VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Projektová dokumentácia k predmetu IPK

Packet sniffer (varianta ZETA)

26. dubna 2020 Dominik Boboš (xbobos00)

Obsah

1 Úvod

Zadaním projektu bolo vytvorenie programu sieťového analyzátoru, ktorý zachytáva a filtruje na určitom sieťovom rozhraní prichádzajúce a odchádzajúce TCP a UDP pakety.

Packet sniffer, nazývaný aj paketový analyzátor pozostáva z dvoch hlavných častí. Prvou je sieťový adaptér, ktorý pripája program k existujúcej sieti. Druhá časť uskutočňuje analýzu samotných zachytených paketov, v našom prípade zisťuje *(*čas), kedy bol TCP alebo UDP paket zachytený, *IP adresu* zdroja a cieľa a taktiež aj *port* zdroja a cieľa[1].

2 Spustenie programu

Program je kompatibilný s linuxovými systémami a takisto aj so systémom macOS 10.13+. K správnej kompilácií je vhodné disponovať prekladačom `gcc 7.5.0` a vyššie. Takisto je potrebný program make, testované na verzií `GNU Make 4.1`.
V priečinku projektu sa nachádza Makefile, ktorý umožní projekt zostaviť použitím:

```
$ make
```

Pri zostavovaní projektu dochádza na referenčnom stroji k warningu o nepoužívaní hodnoty parametru *args* na platforme macOS sa tento warning nevyskytoval, avšak nijako nebráni k správnej činnosti programu.
Vyčistenie zkompilovaného programu ipk-sniffer je možné pomocou:

```
$ make clean
```

Projekt sa spúšta pomocou:

```
$ ./ipk-sniffer -i rozhranie [-p port] [-tcp|-t] [-udp|-u] [-n num]
```

Pokiaľ nie je možné projekt spustiť je potrebné mu poskytnúť administrátorské práva: `$ sudo ./ipk-sniffer -i rozhranie [-p port] [-tcp|-t] [-udp|-u] [-n num]`

- `-i <rozhranie>` - určuje rozhranie, na ktorom bude program pracovať, v prípade, že sa tento argument nevyskytuje, vypíše sa zoznam dostupných rozhraní. V prípade chýbajúceho parametru <*rozhranie*> sa vypíše nápoveda a skončí sa s hodnotou **1**.
- `-p <port>` - voliteľný parameter, filtruje pakety na danom rozhraní podľa portu <*port*>, parameter port môže obsahovať najviac 9 čísel, inak sa program ukončí s hodnotou 1.
- `-tcp | -t` - voliteľný parameter, budú zobrazované len TCP pakety
- `-udp | -u` - voliteľný parameter, budú zobrazované len UDP pakety
- `-n <num>` - určuje počet zachytených paketov. V prípade, že sa argument nevyskytuje, zobrazí sa 1 paket.

Nápovedu je možné zobraziť pomocou prepínaču `-h`, prípadne nesprávnym zadaním iného argumentu. V prípade chybných argumentov alebo bola vyžiadaná nápoveda, program skončí s návratovou hodnotou **1**. V prípade úspechu vráti hodnotu **0**. V prípade zlyhania súčastí knižnice PCAP vráti hodnotu **10**.

3 Použité knižnice

V programe je použitých mnoho knižníc, dajú sa rozdeliť na dve kategórie.

3.1 Esenciálne pre jazyk C

Prvou kategóriou sú potrebné knižnice pre podporu funkcií jazyka C:

- `<stdio.h>, <stdlib.h>, <stdbool.h>, <string.h>` - štandardné funkcie ako *malloc*, práca s reťazcami.
- `<signal.h>` - na zachytenie signálu ukončenia programu pomocou `CTRL+C`.
- `<ctype.h>` - pre funkciu *isprint(..)*.
- `<getopt.h>` - na spracovanie argumentov príkazového riadku.
- `<time.h>,<sys/types.h>` - na správnu prácu s časom.

3.2 Esenciálne pre prácu so sieťovými prvkami

Druhou kategóriou sú knižnice potrebné pre pripojenie sa k sieťovému adaptéru, alebo k používaniu štruktúr paketov:

- `<netdb.h>` funkcie getnameinfo(...) pre nájdenie FQDN a makrá ako NI_MAXHOST.
- `<arpa/inet.h>` funkcie *inet_pton(...)*, *inet_pton(...)*, *inet_pton(...)* pre prácu s IP adresami IPv4, IPv6.
- `<pcap.h>` funkcie z pcap knižnice slúžiace k zostaveniu sieťového adaptéra, ktorý sa pripojí k existujúcemu pripojeniu a taktiež k zachytávaniu paketov.
- `<netinet/ip.h>, <netinet/ip6.h>` - štruktúry hlavičiek IPv4 a IPv6 paketov.
- `<netinet/tcp.h>, <netinet/udp.h>` - štrukúry hlavičiek TCP a UDP paketov.
- `<netinet/if_ether.h>` - štruktúra ethernetovej hlavičky.

