Síťový analyzátor - Varianta ZETA
Dokumentace

Lukáš Baštýř (xbasty00)

24. dubna 2021

Obsah

1 Úvod do problematiky **2**
1.1 argumenty 2
1.2 Protokoly na různých vrstvách 2

2 Implementace **3**
2.1 Důležité použité knihovny 3
2.2 Soubory projektu 3
2.3 Čtení argumentů (`manage_arguments`) 3
2.4 Výpis zařízení 3
2.5 Kontrola a otevření zařízení 4
2.6 Generace filtru (`get_filter`) 4
2.7 Čtení paketů 4
2.8 Zpracování hlavičky paketu (`on_packet`) 4
2.9 Zpracování protokolů (`got_protocol`) 5
2.10 Vytištění paketu (`print_packet`) 5

3 Testování **6**
3.1 IPv4 6
3.2 IPv6 6
3.3 ARP 6

1 Úvod do problematiky

Úkolem bylo vytvořit síťový analyzátor v `C/C++` nebo `C#`, který bude moct na daném síťovém rozhraní číst a filtrovat pakety.

1.1 argumenty

Možné argumenty programu:

- `-i jm.` - název síťového rozhraní, na kterém se má dělat analýza. Pokud není zadané jméno, vypíší se všechna síťová rozhraní
- `--udp, -u` - filtrování UDP paketů
- `--tcp, -t` - filtrování TCP paketů
- `--arp` - filtrování ARP paketů
- `--icmp` - filtrování ICMP nebo ICMPv6 paketů
- `-n č.` - počet paketů co se má zachytit
- `-p č.` - filtrování podle čísla portu

1.2 Protokoly na různých vrstvách

Jedním z problémů, který se musel vyřešit je fakt, že protokoly UDP[1], TCP[2], ARP[3] a ICMP(v6)[4] se nachází na různých vrstvách. ARP je protokol *odkazové vrstvy*[5]. ICMP je protokol *internetové vrstvy*[6]. TPC a UDP jsou oboje protkoly *transportní vrstvy*[7].

[1] *User Datagram Protocol* [online]. 2021 [cit. 2021-04-24]. Dostupné z: `http://en.wikipedia.org/wiki/User%20Datagram%20Protocol`.

[2] *Transmission Control Protocol* [online]. 2021 [cit. 2021-04-24]. Dostupné z: `http://en.wikipedia.org/wiki/Transmission%20Control%20Protocol`.

[3] *Address Resolution Protocol* [online]. 2021 [cit. 2021-04-24]. Dostupné z: `http://en.wikipedia.org/wiki/Address%20Resolution%20Protocol`.

[4] *Internet Control Message Protocol* [online]. 2021 [cit. 2021-04-24]. Dostupné z: `http://en.wikipedia.org/wiki/Internet%20Control%20Message%20Protocol`.

[5] `https://en.wikipedia.org/wiki/Link_layer`

[6] `https://en.wikipedia.org/wiki/Internet_layer`

[7] `https://en.wikipedia.org/wiki/Transport_layer`

2 Implementace

Projekt jsem se rozhodl implementovat v jazyce `C++11`.

2.1 Důležité použité knihovny

- `getopt.h`[8] - zpracování argumentů
- `pcap.h`[9] - Čtení a práce s pakety
- `arpa/inet.h` - Překlad IPv4 a IPv6 adres do lidem čitelné formy
- `time.h` - Výpis data a času ve formátu RFC3339[10]

2.2 Soubory projektu

Analyzátor je složen z následujících souborů

- `ipk-sniffer.cpp` - Hlavní soubor programu
- `ipk-sniffer.hpp` - Hlavičkový soubor obsahující prototypy struktur

2.3 Čtení argumentů (`manage_arguments`)

Pro čtení a zpracování argumentů jsem vytvořil pomocnou funkci `manage_arguments`. Ta pomocí knihovny `getopt` načte argumenty programu. Následně vytvoří strukturu `arguments_t arguments`, do které se uloží jednotlivé argumenty a jejich hodnoty. Pokud nebyl zadán žádný argument, nebo pouze `-i` bez hodnoty, vypíše se seznam aktivních zařízení. Pokud je zadáno rozhraní, uloží se jak rozhraní, tak hodnoty dalších argumentů do struktury. U číselných argumentů (`-n` a `-p`) se zkontroluje, že jejich hodnota obsahuje číslo (v opačném případě se program ukončí).

2.4 Výpis zařízení

Pro výpis zařízení je zde pomocná funkce `print_interfaces`, která najde všechna možná síťová zařízení (pomocí `pcap_findalldevs`), ale vypíše pouze aktivní.

[8] `https://man7.org/linux/man-pages/man3/getopt.3.html`

[9] V. JACOBSON C. Leres, S. McCanne. *libpcap* [online]. Lawrence Berkeley National Laboratory, University of California, Berkeley, CA, 2010 [cit. 2021-04-24]. Dostupné z: `https://www.tcpdump.org`.

