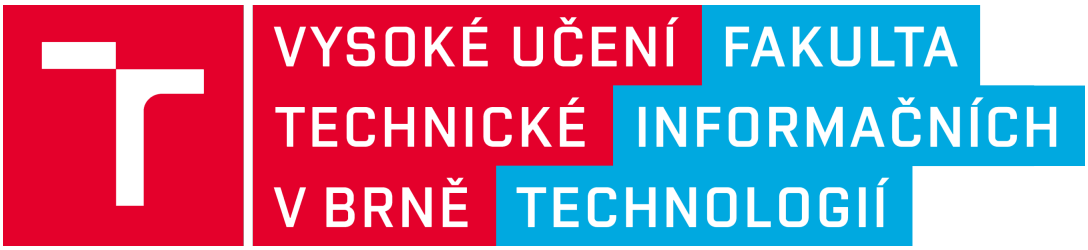


IPK 2020/2021 – Projekt 2

Varianta ZETA: Sniffer paketů

Vojtěch Fiala (xfiala61)

Obsah

1 Úvod

Cílem projektu bylo vytvoření síťového analyzátoru v jazyce `C` / `C++` / `C#`, který bude schopný na určitém síťovém rozhraním zachytávat a filtrovat pakety. Já si pro implementaci zvolil jazyk `C++`. Pro správné fungování je nutné program spouštět s root privilegii.

2 Implementace

Při tvorbě programu jsem využil návodu na programovaní s knihovnou pcap na stránkách projektu **TCPDUMP**[1]. Na stejné stránce je také k dispozici volně dostupný příklad síťového scanneru[2], který zachytává TCP pakety, z něhož jsem při tvorbě programu částečně vycházel. Jelikož jsem využil několika upravených funkcí z tohoto scanneru, jsou licenční podmínky tohoto scanneru uvedeny na začátku mého programu. Program pak také využívá následující volně dostupné knihovny:

- `iostream` – Výpis na standardní výstup
- `string` – Práce se stringy
- `time.h` – Konverze času
- `ctype.h` – Ověření, že je znak tisknutelný
- `pcap/pcap.h` – Odchytávání paketů, rozhraní
- `getopt.h` – Zpracování argumentů
- `arpa/inet.h` – Operace s IP adresami
- `netinet/if_ether.h` – Struktura Ethernet hlavičky
- `netinet/ip.h` – Struktura IP hlavičky
- `netinet/ip6.h` – Struktura IP v.6 hlavičky
- `netinet/tcp.h` – Struktura TCP paketu
- `netinet/udp.h` – Struktura UDP paketu

[1] `https://www.tcpdump.org/pcap.html`
[2] `https://www.tcpdump.org/sniffex.c`

- `netinet/ip_icmp.h` – Struktura ICMP paketu
- `netinet/icmp6.h` – Struktura ICMP paketu pro IP v.6 protokol
- `netinet/in.h` – Struktura IP adresy, konstanty definující protokoly

V případě, že program skončí nekorektně, vypíše na standardní chybový výstup k jaké chybě došlo a vrátí chybovou návratovou hodnotu. Tyto hodnoty jsou:

- `10` – Chyba při zpracovávání argumentů (Neplatný argument, neplatný parametr argumentu..)
- `11` – Chyba při otevírání rozhraní (rozhraní neexistuje, žádné nebylo nalezeno...)
- `12` – Chyba při tvorbě nebo aplikaci filtru.
- `15` – Neplatná velikost paketu.

2.1 Zpracování parametrů

Průběh fungování snifferu začíná zpracováním parametrů. K tomu je využita knihovna `getopt.h`. Při tvorbě způsobu zpracovávání parametrů jsem použil volně dostupné příklady z online stránek *Linux man-pages* projektu [7]. Při zpracování program zkontroluje, jestliže se za parametry, které to vyžadují, nachází argument. V případě nevalidního parametru či chybějícího parametru zahlásí program chybu, vypíše nápovědu a skončí. V případě neplatného parametru program pouze vypíše chybu. Nápovědu programu lze také vyvolat parametrem `--help` a nebo zkráceně `-h`. V případě že program narazí na tento parametr, nehledě na další parametry vypíše nápovědu a úspěšně skončí.

