



IPK – protokol k projektu

Juraj Novosád

21. apríla 2022

Obsah

1	1. Teória	2
2	2. Zadanie	2
3	3. Implementácia	2
3.1	Architektúra programu	2
3.2	Použité knižnice na spracovanie paketov	3
3.3	Spracovanie argumentov príkazového riadku	3
3.4	Spracovanie rámcov	4
4	Testovanie	4

1. Teória

Komunikácia je základom modernej spoločnosti. Medziľudská, ale v tomto kontexte pôjde hlavne o tú medzi počítačmi.

Počítače môžu komunikovať na veľa spôsobov. Najrozšírenejším je však komunikovanie pomocou správ. Správa je normalizovaná jednotka s dátami. Výhoda je v tom, že aj veľký súbor sa dá rozdeliť na veľa malých správ, ktoré sa posielajú postupne, kľudne aj s prestávkami. Takto jeden počítač neobsadzuje a nezanáša prenosové médium jedným dlhým prenosom, ale veľa malými jednotkami.

Internet funguje na princípe týchto balíkov, nazývajú sa pakety. Štruktúra každého paketu je daná jeho protokolmi. Protokol musí byť formálne definovaný a určuje význam jednotlivých bitov paketu. Pričom paket protokolu x obvykle ako putuje po sieti zapúzdruje do seba paket s protokolom y a ten môže zapúzdruvať ďalší. Je to preto, že v rôznych častiach siete sa orientuje podľa rôznych protokolov.

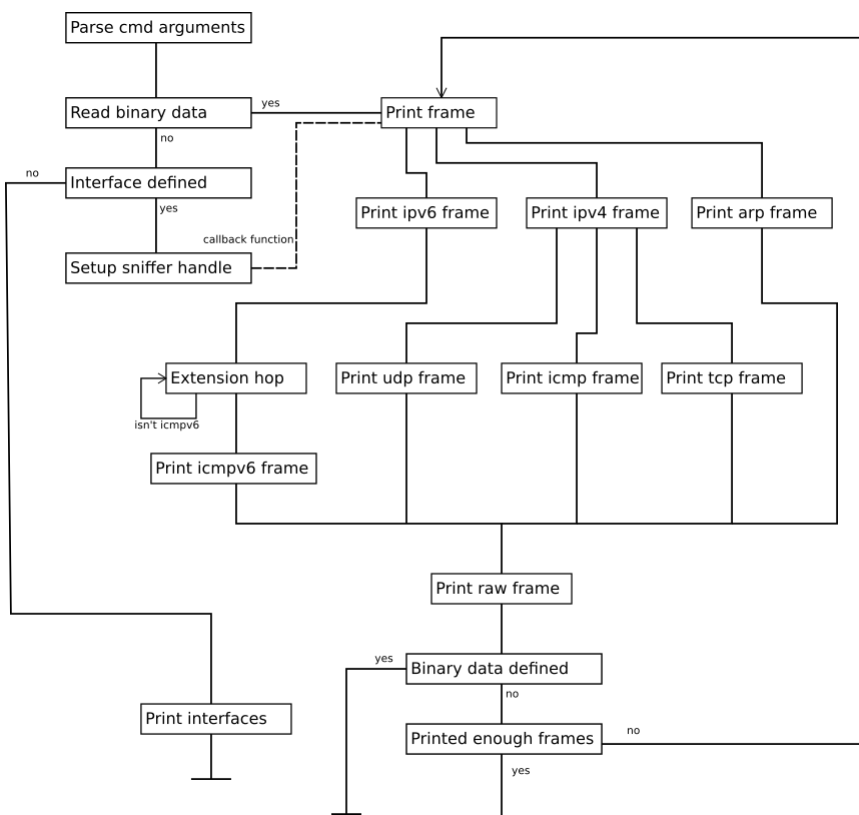
2. Zadanie

Cieľom projektu bolo implementovať program na zachytávanie paketov a ich dekódovanie do ľudske čitateľnej podoby. Inak povedané sniffer paketov.

3. Implementácia

Architektúra programu

Nasledujúci diagram abstraktne znázorňuje tok programu.



Obr. 1: Architektúra programu

Vykonávanie sa začína spracovaním argumentov z príkazového riadku

Na lepšie otestovanie som implementoval možnosť spracovania paketu uloženého ako binárne dáta. K tomu sa viaže blok `read binary data`. Viac o tom v časti o testovaní. Inak sa kontroluje, či je zadané rozhranie a nastaví sa rukoväť a filter na odpočúvanie. Print frame dostane raw data paketu, prečíta ethernet hlavičku a rozhodne aký protokol je vnútri, podľa toho zavolá ďalšiu funkciu. Ostatné funkcie fungujú podobne.

Extension hop pred `icmpv6`, je funkcia, ktorá preskáče extension hlavičky v ipv6 pakete až dokým sa nedostane na koniec alebo k `icmpv6` paketu.

Vypíšu sa raw data paketu. Pokiaľ sa spracovával paket zo súboru, program sa ukončí, takto sa dá spracovať iba jeden paket. Inak sa zvýši počítadlo paketov a kontroluje sa či ich bolo spracované požadované množstvo.

Použité knižnice na spracovanie paketov

Na spracovanie paketov som použil nasledovné knižnice:

- `netinet/ether.h`
- `net/ethernet.h`
- `netinet/if_ether.h`
- `netinet/ip.h`
- `arpa/inet.h`
- `netinet/in.h`
- `netinet/udp.h`
- `netinet/tcp.h`
- `netinet/ip_icmp.h`
- `netinet/ip6.h`
- `netinet/icmp6.h`

Užitočné z tých knižníc boli hlavne štruktúry rámcov a funkcie ako `htonl`, `ntohs`

Spracovanie argumentov príkazového riadku

Na spracovanie argumentov som implementoval moduly `double_key_map` a triedu `cmd_params_c`.