[10] `https://tools.ietf.org/html/rfc3339`

2.5 Kontrola a otevření zařízení

Zkontroluje se, že hodnota argument `-i` je existující zařízení. O to se postará funkce `pcap_lookupnet`[11], která zkontroluje, že zadané zařízení opravdu existuje. Následně se toto zařízení otevře pro chytání paketů.

2.6 Generace filtru (`get_filter`)

Filtr je řetězec, který definuje jaké pakety a port se mají procházet a jaké vynechat. Ten se generuje pomocí funkce `get_filter`. Jedná se prakticky o stavový automat, který na základě argumentů TCP, UDP, ARP, ICMP a portu vytvoří řetězec se správnou syntaxí pro funkce `pcap_compile`[12] a `pcap_setfilter`[13], které vytvoří a aplikují filtrový program.

2.7 Čtení paketů

Čtení paketů se provádí pomocí funkce `pcap_loop`[14]. Pokud narazí na paket, zavolá funkci `on_packet`, která se postará o získání hlaviček.

2.8 Zpracování hlavičky paketu (`on_packet`)

Jako první chceme z paketu získat informace o zdrojovém a cílovém zařízení a o následujícím protokolu. K tomu slouží tzv. *ethernetová hlavička*, která se nachází v *ethernetovém rámci*[15] ("první"část paketu).

Ethernetová hlavička se získá ve funkci `on_packet` a to pomocí přetypování. Protože paket samotný je ve formátu `const u_char *` (čili řetězec), je možné jeho část *nahrát* do struktury. V našem případě se jedná o strukturu `ethernet_header *ethernet`, která bude následně obsahovat MAC[16] adresy zdrojového a cílového zaříze-ní a typ následujícího protokolu (IPv4, IPv6, ARP).

Pokud se jedná o ARP protokol, uloží se MAC adresy zařízení a pokračuje se na vypsání protokolu.

Jedná-li se o IPv4 nebo IPv6, uloží se IP adresy zařízení. Následně se zjistí typ dalšího protokolu (ICMP(v6), TCP, UDP).

Mimo jiné se v této funkci také vypíše čas, kdy byl paket získán.

[11] `https://www.tcpdump.org/manpages/pcap_lookupnet.3pcap.html`

[12] `https://www.tcpdump.org/manpages/pcap_compile.3pcap.html`

[13] `https://www.tcpdump.org/manpages/pcap_setfilter.3pcap.html`

[14] `https://www.tcpdump.org/manpages/pcap_loop.3pcap.html`

[15] *Ethernet frame* [online]. 2021 [cit. 2021-04-24]. Dostupné z: `http://en.wikipedia.org/wiki/Ethernet%20frame`.

[16] *MAC adresa* [online]. 2021 [cit. 2021-04-24]. Dostupné z: `https://cs.wikipedia.org/wiki/MAC_adresa`.

2.9 Zpracování protokolů (`got_protocol`)

Funkce `got_protocol` se stará o vypsání specifických informací k protokolum. U TCP a UDP se jedná o IP adresu a port zdrojového a cílového zařízení. U ICMP a ICMPv6 pouze IP adresa. U ARP se jedná pouze o MAC adresu.

2.10 Vytištění paketu (`print_packet`)

Jako poslední se vytiskne celý paket a to následovně.

Vypíšeme hexadecimální offset, kde se zrovna v paketu nacházíme (začátek = 0, konec = velikost paketu). Poté vypíšeme hexadecimální hodnotu jednotlivých oktetů (celkově 16 oktetů na řádek). Jako poslední přidáme jejich ASCII reprezentaci (pokud je to možné). Takto projdeme celý paket.

3 Testování

Pro testování jsem využil software `Wireshark`, který slouží k odchytávání a zobrazení paketů.

3.1 IPv4

Testování u IPv4 probíhalo jednoduše. Prostě jsem nechal tento program běžet společně s `Wiresharkem` a kontroloval, jestli je výstup obou programů stejný. Pokud ano tak všechno funguje správně. Toto ale platí hlavně na UDP a TCP pakety. ICMP pakety jsem otestoval pomocí nástroje *ping*, který vygeneruje ICMP paket. Výstupy jsem opět porovnal.

3.2 IPv6

Vzhledem k tomu, že nemám IPv6 adresu, neměl jsem zprvu jak otestovat tuto funkcionalitu. Čirou náhodo jsem ale dostal 1 IPv6 UDP paket. Výpis jsem porovnal s výstupem z programu `Wireshark` a všechno sedělo. Bohužel ale nemám jak otestovat TCP pakety. ICMPv6 pakety jsem musel testovat na lokálním síťovém zařízení `lo` následujícím příkazem `ping -6 ::1`.

3.3 ARP

Pro otestování ARP paketů stačilo chvíli počkat, než se router zeptá komu patří jaká IP adresa. Výsledek jsem opět porovnal s programem `Wireshark`.