Při zpracování parametrů program využívá C++ struktury `std::string`, do kterých v případě zadání některého z argumentů pro filtrování (tcp, udp, arp, icmp) zapíše celý název protokolu pro tvorbu filtru. V případě zadání parametru portu je do string struktury obsahující protokol tcp či udp zapsáno i číslo portu. Jestliže byl zadán parametr icmp a/nebo arp a také parametr port, ale nikoliv parametry udp či tcp, je port přidán za pomoci logické spojky or na konec filtru. Pokud některý z filtrovacích parametrů nebyl zadán, v jeho odpovídající string struktuře je prázdný string `""`.

Následuje tvorba filtru. Ta probíhá tak, že jsou stringy se zadanými protokoly sloučeny do jednoho a mezi ně jsou vloženy logické spojky *or*. Výsledný filtr tedy může vypadat například takto: `(arp or (udp and port 42) or (tcp and port 42))`. Tento konkrétní filtr odpovídá následujícímu volání programu: `./ipk-sniffer -i enp0s25 --tcp --udp -p 22 --arp`.

V případě, že nebyl zadán parametr *-i* pro určení rozhraní či nebylo za ním zadáno rozhraní, jsou vypsána na standardní výstup všechna dostupná rozhraní a program úspěšně skončí. Pro tvorbu načtení a výpis rozhraní jsem využil příklad z dokumentace nástroje *WinPcap* [8].

2.2 Chytání paketů

Po zpracování parametrů následuje odchytávání paketů. To je inspirováno ukázkou z návodu na pcap programování na stránkách projektu TCPDUMP [1]. Odchytávání paketů začíná otevřením relace na daném rozhraní. Timeout pro jednotlivé pakety je nastaven na 1000ms aneb 1 vteřinu. Následuje zkonvertování filtru na funkcí požadovaný typ char* (inspirováno [4]). Zároveň probíhají kontroly, zdali nedošlo k chybě. Pokud ne, dostává se program k hlavní smyčce - funkci `pcap_loop`. Té je předána otevřená relace, počet paketů co má přečíst a callback funkce pro zpracování jednotlivých paketů. Po odchytnutí požadovaného množství paketů je relace uzavřena a program úspěšně končí.

2.3 Základní zpracování jednotlivých paketů

Callback funkce `read_packet` má k dispozici získaný paket a pcap hlavičku. Zpracování paketu začne zpracováním času, kdy byl paket získán. Tento čas je v UNIX formátu[3] a je proto potřeba ho zkonvertovat do člověku lépe chápatelné podoby. Ze zadání jsem pochopil, že se má po přepočtu zobrazit desetinná část pouze na 3 místa, takže je počet mikrosekund vydělen celočíselně číslem 1000. Následuje zkonvertování času na lépe zpracovatelnou strukturu `time_t`, která je poté přeformátována na string a vypsána na standardní výstup.

Při konvertování času jsem se inspiroval [3]. Také jsem využil dokumentaci funkce `strftime` [2] pro získání formátovacích znaků.

[3] `https://en.wikipedia.org/wiki/Unix_time`

Po zpracování času program vytvoří ethernetovou hlavičku (Struktura typu `ether_header`). Z té vyčte protokol, na základě kterého určí, jak postupovat dále. Jestliže se jedná o protokol IP, IPV6 nebo ARP, dochází k dalšímu zpracování. Jestliže se jedná o jiný protokol, program vypíše akorát velikost paketu a jeho obsah.

2.4 Výpis obsahu paketu

Výpis obsahu paketu probíhá skrz funkci `print_packet_content`, která zpracovává v cyklu obsah paketu po 16 bytech. Jestliže zbývá vypsat méně než 16 bytů, je cyklus následně ukončen. Samotný výpis probíhá skrz funkci `print_packet_line`, která vypisuje jednotlivé byty v hexadecimalním formátu a také vhodně doplňuje mezery mezi ně. Také zajišťuje výpis ASCII hodnoty bytu, je-li to možné. Jestliže aktuální byt nemá žádný tisknutelný odpovídající ASCII znak, je nahrazen tečkou. Výpis obsahu paketu je inspirován ukázkovým souborem `sniffex.c`, který je dostupný na stránkách projektu TCPDUMP[4]. Jak již bylo zmíněno v sekci 2, licenční podmínky tohoto ukázkového programu jsou vepsány na začátek mého programu.

