>print_packet</code>	4
1.4.5 Funkcia <code>print_header</code>	4
1.4.6 Funkcia <code>print_data</code>	4
1.4.7 Funkcia <code>parse</code>	4
1.4.8 Funkcia <code>handle_ip_packet</code>	4
1.4.9 Funkcia <code>handle_ip6_packet</code>	5
1.4.10 Funkcia <code>handle_arp_packet</code>	5
1.4.11 Funkcia <code>get_packet_time</code>	5
1.5 Chybové stavy	5
1.6 Použité knižnice	5
2 Preklad programu	5
3 Možnosti spustenia	6
4 Výstup programu	6
5 Testovanie	6
5.1 Ukážky	7
Záver	8
Literatúra	8

Úvod

Sniffer paketov je program ktorý slúži na zachytávanie paketov v sieti. Zachytávať pakety je možné na rôznych sieťových zariadeniach ktoré sú ethernetového typu. V programe je implementovaná podpora filtrovania paketov na základe portu alebo vybraného protokolu. Fitlre je možné kombinovať. Podpora pre TCP UDP ICMP ICMP6 ARP typy paketov[3]. Výpis programu sa uskutočňuje na štandardný výstup stdout.

1 Implementácia

1.1 Základne štruktúra

Program je napísaný v jazyku c++ a využíva sa v ňom objektovo orientované programovanie. Na implementáciu bolo použitá open source knižnica libcap[1], konkrétne sa jedná o pcap.h ktorá poskytuje možnosť monitorovať pohyb paketov po sieti. V programe sa nachádzajú 2 triedy sniffer a packet. Program je rozdelený do 6 súborov main.cpp sniffer.cpp packet.cpp + ich hlavičky. V súbore mainc.cpp sa spracovávajú argumenty, vytvára sa objekt sniffer, inicializuje sa a spúšťa sniffovanie paketov.

1.2 Spracovanie argumentov

K spracúvaniu argumentov je využitá knižnica getopt[2]. Program prijíma krátke aj dlhé možnosti prepínačov.

Dostupné možnosti pri spúšťaní programu:

```
1 [] - required
2 {} - optional

4 [-i interface | --rozhranie interface ]
5 {-p port}
6 [--tcp|-t] [--udp|-u] [--arp] [--icmp]
7 {-n num}
8 {-h | --help}
```

Pri zadaní prepínaču -i bez udania argumentu program vypíše na štandardný výstup dostupné sieťové zariadenia.[5]

1.3 Trieda sniffer

Trieda sniffer je napísaná v 2 súboroch sniffer.cpp a sniffer.h. Obsahuje funkcie na inicializáciu snifferu, zachytávanie paketov a vytvára objekty z triedy packet.

1.3.1 Zoznam funkcií

```
1 sniffer() # Constructor
2 ~sniffer() # Destructor
3 int init(char* interface, char* filter, int timeout, int promisc, int packet_cnt);
4 int print_interfaces();
5 int capture_packets();
6 void exit_sniffer();
```

a funkcia `void handle_packet(args);` ktorá nepatrí do triedy `sniffer` ale nachádza sa v súbore `sniffer.h`.

1.3.2 Funkcia `init`

Funkcia `init` slúži na inicializovanie základných hodnôt potrebné k spusteniu sniffovania paketov v sieti. Je potrebné nastaviť zariadenie na ktorom sa bude sniffovať. Filter ktorý bude použitý pri spracovaní paketov. Čas ukončenia programu pri neodpovedaní sieťového zariadenia. Mód promiskuitného použitia a počet paketov ktoré sa majú zachytiť a vypísať na obrazovku.

Následne pomocou vybraného zariadenia zistíme jeho IPV4 adresu a jeho masku pomocou funkcie `pcap_lookupnet` ktorá sa nachádza v knižnici `pcap.h`. Po vykonaní tejto funkcie sa spúšťa funkcia `pcap_open_live` ktorá sa používa na získanie popisovača zachytávania paketov. Zadaní 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 ktorá sa kontroluje vo funkcii `main`.

