Vysoké učení technické v Brně

Fakulta informačních technologií


# Počítačové komunikace a sítě

2019/2020

# Dokumentace projektu

**Varianta Zeta – Sniffer paketů**

Daniel Pátek (xpatek08) Brno, 1. května 2020

# 1. Obsah

# 2. Úvod

Cílem řešeného projektu byl návrh a implementace síťového analyzátoru. Tento analyzátor bude schopný na určitém síťovém rozhraní zachytávat a filtrovat pakety. Také bylo nutné vytvořit k projektu dokumentaci.
Program podporuje zachytávání paketů na protokolu TPC a UDP. Déle podporuje filtrování vypisovaných paketů podle protokolu, portu a rozhraní. Pakety se vypisují na `stdout` s oddělenou hlavičkou. Před vypsáním paketu je vytvořen úvodní řádek, který obsahuje přesný systémový čas zachycení paketu, IP adresy a porty zdroje a cíle.

# 3. Implementace

## 1. Základní údaje

K implementaci jsem použil knihovnu `Libpcap`. Jedná se o open source knihovnu jazyka C, která poskytuje API pro zachycení paketů z datových linek. Funguje na všech Unix systémech.
Samotný kód je uložen v jednom souboru `sniffer.c`. Co se týče vložených hlaviček – `pcap.h` poskytuje funkce a konstanty a hlavičky od `netinet` a `arpa.h` má potřebné datové struktury pro práci s pakety. Program obsahuje dvě globální proměnné. První slouží jako deskriptor pro knihovnu `pcap.h` a ve druhé je uložena délka hlavičky transportní vrstvy paketů.
Celý program je rozložen na šest hlavních funkcí, které jsou popsány níže.

## 2. Funkce main()

Hlavní část programu, funkce `main()`, má za úkol zpracování argumentů programu, vytvoření řetězce pro filtrování a následné spuštění snifferu.
Součástí zpracování argumentů je i funkce `printHelp()`, jež vytiskne nápovědu k použití programu, a funkce `printInterfaces()`, která má za úkol v případě nespecifikování žádného rozhraní, na kterém chceme zachytávat pakety, vypsat všechny možné rozhraní.
Dále funkce kontroluje, zda jsou zadány filtry zároveň na TCP i na UDP komunikaci. V takovém případě skončí s chybovým hlášením.
S pomocí funkce `signal()` nastavuje, jak má program správně skončit v případě přerušení systémem nebo uživatelem.

# 3. Funkce open_pcap_socket()

Funkce `main()` volá funkci `open_pcap_socket()`.
Tato funkce slouží zejména k získání soket deskriptoru pro sniffer paketů. Tento soket bude sloužit jako koncový bod při práci s pakety v datové vrstvě.
Tato funkce je posloupností pěti dalších funkcí knihovny pcap.h. Nejprve se zkontroluje, zda funkce má k dispozici rozhraní definované uživatelem. Pokud ne, pomocí funkce `pcap_lookupdev()` získá první dostupné. Tato funkce by v programu neměla být nikdy spuštěna, slouží pouze jako pojistka pro případné další použití této funkce.
Po kontrole je přistoupeno k funkci `pcap_open_live()`, která otevře vybrané rozhraní pro síťové přenosy a vrátí potřebný deskriptor.
Dále je zapotřebí aplikovat filtr. K tomu slouží následující trojice knihovních funkcí. `Pcap_lookupnet()` nám získá masku sítě našeho připojení pro následující funkci. `Pcap_compile()` převede námi vytvořený řetězec filtrů na knihovnou interpretovatelný kód, který může být aplikován pomocí třetí funkce `pcap_setfilter()`.
Tím je náš filtr použit přímo na deskriptoru síťového soketu pro sniffer paketů.

```
// convert the filter expression into a packet filter code
// we will get the bpf struct, used for filtering, based on our filter expression
if (pcap_compile(pcap_descriptor, &bpf, (char *) filter_expression, 0, netmask)) {
    printf("pcap_compile(): %s\n", pcap_geterr(pcap_descriptor));
    return NULL;
}

// assign the packet filter to the given libpcap socket
if (pcap_setfilter(pcap_descriptor, &bpf) < 0) {
    printf("pcap_setfilter(): %s\n", pcap_geterr(pcap_descriptor));
    return NULL;
}
```

*Obrázek 1 - Aplikace řetězcového filtru*

# 4. Funkce start_capture()

Tato funkce je také volána z funkce `main()`. Slouží především k zjištění typu přenosu datové linky. Podle toho, o jaký typ se jedná, je následně definována délka hlavičky datové vrstvy. Tato informace je uložena v globální proměnné `header_length`.
Následně je z této funkce volána `pcap_loop()`. Ta slouží k samotnému „sniffování“ paketů. Jako argumenty si bere již zmíněný deskriptor, uživatelem definovaný počet paketů k zobrazení a taky ukazatel na funkci, která zajišťuje analýzu každého paketu.

