VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Počítačové komunikace a sítě – 2. projekt

Varianta ZETA: Sniffer paketů

22. dubna 2021

Michal Pyšík (xpysik00)

Úvod

Sniffer paketů, nebo také analyzátor paketů, je program sloužící k zachycování (a případné analýze) jednotlivých paketů, sloužících ke komunikace v počítačové síti. Pakety se skládají z hlavičky, která se může dělit na více částí a obsahuje různé kontrolní informace, a samotných dat.

Různé implementace tohoto konceptu se liší především množstvím informací, které z daného balíčku extrahují. Tato konkrétní implementace zkoumá pouze typ protokolu, samotný protokol, adresy (a případně porty) odesílatele a příjemce, a samozřejmě také vypisuje celý obsah paketu. Pro více informací o správném užívání a omezení tohoto programu se prosím odkažte na přiložený soubor `README`.

Návrh aplikace

V této kapitole se zaměřuji na abstraktní pohled na rozdělení samotného programu na části (realizováno primárně funkcemi), vybrané konkrétní detaily implementace poté naleznete v následující kapitole.

Nejprve je samozřejmě nutné zpracovat argumenty od uživatele. Pokud vše proběhlo bez chyby, můžou na základě jeho rozhodnutí nastat 2 možnosti: Výpis všech dostupných aktivních rozhraní, nebo připojení na vybrané rozhraní, na kterém se bude odposlouchávat. Dále předpokládejme výběr druhé možnosti.

Pakety se zpracovávají postupně do doby, než je zpracován a vypsán cílový počet paketů. Nejprve se zkontroluje ethernet hlavička, pomocí které se rozhodne zda se jedná o IPv4, IPv6 nebo ARP paket (nebo jiné, ty se zahazují). U IPv4 a IPv6 se možnosti větví na TCP, UDP a ICMP(v6) paket. Pro každý paket který projde takovýmto filtrem, se zavolá daná rutina vyhovující protokolu. Každá rutina se skládá ze stejného výpisu data a času, výpis dat z hlavičky, a nakonec výpis samotných dat. Jediné co se v nich zpracovává jinak, je právě hlavička.

Jelikož se jedná o aplikaci která pracuje s ne-plně uživatelem předvídatelnými elementy (výskyt paketů na daném rozhraní), je velice důležitá možnost korektně ukončit program předčasně. Při ukončení uživatelem se tedy zachytí signál, který aktivuje rutinu pro uvolnění alokované paměti, notifikaci uživatele a ukončení programu.

Implementační detaily

Zpracování argumentů příkazové řádky probíhá pomocí knihovny `getopt` [4], včetně pomocné struktury na dlouhé verze argumentů. Jsou nastavené vhodné přepínače a globální proměnné, z nichž za zmínku stojí `char which_packets`, kde uvažujeme pouze spodní 4 bity a každý bit značí přítomnost argumentu s názvem vybraného protokolu.

Nalezené pakety jsou po jednom zpracovávány funkcí `void process_packet`, pro realizaci byla použita knihovna `pcap` [3]. Každý vypsaný paket dekrementuje počítadlo `int how_many_packets`, dokud není jeho hodnota 0. Postup na zpracování paketů je z velké části inspirován tímto řešením [6]. Z posledních 2 bytů ethernet hlavičky zjistíme, zda se jedná o IPv4, IPv6 nebo ARP paket V případě IP vyčteme z IP hlavičky také číselný identifikátor protokolu.

Pakety vybrané ke zpracování jsou předány funkci `process_TCP`, `process_UDP`, `process_ICMP` nebo `process_ARP` (všechny typu `void`). První 3 z jmenovaných funkcí podporují také nepovinný argument `bool IPv6_mode`, jehož implicitní hodnota bez uvedení tohoto paremetru je `false` a značí, zda se jedná o IPv4 nebo IPv6 paket, což je podstatná informace kvůli odlišným délkám hlaviček a formátu samotných IP adres. Každá z těchto funkcí ve svém těle volá na výpis data a času funkci `void print_RFC3339`, a pro výpis všech dat funkci `void print_raw_data`. Funkce se liší pouze zpracováním hlaviček samotného protokolu, ze kterého zjišťují (alespoň v případě TCP a UDP) porty příjemce a odesílatele. V případě zpracování ARP rámců se místo IP adres tisknou MAC adresy příjemce a odesílatele, přičemž uvažujeme vždy délku 6 bytů (jelikož jsou čteny přímo z ethernetové hlavičky).

Pro ošetření korektního ukončení programu při zásahu uživatele byla použita knihovna `signal.h` [1]. Při signálu ukončení je volána funkce `void on_user_interrupt`, která uživatele informuje o předčasném ukončení, uvolní všechnu alokovanou paměť a ukončí program s návratovou hodnotou 0 (jelikož se nejedná o chybu).

Testování

Otestovat bylo potřeba primárně 7 scénářů: TCP, UDP a ICMP(v6) pakety, z nichž každý pro IPv4 a IPv6, a nakonec ARP rámce. V této sekci najdete 2 konkrétní ukázky. Pro testování mi skvěle posloužil nástroj `nping` [5], a pro samotné ověření správné funkčnosti program `Wireshark` [2], jehož stylu výpisu samotných binárních dat jsem se snažil co nejvíce přiblížit.