4 Implementácia

Program je implementovaný v jazyku **C** v súbore `ipk-sniffer.c`. Ipk-sniffer.c je rozdelený do niekoľkých funkcií. Na začiatku sa do premenných načítajú vstupné argumenty uvedené v sekcií 2 pomocou funkcie *args_parse(int argc, char *argv[], char *iface, char *port, int *pnum, int *tcp, int *udp)*. Jednotlivé časti sú v komentári kódu ozdrojované.

4.1 Zostavenie sieťového adaptéru

Implementácia sieťového adaptéru prípajajúceho sa na existujúcu sieť je v hlavnom tele *int main(int argc, char *argv[])* využívajú funkcie knižnice *pcap.h*. Na začiatku je potrebné nastaviť interface, ten je poskytnutý užívateľom vďaka argumentu *-i* a je uložený do premennej *char *iface*, prípadne je možné zobraziť zoznam dostupných zariadení ak tento parameter vynecháme (program skončí s hodnotou **0**).

Potom možeme priradiť zariadeniu masku podsiete *bpf_u_int32 pMask* a ip adresu *bpf_u_int32 pNet* prostredníctvom funkcie *pcap_lookupnet*. Spoločne sa jej predáva aj zariadenie, na ktorom pracujeme. V prípade chyby sa vypíše chybová hláška uložená v *errbuf* a ukončí sa program s hodnotou **10**.

Následne je možné otvoriť zariadenie k zachytávaniu paketov, k tomu slúži funkcia *pcap_open_live(iface, BUFSIZ, 0, 1000, errbuf)* a hodnota z funkcie sa uloží do *pcap_t *opensniff*, v prípade chyby je vypísaná chybová hláška a program končí s hodnotou **10**. Hodnota **0** vypína *promiskuitný režim*, avšak aj tak sa môže stať, že v špecifických prípadoch ostane zapnutý (zavisí aj od platformy, na ktorej sa program používa)[9].

Teraz je možné zostaviť filter *char filter[50])*, ktorý sa zostavuje na základe používateľom zadaných argumentov. V prípade, že nebol explicitne zadaný požiadavok na filtráciu, implicitne sa nastavuje filter prepúšťajúci len UDP a TCP pakety. Funkciou *pcap_compile* s parametrami *pcap_compile(opensniff, &fp, filter, 0, pNet)* skompilujeme náš adaptér a následne ho môžeme aplikovať pomocou *pcap_setfilter(opensniff, &fp)*, obidve funkcie v prípade chyby vypíšu hlášku a ukončia program s návratovou chybou **10** [4].

Takto zostavaný adaptér teraz môžeme pomocou *pcap_loop(opensniff, pnum, callback, NULL)* uviesť do „nekonečného cyklu“, kedy sa zachytávajú pakety v počte *pnum* (*implicitne pnum = 1*) a pri každom zachytenom pakete sa volá funkcia *callback* [3].

4.2 Práca a analýza s paketmi

Funkcia *callback(u_char *args, const struct pcap_pkthdr* pkthdr,const u_char* buffer)* spracováva každý zachytený paket. Pomocou `struct ether_header *p` sa zisťuje či daný paket používa **IPv4**, alebo **IPv6**, v prípade, že *ntohs(p->ether_type) == ETHERTYPE_IPV6* nastaví sa `bool ipv6 = true;`. Timestamp paketu - teda čas, kedy bol paket zachytený je uložený v *pkthdr->ts.tv_sec* a mikro sekundy v *pkthdr->ts.tv_usec*, pomocou funkcie *localtime(...)* sa správne od Unixovej epochy vypočíta čas. Program je možné kedykoľvek ukončiť pomocou **CTRL+C**. V prípade chyby je program ukončený s návratovou hodnotou *20*.