Pri akekoľvek chybe sa vypíše chybová hláška na štandardný errorový výstup `stderr` a návratová hodnota funkcie sa nastaví na 1 ktorá signalizuje chybu

1.3.3 Funkcia `print_interfaces`

Funkcia `print_interfaces` slúži na výpis všetkých dostupných sieťových zariadení. Pri chybe návratová hodnota funkcie sa nastaví na 1 ktorá signalizuje chybu a vypisuje sa chybová hláška.

1.3.4 Funkcia `capture_packets`

V tejto funkcii sa zahajuje zachytávanie paketov pomocou funkcie z knižnice `pcap.h` s názvom `pcap_loop` a posielanie ich do funkcie `handle_packet` ktorá ich spracúva.

1.3.5 Funkcia `handle_packet`

Obslužná funkcia pre spracovanie paketov. Tvorí sa nový objekt triedy `packet` a inicializuje sa pomocou dát ktoré boli predané z funkcie `pcap_loop`. Následne sa nad vytvoreným paketom volá funkcia `parse` ktorá packet spracuje. Po spracovaní sa zavolá funkcia na výpis paketu `print_packet`. Výpis paketu je podmienený podľa toho či Paket bol spracovaný správne.

1.3.6 Funkcia `exit_sniffer`

Ukončuje korektne beh snifferu ak stále sa nachádza vo funkcii `pcap_loop`. Beh tejto funkcie indikuje premenná nachádzajúca sa v triede `sniffer` a volá sa `loop_running`. K vypnutiu behu tejto funkcie slúži funkcia s názvom `pcap_breakloop` a následne sa aj uvoľní pamäť popisovača.

1.4 Trieda `packet`

Trieda `packet` je napísaná v 2 súboroch `packet.cpp` a `packet.h`. Táto trieda slúži na vytváranie paketov, spracovanie paketov podľa typu a vytvorenie hlavičky na základe metadát získaných z prijatých dát paketu. Obsahuje funkciu na formátovaný výpis celého paketu.

1.4.1 Zoznam funkcii

```
1 packet(const pcap_pkthdr* pkt_header, const u_char* pkt_data); # Constructor
2 ~packet() # Destructor
3 void print_packet();
4 void print_header();
5 void print_data();
6 void parse();
7 void handle_ip_packet();
8 void handle_ip6_packet();
9 void handle_arp_packet();
10 string get_packet_time();
```

1.4.2 Konštruktor

Pri vytváraní paketu sa alokuje pamäť do ktorej sa ukladá hlavička prijatého paketu kde je uložená dĺžka prijatých dát a časová stopa kedy bol paket prijatý. Ukladá sa aj dátová časť paketu ktorá bude slúžiť na zistenie metadáta prijatého paketu.

1.4.3 Deštruktor

Slúži na uvoľnenie alokovaných zdrojov pri vytváraní paketu.

1.4.4 Funkcia `print_packet`

Výpis celého paketu. Volá podfunkcie na výpis jednotlivých častí paketu.

1.4.5 Funkcia `print_header`

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

1.4.6 Funkcia `print_data`

Výpis formátovaných dát paketu. Výpis každého bajtu v hexadecimalnej forme a aj v ASCII formáte ak je znak možné vypísať, ak nie vypisuje sa bodka.

1.4.7 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 rozhodné ktorá funkcia na spracovanie ďalšej vrstvy paketu sa zavolá.

1.4.8 Funkcia `handle_ip_packet`

Spracovanie paketov typu IPv4. Z IPv4 hlavičky sa zistí o aký protocol sa jedná TCP UDP ICMP spracuje sa a metadáta sa zapíšu do hlavičky.

1.4.9 Funkcia `handle_ip6_packet`

Spracovanie paketov typu IPv6. Z IPv6 hlavičky sa zistí o aký protocol sa jedná TCP UDP ICMP spracuje sa a metadáta sa zapíšu do hlavičky.

1.4.10 Funkcia `handle_arp_packet`

Spracovanie paketov typu ARP. Z ARP hlavičky sa zistí zdrojová a cieľová adresa a zapíše sa do hlavičky.