Ukázka 1 – Identifikace a zpracování ICMPv6 paketu

```
michal@michal-VirtualBox:~$ sudo ping6 -c 1 -I enp0s3 fe80::32a9:5175:fc54:c9ac
PING fe80::32a9:5175:fc54:c9ac(fe80::32a9:5175:fc54:c9ac) from fe80::32a9:5175:fc54:c9ac%enp0s3 enp0s3: 56 data bytes
64 bytes from fe80::32a9:5175:fc54:c9ac%enp0s3: icmp_seq=1 ttl=64 time=0.044 ms

--- fe80::32a9:5175:fc54:c9ac ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.044/0.044/0.044/0.000 ms
```

Obrázek 1: Odeslání ICMPv6 paketu

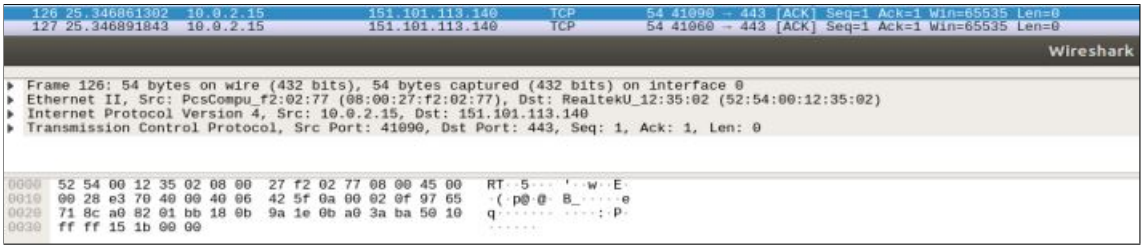
```
michal@michal-VirtualBox:~/Desktop/IPK_2$ sudo ./ipk-sniffer -i lo --icmp

2021-04-12T14:58:52.524+02:00 fe80::32a9:5175:fc54:c9ac > fe80::32a9:5175:fc54:c9ac, length 118 bytes

0x0000:  00 00 00 00 00 00 00 00  00 00 00 00 86 dd 60 0f  ........ ......`.
0x0010:  2b df 00 40 3a 40 fe 80  00 00 00 00 00 00 32 a9  +..@:@.. ......2.
0x0020:  51 75 fc 54 c9 ac fe 80  00 00 00 00 00 00 32 a9  Qu.T.... ......2.
0x0030:  51 75 fc 54 c9 ac 80 00  b2 8b 06 3f 00 01 0c 44  Qu.T.... ...?...D
0x0040:  74 60 00 00 00 00 ee ff  07 00 00 00 00 00 10 11  t`...... ........
0x0050:  12 13 14 15 16 17 18 19  1a 1b 1c 1d 1e 1f 20 21  ........ ...... !
0x0060:  22 23 24 25 26 27 28 29  2a 2b 2c 2d 2e 2f 30 31  "#$%&'() *+,-./01
0x0070:  32 33 34 35 36 37                                234567
```

Obrázek 2: Nalezení a výpis obsahu ICMPv6 paketu

Ukázka 2 – Porovnání výpisu IPv4 TCP paketu s programem Wireshark



Obrázek 3: Zobrazení daného IPv4 TCP paketu programem Wireshark

```
michal@michal-VirtualBox:~/Desktop/IPK_2$ sudo ./ipk-sniffer -i enp0s3 --tcp

2021-04-12T16:57:09.308+02:00 10.0.2.15 : 41090 > 151.101.113.140 : 443, length 54 bytes

0x0000:  52 54 00 12 35 02 08 00  27 f2 02 77 08 00 45 00  RT..5... '..w..E.
0x0010:  00 28 e3 70 40 00 40 06  42 5f 0a 00 02 0f 97 65  .(.p@.@. B_.....e
0x0020:  71 8c a0 82 01 bb 18 0b  9a 1e 0b a0 3a ba 50 10  q....... ....:.P.
0x0030:  ff ff 15 1b 00 00                                ......
```

Obrázek 4: Zobrazení daného IPv4 TCP paketu programem ipk-sniffer

Reference

[1] CGF: Pubs.opengroup.org: *¡signal.h¿*. [online], 2000, [vid. 2021-04-10].
Dostupné z: https://pubs.opengroup.org/onlinepubs/009695399/basedefs/signal.h.html

[2] Combs, G.; TEAM, T. W.: Wireshark. [Počítačový software], v3.4.4: 10. březen 2021.
Dostupné z: https://www.wireshark.org/

[3] Jacobson, V.; Leres, C.; McCanne, S.: Tcpdump.org: *Man page of PCAP*. [online], 29. leden 2021, [vid. 2021-04-08].
Dostupné z: https://www.tcpdump.org/manpages/pcap.3pcap.html

[4] Koenig, T.; Kerrisk, M.: Man7.org: *getopt(3) - Linux manual page*. [online], 1. duben 2021, [vid. 2021-04-08].
Dostupné z: https://www.gnu.org/software/libc/manual/html_node/Getopt.html

[5] Lyon, G. F.: Nping. [Počítačový software], v7.91: 9. říjen 2020.
Dostupné z: https://nmap.org/nping/

[6] MOON, S.: BinaryTides: *How to code a Packet Sniffer in C with Libpcap on Linux*. [online], 31. červenec 2020, [vid. 2021-04-9].
Dostupné z: https://www.binarytides.com/packet-sniffer-code-c-libpcap-linux-sockets/