2.5 Zpracování protokolu IP

Pokud je protokol, kterým byl přijat paket, typu IP, vytvoří si program strukturu typu `ip`, ze které jednoduše zjistí podprotokol paketu, délku IP headeru a také IP adresu odesílatele i příjemce. IP adresa odesílatele je ihned vypsána na standardní výstup. Je také vypočítán offset, na kterém se nachází další podprotokol. Ten je spočítán jako 4x délka IP protokol hlavičky. Na tuto hodnotu jsem přišel analýzou paketu přijatého přes IP protokol v programu `Wireshark`[5]. Na základě získaného podprotokolu (TCP, UDP, ICMP) poté rozhodne, jak se zachová dále. Jestliže je paket jiného, neznámého protokolu, je vypsána IP adresa příjemce, velikost a obsah paketu. Jinak dojde k dalšímu zpracování paketu na základě získaného podprotokolu.

[4] `https://www.tcpdump.org/sniffex.c`

2.5.1 Zpracování protokolů TCP a UDP

Pakety o podprotokolu typu TCP nebo UDP jsou zpracovávány obdobně – slouží k tomu funkce `read_tcp_udp`, která si v závislosti na typu paketu vytvoří odpovídající strukturu, ze které vyčte port odesílatele a port příjemce. Ty jsou poté, společně s IP adresou příjemce, vypsány na standardní výstup. Dále je také vypsána velikost a obsah paketu.

2.5.2 Zpracování protokolu ICMP

Jelikož protokol ICMP neobsahuje údaje o portech, v případě jeho zachycení je vypsána pouze IP adresa odesílatele a IP adresa příjemce. Dále je také vypsána velikost a obsah.

2.6 Zpracování protokolu IP6

Jestliže byl protokol paketu IP verze 6, je vytvořena struktura typu `ip6_hdr`, která představuje IP6 hlavičku. Z ní je získána IPv6 adresa odesílatele i příjemce. K tomu je použita funkce `inet_ntop` [5]. Dále je spočítán offset podprotokolů. Jelikož struktura `ip6_hdr` neobsahuje celkovou velikost hlavičky, je offset ručně dopočítáván. Na základě analýzy paketu o protokolu IP6 v programu Wireshark[5] jsem se dozvěděl, že velikost obsahu paketu je uložena na bytech 18 a 19 v IP6 hlavičce. Hodnota těchto bytů je tedy sečtena a dále odečtena od celkové velikosti paketu, od které je také odečtena velikost Ethernet headeru. Výsledek je offset, na kterém se nachází hlavička podprotokolu.

Po zjištění offsetu je určen typ podprotokolu. Hodnoty, podle kterých program určuje jednotlivé typy podprotokolů, jsem nalezl na stránkách organizace IANA [6]. Podprotokoly mohou být typu ICMP6, TCP nebo UDP. Zpracování paketů dále probíhá obdobně jako v případě IP protokolu. Jediným rozdílem je, že vypisované IP adresy jsou typu IPv6.

2.7 Zpracování protokolu ARP

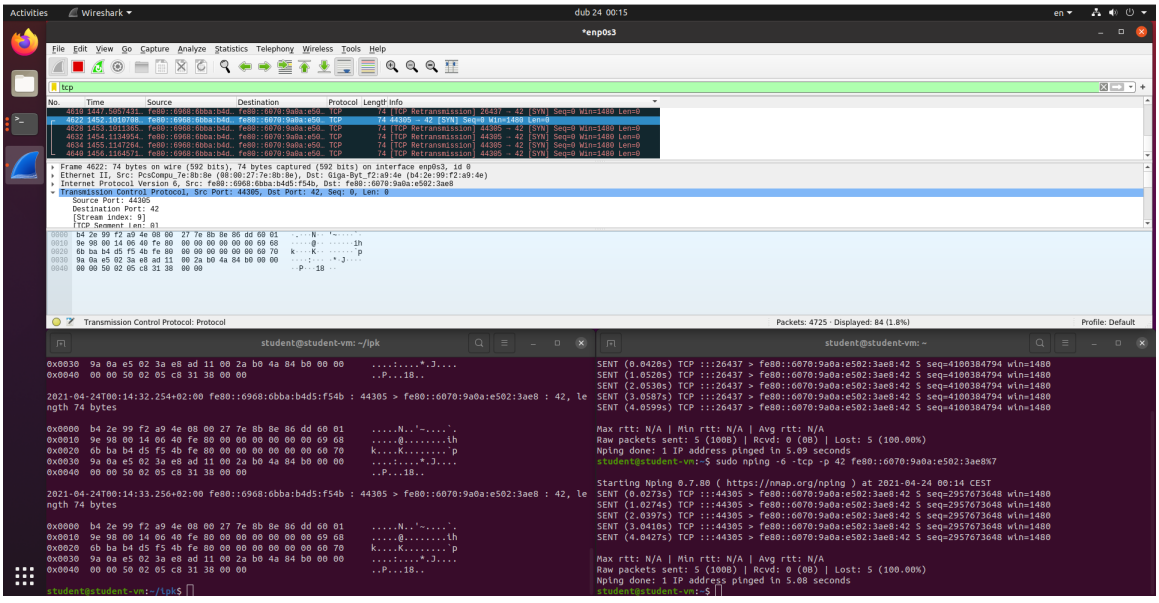
V případě, že byl odchytnutý protokol typu ARP, je jeho zpracování poněkud odlišné od předchozích. Program nevypisuje port, ale pouze MAC adresu odesílatele