1.4.11 Funkcia `get_packet_time`

Funkcia slúži na formátovanie časovej stopy prijatia paketu. Použitý časový formát je RFC3339.[4]

1.5 Chybové stavy

Program by sa nemal v žiadnom prípade ukončiť na násilne spadnutie. Možné chybové stavy sú ošetrené a odchyťované počas behu programu s príslušnými chybovými hláškami ktoré sú vypisované na štandardný errorový výstup `stderr`. Ukončenie pomocou signálu `Keyboard interrupt` je odchytené a po ňom sa program korektné ukončí volaním funkcie `exit_sniffer`.

1.6 Použité knižnice

List dôležitých knižníc použitých pri implementácii.

```
1 # Main functionality
2 <pcap/pcap.h>
3 <iostream>
4 <stdio.h>
5 <stdlib.h>
6 <getopt.h>
7 <signal.h>
8 <string>
9 # Packet headers
10 <netinet/udp.h>
11 <netinet/tcp.h>
12 <netinet/icmp6.h>
13 <netinet/ip_icmp.h>
14 <netinet/ip6.h>
15 <netinet/if_ether.h>
16 <net/ethernet.h>
```

2 Preklad programu

Program sa prekladá pomocou príkazu `make`. V adresárovej štruktúre sa nachádza súbor s názvom `Makefile` ktorý slúži na správne zostavenie programu. Príkaz `make clean` slúži na vymazanie dočasných súborov a samotného programu.

3 Možnosti spustenia

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

4 Výstup 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 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 02 15 .....@.. ....
5 0x0020: 5D FF FE AB 2A C7 FE 80 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

5 Testovanie

Testovanie prebiehalo pomocou vytvárania paketov a porovnávania výstupu z programu  wireshark¹. Kontrola správneho zachytávania paketov prebehla na všetkých typoch ktoré program podporuje TCP UDP ICMP ICMP6 ARP. Na vytváranie paketov bol použitý program ping a nping.

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

5.1 Ukážky

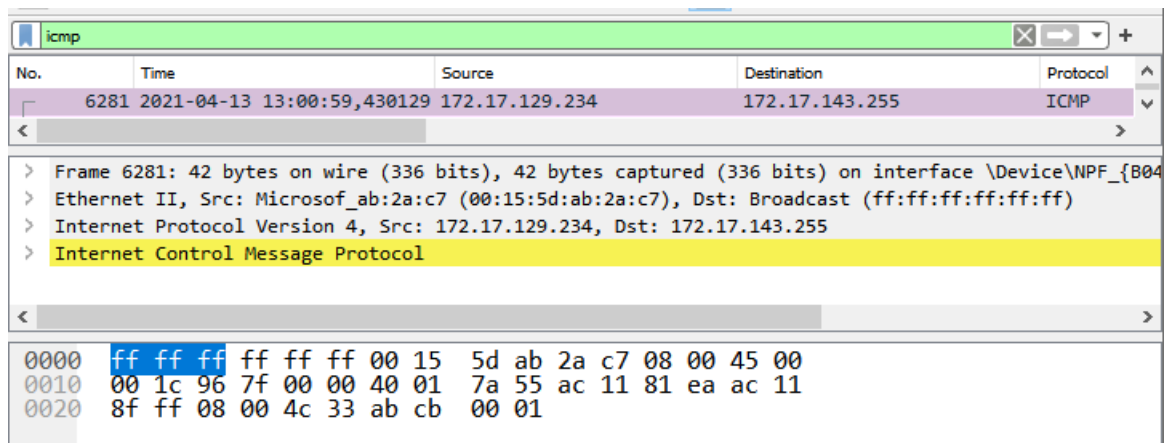
Vytváranie paketov

```
1 $ nping --icmp 127.0.0.1      # ICMP packet
2 $ nping --tcp 127.0.0.1      # TCP packet
3 $ nping --udp 127.0.0.1      # UDP packet
4 $ nping --arp 127.0.0.1      # ARP packet

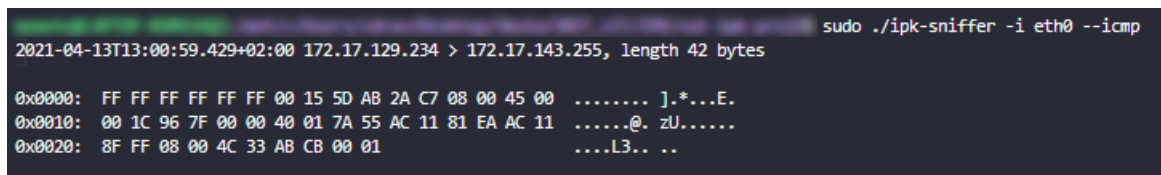
6 $ nping -6 --tcp ::1          # TCPv6 packet
7 $ nping -6 --udp --badsum ::1 # UDPv6 packet
8 $ ping -6 ::1                # ICMP6 packet
```