# 5. Funkce parse_packet()

Tato funkce zajišťuje analýzu příchozího paketu. Má za úkol vypsat úvodní řádek dle zadání projektu a následně spustit funkci pro výpis surových dat.
Ze všeho nejdříve tato funkce zjistí aktuální čas s vysokou přesností tak, aby se co nejvíce blížila času zachycení paketu.
Následně je určen typ IP komunikace. Program podporuje komunikaci přes IPv4 nebo IPv6. Pro každou možnost program provádí analýzu jinou cestou.
V případě IPv4 se pomocí funkce `inet_ntoa()` určí IP adresa cíle i zdroje paketu, dále se z hlavičky zjistí, o jaký typ protokolu se jedná a vypíše se úvodní řádek. Oproti zadání zde mám jednu změnu, kdy v úvodním řádku vytisknu i typ protokolu, který se analyzuje.
V případě IPv6 se obě IP adresy zjistí pomocí funkce `inet_ntop()`. Oproti IPv4 je zapotřebí zjistit, zda má paket rozšířenou hlavičku. V knihovní struktuře hlavičky se jedná o parametr `ip6_nxt`. Pokračuje se zjištěním typu protokolu jako v předchozím případě.
Po vypsání úvodního řádku se volá funkce pro vypsání dat z paketu.

# 6. Funkce printData()

Jedná se o funkci, která vypisuje data z paketu po jednotlivých bajtech. Vypisuje jak v hexadecimální, tak v ASCII podobě.
Funkci tvoří hlavní cyklus, který zajišťuje výpis všech bajtů. Velkou roli zde hraje logická proměnná `headPrinted`. Díky ní se může v pořádku oddělit hlavička a pokračovat ve výpisu bez narušení formátu paketu.

```
// if it finished the hexa line or head is printing
if (headPrinted ? ((i - afterHeaderOffset) % 16 == 0 && i != headSize) :
(i != 0 && i % 16 == 0)) {
    // continue to print characters (print dot if char is not printable)
    printf(" ");
    for (j = i - 16; j < i; j++) {
        if (isprint(packet[j])) {
            printf("%c", (unsigned char) packet[j]);
        }
        else {
            printf(".");
        }
        // print the space between bytes
        if (j == i - 9) {
            printf(" ");
        }
    }
```

*Obrázek 2 - Zajištění vypsání dat ve formátu ASCII*

Funkce dále zajišťuje také vypsání počtu bajtů na začátku každého řádku.
Také se stará o vypisování mezer mezi bajty i mezi jednotlivými částmi výpisu.

```
20:27:20.295145 UDP 10.0.0.9 : 5353 > 224.0.0.251 : 5353

0x0000:  01 00 5E 00 00 FB A8 9C  ED F9 1E 3C 08 00 45 00  ..^..... ...<..E.
0x0010:  00 7A 87 56 40 00 FF 11  09 18 0A 00 00 09 E0 00  .z.V@... ........
0x0020:  00 FB 14 E9 14 E9 00 66  E7 3E                    .......f .>

0x0030:  00 03 00 00 00 02 00 00  00 00 00 00 2A 5F 25 39  ........ ....*_%9
0x0040:  45 35 45 37 43 38 46 34  37 39 38 39 35 32 36 43  E5E7C8F4 7989526C
0x0050:  39 42 43 44 39 35 44 32  34 30 38 34 46 36 46 30  9BCD95D2 4084F6F0
0x0060:  42 32 37 43 35 45 44 04  5F 73 75 62 0B 5F 67 6F  B27C5ED. _sub._go
0x0070:  6F 67 6C 65 63 61 73 74  04 5F 74 63 70 05 6C 6F  oglecast ._tcp.lo
0x0080:  63 61 6C 00 00 0C 00 01  C0 3C 00 0C 00 01        cal..... .<....
```

*Obrázek 3 - Příklad vypsání UDP paketu*

# 7. Funkce clear()

Tato funkce se volá při ukončení programu, ať už kvůli chybě, systémovému přerušení, uživatelskému přerušení nebo vypsání zadaného počtu paketů. Používá strukturu `pcap_stat` za účelem vypsání statistik o odchycených paketech. Důležité je uzavření soketu pro síťové přenosy.

```c
void clear() {
    // stats of packets
    struct pcap_stat stats;

    // print the stats (if some exist)
    if (pcap_stats(pcap_descriptor, &stats) >= 0) {
        printf("%d packets received\n", stats.ps_recv);
        printf("%d packets dropped\n", stats.ps_drop);
    }

    // close the pcap descriptor
    pcap_close(pcap_descriptor);
    exit(EXIT_SUCCESS);
}
```