[5] `https://www.wireshark.org/`

a příjemce. Tyto adresy jsou získány čtením odpovídajících bytů v paketu. Z analýzy paketu o protokolu ARP v již zmíněném programu Wireshark jsem zjistil, že byty na pozici 22-27 obsahují MAC adresu odesílatele a byty na pozici 32-37 MAC adresu přijemce. Tyto adresy jsou tedy vypsány, společně s délkou a obsahem paketu, na standardní výstup a program pokračuje dále.

3 Testování

Testování probíhalo na poskytnutém referenčním virtuálním stroji s OS GNU/Linux a distribucí Ubuntu. Probíhalo tak, že jsem za pomoci nástroje `nping`[6] posílal TCP, UDP a ICMP pakety jak přes IPv4, tak přes IPv6 na počítač, na kterém virtuální stroj běžel. Stejným způsobem jsem také posílal ARP pakety. Tyto pakety jsem zachytával skrz OpenSource nástroj `Wireshark`[7], kde jsem si zobrazil jejich parametry (IP adresy u TCP...) a jejich obsah a tyto údaje porovnával s obsahem paketů, které zachytil můj program. Jak testování probíhalo lze vidět na obrázku níže.



Obrázek 1: Zachytávání IPv6 TCP paketů na konkrétním portu

[6] `https://man7.org/linux/man-pages/man1/nping.1.html`

[7] `https://www.wireshark.org/`

Literatura

[1] Carstens, T.: Programming with pcapTCPDUMP/LIBPCAP public repository. [Online], 2002, [Cit. 21. 04. 2021]. URL: https://www.tcpdump.org/pcap.html

[2] cppreference.com: std::strftime - cppreference.com. [Online], 2020, [Cit. 21. 04. 2021]. URL: https://en.cppreference.com/w/cpp/chrono/c/strftime

[3] https://stackoverflow.com/users/1741542/olaf-dietsche: Unix-Time to readable date. [Stack Overflow], 23. 11. 2012, https://stackoverflow.com/questions/13536886/unixtime-to-readable-date [Cit. 21. 04. 2021]. URL: https://stackoverflow.com/a/13536943

[4] https://stackoverflow.com/users/3969405/hairlessbear: std::string to char*. [Stack Overflow], 8.9.2011, https://stackoverflow.com/questions/7352099/stdstring-to-char [Cit. 21. 04. 2021]. URL: https://stackoverflow.com/a/42308974/

[5] IBM Corporation: Help - Eclipse SDK. [Online], 2007, [Cit. 22. 04. 2021]. URL: http://www.qnx.com/developers/docs/6.5.0/index.jsp?topic=%2Fcom.qnx.doc.neutrino_lib_ref%2Fi%2Finet_ntop.html

[6] Internet Assigned Numbers Authority: Protocol Numbers. [Online], 2021, [Cit. 22. 04. 2021]. URL: https://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml

[7] Koenig, T.: getopt(3) — Linux manual page. [Online], 2021, [Cit. 20. 04. 2021]. URL: https://man7.org/linux/man-pages/man3/getopt.3.html

[8] Politecnico di Torino: WinPcap: Obtaining the device list. [Online], 2009, [Cit. 20. 04. 2021]. URL: https://www.winpcap.org/docs/docs_412/html/group__wpcap__tut1.html