Každému argumentu prislúcha bitová maska a funkcia. Napríklad pre parameter portu je maska `00000010`. Táto maska je v double key map priradená dvojici stringov, napríklad ("`-p`", "`--port`", `PORT_FLAG`). Takto sa cez `-p`, ale aj cez `--port` dostanem k bitovej maske(flagu parametru). Na základe flagu sa potom vyberá funkcia, ktorá správne inicializuje daný parameter a skontroluje či nie je konfliktný.

Trieda `cmd_params_c`, má na zodpovednosť aj inicializovanie filtru na rukováti. Používa na to `pcap_compile` a `pcap_setfilter`. Takto sa nepatričné rámce odfiltrujú ešte pred tým ako sa dostanú do spracovania paketov.

Spracovanie rámcov

Spracovanie sa začína vo funkcii `print_packet`. Tam sa vypíše timestamp a MAC adresy príjemcu a odosielateľa. Paket sa na základe typu posunie do ďalšej funkcie, dáta s ofsetom dĺžky ethernetovej hlavičky. Vetvenie ktoré funkcie sa postupne volajú je popísané v 3.1. Prakticky každá funkcia spracuje svoj header, vypíše dôležité informácie a ostatok rámca podá ďalšej. Funkcie na spracovanie ipv rámca a arp tiež vracajú dĺžku rámca, aby `print_packet` správne vypísal raw data packetu. Spracovanie a vyčítanie hlavičky prebieha, tak že binárne dáta sa pretypujú na zodpovedajúcu štruktúru. Zo štruktúry sa potom dáta čítajú ako z premennej, ľahko. V knižniciach sú definované aj konštanty, čo ktoré hodnoty znamenajú na rôznych miestach packetu. Toto využívam, vo funkciách sú na to switche, ktoré na záklde toho rozhodnú napríklad typ arp packetu. Na základe toho sa vypíše správa v ľudskej reči. Správy vypisované na konzolu sú definované v module `frame_msgs.h`.

Testovanie

Na testovanie som ako referenčný program používal **Wireshark**.

Prvá fáza testovania bola zameraná na oskúšanie, či je paket rozparsovaný správne. Na to som implementoval funkcionality, keď program načíta binárne dáta zo súboru a spracuje ich ako paket. Táto možnosť sa aktivuje `-b <file_name>` alebo `--binary <file_name>`. Takto sa načíta vždy jeden paket a toto nastavenie má vyššiu preferenciu ako ostatné. Teda filtre, alebo nastavenie rozhrania bude ignorované. Súbor s binárnymi paketmi sa získajú z Wiresharku ako hex stream, ktorý sa potom v nástroji hex to bin prerobí na binárny súbor. V druhej časti vývoja som sa zamerlal na testovanie reálneho zachytávania rámcov a ich porovnávanie s Wiresharkom. Porovnanie na obrázku nižšie.

```
Arrival Time: Apr 21, 2022 20:15:35.910211826 CEST
[Time shift for this packet: 0.000000000 seconds]
Epoch Time: 1650564935.910211826 seconds
[Time delta from previous captured frame: 0.000239886 seconds]
[Time delta from previous displayed frame: 0.992149747 seconds]
[Time since reference or first frame: 4.403724966 seconds]
Frame Number: 238
Frame Length: 98 bytes (784 bits)
Capture Length: 98 bytes (784 bits)
[Frame is marked: False]
[Frame is ignored: False]
[Protocols in frame: eth:ethertype:ip:icmp:data]
[Coloring Rule Name: ICMP]
[Coloring Rule String: icmp || icmpv6]
▶ Ethernet II, Src: IntelCor_f2:71:f5 (5c:87:9c:f2:71:f5), Dst: ArubaaHe_06:87:10 (00:1a:1e:06:87:10)
▶ Internet Protocol Version 4, Src: 100.64.199.51, Dst: 104.22.13.230
▼ Internet Control Message Protocol
  Type: 8 (Echo (ping) request)
  Code: 0
  Checksum: 0x08e9 [correct]
  [Checksum Status: Good]
  Identifier (BE): 6 (0x0006)
  Identifier (LE): 1536 (0x0600)
  Sequence number (BE): 89 (0x0059)
  Sequence number (LE): 22784 (0x5900)
```

Obr. 2: Rámec analyzovaný Wiresharkom

```

juraj@IT000-NTB19:/media/ntfs/Dokumentos/VUT/4.semester/IPK/projekt_2$ sudo ./sniffer -i wlo1 --icmp -n 1
[sudo] password for juraj:
Timestamp: 2022-04-21T20:15:35.910211+02:00
Src MAC: 5c:87:9c:f2:71:f5
Dst MAC: 0:1a:1e:6:87:10
Frame length: 84
Time to live: 64
Src IP: 100.64.199.51
Dst IP: 104.22.13.230
Header length: 20
Protocol: icmp
Sequence number: 22784
Type: Echo request (8)
Code: 0
0x0000: 00 1a 1e 06 87 10 5c 87 9c f2 71 f5 08 00 45 00 .....\\...q...E.
0x0010: 00 54 d8 f7 40 00 40 01 c0 41 64 40 c7 33 68 16 .T..@...Ad@.3h.
0x0020: 0d e6 08 00 08 e9 00 06 00 59 47 9f 61 62 00 00 .....YG.ab..
0x0030: 00 00 79 e3 0d 00 00 00 00 00 10 11 12 13 14 15 ..y.....
0x0040: 16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25 .....! "#$%
0x0050: 26 27 28 29          &' ( )

```

Obr. 3: Rámec analyzovaný snifferom