Odchyťovanie paketov a porovnávanie:

Test zachytenia ICMP paketu:

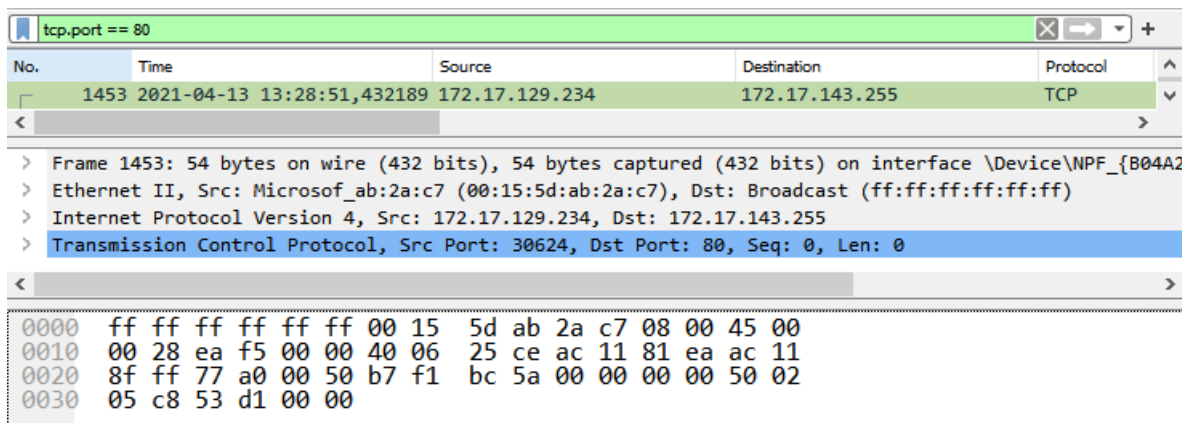


Obr. 1: Program Wireshark zachytenie paketu ICMP

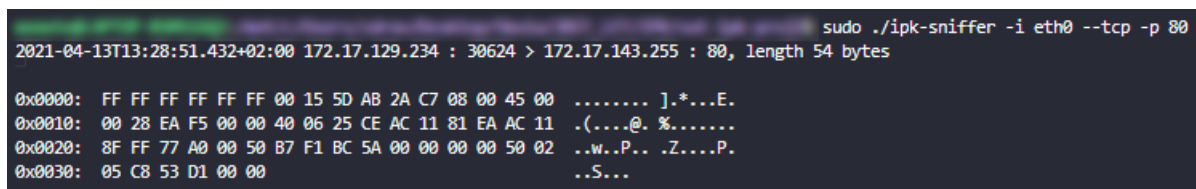


Obr. 2: Zachytenie paketu ICMP pomocou môjho programu

Test zachytenia TCP packetu na porte 80:



Obr. 3: Program Wireshark zachytenie packetu TCP port 80



Obr. 4: Zachytenie packetu TCP port 80 pomocou môjho programu

Záver

Projekt je plne funkčný podľa zadania a bol riadne otestovaný.

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] 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>.
- [4] 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>.
- [5] 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>.