*Obrázek 4 - Funkce pro ukončení programu*

# 4. Příklady výstupu programu

```
:~/Plocha/sniffer-master$ sudo ./sniffer -i wlp5s0 -n 1 -t -p 80
20:29:46.979298 TCP 10.0.0.25 : 33152 > 77.75.75.172 : 80

0x0000:  C8 D1 2A 09 49 DF 00 1C  26 1B 6C A3 08 00 45 00  ..*.I... &.l...E.
0x0010:  00 3C D3 3B 40 00 40 06  C4 70 0A 00 00 19 4D 4B  .<.;@.@. .p....MK
0x0020:  4B AC 81 80 00 50 49 55  94 9A 00 00 00 00 A0 02  K....PIU ........
0x0030:  FA F0 7F 2F 00 00 02 04  05 B4 04 02 08 0A DD 5B  .../.... .......[
0x0040:  ED B3 00 00 00 00 01 03  03 07                    ........ ..


1 packets received
0 packets dropped
 1    :~/Plocha/sniffer-master$
```

*Obrázek 7 - Příklad spuštění programu*

```
20:28:09.735653 TCP 10.0.0.25 : 52584 > 77.75.75.176 : 80

0x0000:  C8 D1 2A 09 49 DF 00 1C  26 1B 6C A3 08 00 45 00  ..*.I... &.l...E.
0x0010:  00 7D 3A 10 40 00 40 06  5D 57 0A 00 00 19 4D 4B  .}:.@.@. ]W....MK
0x0020:  4B B0 CD 68 00 50 C1 F8  6F 43 39 80 4B 6E 80 18  K..h.P.. oC9.Kn..
0x0030:  01 F6 D3 CF 00 00 01 01  08 0A FE 7D CE 87 31 C5  ........ ...}..1.
0x0040:  4F 53                                            OS

0x0050:  47 45 54 20 2F 20 48 54  54 50 2F 31 2E 31 0D 0A  GET / HT TP/1.1..
0x0060:  48 6F 73 74 3A 20 73 65  7A 6E 61 6D 2E 63 7A 0D  Host: se znam.cz.
0x0070:  0A 55 73 65 72 2D 41 67  65 6E 74 3A 20 63 75 72  .User-Ag ent: cur
0x0080:  6C 2F 37 2E 35 38 2E 30  0D 0A 41 63 63 65 70 74  l/7.58.0 ..Accept
0x0090:  3A 20 2A 2F 2A 0D 0A 0D  0A                       : */*... .
```

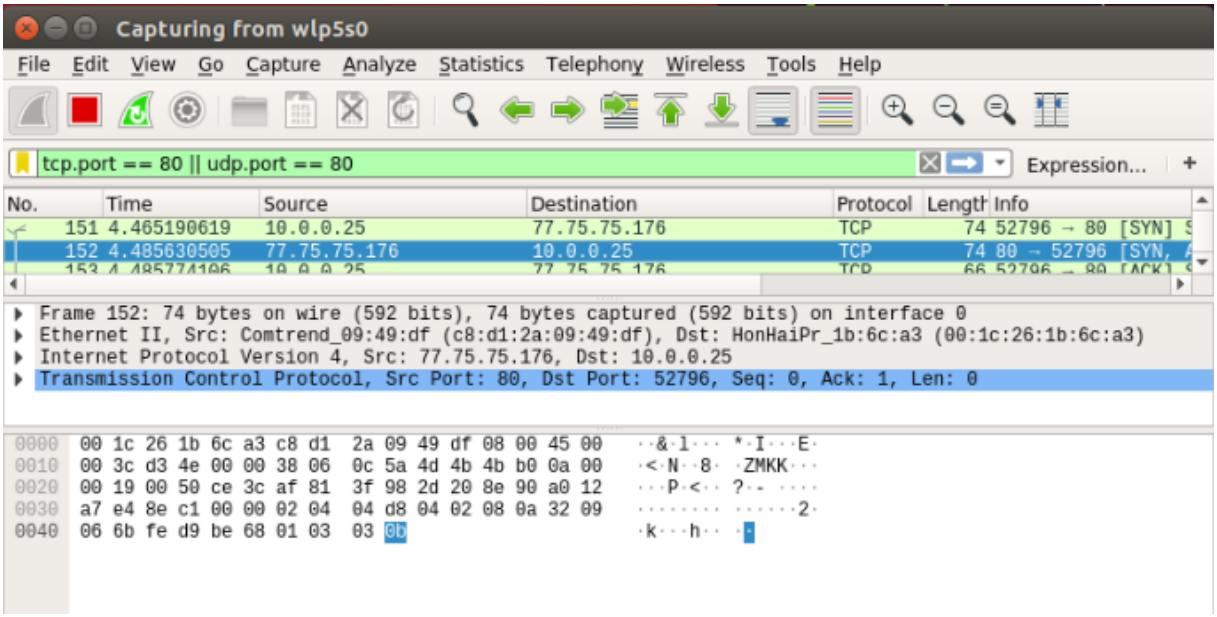
*Obrázek 6 - Příklad paketu s HTTP komunikací*

```
20:28:04.059154 UDP fe80::7ed6:61ff:feb0:9991 : 5353 > ff02::fb : 5353

0x0000:  33 33 00 00 00 FB 7C D6  61 B0 99 91 86 DD 60 0F  33....|. a.....`.
0x0010:  54 56 00 FD 11 FF FE 80  00 00 00 00 00 00 7E D6  TV...... ......~.
0x0020:  61 FF FE B0 99 91 FF 02  00 00 00 00 00 00 00 00  a....... ........
0x0030:  00 00 00 00 00 FB 14 E9  14 E9 00 FD 77 7F        ........ ....w.

0x0040:  00 00 84 00 00 00 00 04  00 00 00 03 02 31 30 01  ........ .....10.
0x0050:  30 01 30 02 31 30 07 69  6E 2D 61 64 64 72 04 61  0.0.10.i n-addr.a
0x0060:  72 70 61 00 00 0C 80 01  00 00 00 78 00 11 09 41  rpa..... ...x...A
0x0070:  6E 64 72 6F 69 64 2D 32  05 6C 6F 63 61 6C 00 01  ndroid-2 .local..
0x0080:  31 01 39 01 39 01 39 01  30 01 42 01 45 01 46 01  1.9.9.9. 0.B.E.F.
0x0090:  46 01 46 01 31 01 36 01  36 01 44 01 45 01 37 01  F.F.1.6. 6.D.E.7.
0x0100:  30 01 30 01 30 01 30 01  30 01 30 01 30 01 30 01  0.0.0.0. 0.0.0.0.
0x0110:  30 01 30 01 30 01 30 01  30 01 38 01 45 01 46 03  0.0.0.0. 0.8.E.F.
0x0120:  69 70 36 C0 1E 00 0C 80  01 00 00 00 78 00 02 C0  ip6..... ....x...
0x0130:  2E C0 2E 00 01 80 01 00  00 00 78 00 04 0A 00 00  ........ ..x.....
0x0140:  0A C0 2E 00 1C 80 01 00  00 00 78 00 10 FE 80 00  ........ ..x.....
0x0150:  00 00 00 00 00 7E D6 61  FF FE B0 99 91 C0 0C 00  .....~.a ........
0x0160:  2F 80 01 00 00 00 78 00  06 C0 0C 00 02 00 08 C0  /.....x. ........
0x0170:  3F 00 2F 80 01 00 00 00  78 00 06 C0 3F 00 02 00  ?./..... x...?...
0x0180:  08 C0 2E 00 2F 80 01 00  00 00 78 00 08 C0 2E 00  ..../... ..x.....
0x0190:  04 40 00 00 08                                   .@...
```

*Obrázek 5 - příklad UDP paketu s IP adresami verze 6*

# 5. Testování programu

Program jsem testoval metodou porovnávání výstupu s ověřeným zdrojem.
Pro generování testovacích paketů jsem využil příkaz `curl` případně `curl -6` pro IPv6 verzi.
Jako ověřený zdroj považuji program `wireshark`. Zde jsem si nastavil stejné filtrování jako v mém programu a následně mohl porovnat výstupy obou programů.




*Obrázek 9 - Program wireshark použitý k testování*

```
:~/Plocha/sniffer-master$ sudo ./sniffer -i wlp5s0 -n 2
 -p 80
22:08:34.923285 TCP 10.0.0.25 : 52796 > 77.75.75.176 : 80

0x0000:  C8 D1 2A 09 49 DF 00 1C  26 1B 6C A3 08 00 45 00  ..*.I... &.l...E.
0x0010:  00 3C 36 5F 40 00 40 06  61 49 0A 00 00 19 4D 4B  .<6_@.@. aI....MK
0x0020:  4B B0 CE 3C 00 50 2D 20  8E 8F 00 00 00 00 A0 02  K..<.P-  ........
0x0030:  FA F0 62 7C 00 00 02 04  05 B4 04 02 08 0A FE D9  ..b|.... ........
0x0040:  BE 68 00 00 00 00 01 03  03 07                    .h...... ..


22:08:34.935196 TCP 77.75.75.176 : 80 > 10.0.0.25 : 52796

0x0000:  00 1C 26 1B 6C A3 C8 D1  2A 09 49 DF 08 00 45 00  ..&.l... *.I...E.
0x0010:  00 3C D3 4E 00 00 38 06  0C 5A 4D 4B 4B B0 0A 00  .<.N..8. .ZMKK...
0x0020:  00 19 00 50 CE 3C AF 81  3F 98 2D 20 8E 90 A0 12  ...P.<.. ?.- ....
0x0030:  A7 E4 8E C1 00 00 02 04  04 D8 04 02 08 0A 32 09  ........ ......2.
0x0040:  06 6B FE D9 BE 68 01 03  03 0B                    .k...h.. ..
```

*Obrázek 8 - Výstup mého programu je shodný s výstupem z programu wireshark*

# 6. Zdroje

Při zpracovávání projektu jsem využil pouze internetové zdroje.
Níže se nachází seznam všech zdrojů, ze kterých jsem čerpal.


1. pcap_findalldevs example. *Embedded Guru* [online]. Dostupné z: http://embeddedguruji.blogspot.com/2014/01/pcapfindalldevs-example.html
2. GitHub - yuan901202/ethernet_packet_sniffer: [NWEN302] Ethernet Packet Sniffer in C. *The world's leading software development platform · GitHub* [online]. Copyright © 2020 GitHub, Inc. [cit. 02.05.2020]. Dostupné z: https://github.com/yuan901202/ethernet_packet_sniffer
3. TCPDUMP/LIBPCAP public repository. *TCPDUMP/LIBPCAP public repository* [online]. Dostupné z: https://www.tcpdump.org/
4. WinPcap · Documentation. *WinPcap - Home* [online]. Copyright © 2018 [cit. 02.05.2020]. Dostupné z: https://www.winpcap.org/docs/default.htm
5. arpa_inet.h.0p - Linux manual page. *Michael Kerrisk - man7.org* [online]. Dostupné z: http://man7.org/linux/man-pages/man0/arpa_inet.h.0p.html
6. <netinet/in.h>. *The Open Group Publications Catalog* [online]. Copyright © 1997 The Open Group [cit. 02.05.2020]. Dostupné z: https://pubs.opengroup.org/onlinepubs/007908799/xns/netinetin.h.html
7. c - Problems finding pcap.h and linking - Stack Overflow. *Stack Overflow - Where Developers Learn, Share, & Build Careers* [online]. Dostupné z: https://stackoverflow.com/questions/13337349/problems-finding-pcap-h-and-linking
8. pcap_findalldevs(3): list of capture devices - Linux man page. *Linux Documentation* [online]. Dostupné z: https://linux.die.net/man/3/pcap_findalldevs
9. pcap-linktype(7) - Linux man page. *Linux Documentation* [online]. Dostupné z: https://linux.die.net/man/7/pcap-linktype
10. inet_ntop(3) - Linux manual page. *Michael Kerrisk - man7.org* [online]. Dostupné z: http://man7.org/linux/man-pages/man3/inet_ntop.3.html
11. header - What is the size of udp packets if i send 0 payload data in c#? - Stack Overflow. *Stack Overflow - Where Developers Learn, Share, & Build Careers* [online]. Dostupné z: https://stackoverflow.com/questions/4218553/what-is-the-size-of-udp-packets-if-i-send-0-payload-data-in-c

# 7. Závěr

Projekt byl pro mě velmi přínosný. Naučil jsem se, jakým způsobem odchytávat pakety z internetové komunikace. Také jsem se dozvěděl, co to paket vlastně je, jakým způsobem se s sním pracuje a co všechno obsahuje.
Rozšířil jsem si své znalosti také v problematice internetových protokolů transportní i síťové vrstvy. Zejména způsob komunikace využívající protokol UDP nebo TCP.
Zjistil jsem také, co všechno obsahuje hlavička paketu a jak ji odlišit od dalších dat v paketu.
V neposlední řadě jsem získal zkušenosti s programem wireshark a naučil se programovat s využitím knihovny Libpcap.
Dokázal jsem dodržet všechny podmínky ze zadání projektu. Program je funkční a otestovaný. Jediné, co se liší od vzorového výstupu ze zadání, je již zmíněné vypsaní typu protokolu v úvodním řádku paketu.