Dôležitá štruktúra v projekte je `struct pckt_info`:

```
struct pckt_info
{          // velkosť bufferu 1025 odpovedá NI_MAXHOST
   char src_addr[1025]; // obsahuje zdrojová IP/FQDN
   char dest_addr[1025]; // obsahuje cieľová IP or FQDN
   unsigned src_port;  // obsahuje zdrojový PORT
   unsigned dest_port; // obsahuje cieľový PORT
   int header_size;  // velkosť hlavičky celého paketu v [B]
};
```

Ukladajú sa do nej potrebné dáta, ktoré sa budú neskôr vypisovať na štadardný výstup. Kedže TCP a UDP majú rozdielne hlavičky [8], je potrebné z hlavičky IP zistiť akého protokolu sú nositeľom. Podľa hodnoty uloženej z IP do premennej *int tcp_udp_switch* je možné zavolať konkrétne funkcie pre jednotlivé protokoly. K zisťovaniu **FQDN** teda k zisťovaniu presne stanovenému menu domény [10] slúži funkcia:

```
char *host_name(struct in\_addr ip\_addr) // pre IPv4 adresy
char *host_nameIPv6(struct in6\_addr ip\_addr) // pre IPv6 adresy
```

K nájdeniu mena využíva funkciu `getnameinfo`. Návratová hodnota funkcie *host_name | *host_nameIPv6 je v prípade nájdenia FQDN *char *node*, v prípade neúspechu vráti IP adresu v *char *ip*, v prípade chyby vráti funkcia hodnotu `NULL`. Funkcia vznikla modifikáciou funkcie z [2].

4.2.1 Spracovanie IP

Spracovanie prebieha v UDP aj TCP rovnako. Pomocou štruktúr *struct* `ip` | *struct* `ip6_hdr` je možné zistiť IP adresu zdroja a cieľa.

```
  //pričítame velkosť hlavičky ethernetu
struct ip6_hdr *iph = (struct ip6_hdr *)(buffer + sizeof(struct ether_header));
struct ip *iph = (struct ip *)(buffer + sizeof(struct ether_header));
```

Tieto hodnoty sa ďalej posielajú do funkcie *host_name* prípadne *host_nameIPv6*, ktorých výstup sa ukladá do `char *temp_src; char *temp_dest;`. Ďalej je možné zo štruktúry struct *struct* `ip` zistiť veľkosť IPv4 (nakoľko môže byť v rozmedzí od 20 do 60 bajtov, *slide 10*[7]). Táto hodnota sa nachádza v `iph->ip_hl*4` (aby sme získali počet bajtov) a ukladá sa do *int* `iphdr_len`. Pri IPv6 je veľkosť stálych 40 bajtov (*slide 4*[6]).

4.2.2 Spracovanie TCP

Funkcia k analýze TCP hlavičky. V premennej *buffer* sa nachádza celý paket, *ipv6* je rovná `true` v prípade, že paket obsahuje IPv6 protokol.

```
struct pckt\_info tcp\_packet(const u\_char *buffer, bool ipv6)
```

Spracovanie IP adries prebieha ako je uvedené v sekcií **??** a hodnoty `temp_src` a `temp_dest` sa ukladajú do `header.src_addr` a `header.dest_addr`.

Pomocou štruktúry:

```
struct tcphdr *tcph = (struct tcphdr*)(buffer + iphdr_len + sizeof(struct
    ether_header));
```

Zisťujeme port zdroja a cieľa a ukladá sa do:

```
  header.src_port = ntohs(tcph->th_sport);
  header.dest_port = ntohs(tcph->th_dport);
```

Pomocou **tcph* vieme zistiť velkosť tcp paketu v bajtoch, vďaka `tcph->th_off*4`. Keď sčítame celkové tieto veľkosti hlavičiek získame celkovú veľkosť hlavičky paketu, ktorá sa uloží do *header.header_size*. Funkcia vráti celú premennú *header*. V prípade chyby sa do premennej header.header_size uloží hodnota -1.

4.2.3 Spracovanie UDP

Spracovanie je obdobné s funkciou k analýze TCP hlavičky, jediný rozdiel je v používanej štruktúre. V premennej *buffer* sa nachádza celý paket, *ipv6* je rovná `true` v prípade, že paket obsahuje IPv6 protokol.

```
struct pckt\_info tcp\_packet(const u\_char *buffer, bool ipv6)
```

Spracovanie IP adries prebieha ako je uvedené v sekcií spracovanie IP 4.2.1 a Spracovanie TCP 4.2.2.

Pomocou štruktúry:

```
struct udphdr *udph = (struct udphdr*)(buffer + iphdr_len + sizeof(struct
    ether_header));
```

Zisťujeme port zdroja a cieľa a ukladá sa do:

```
header.src_port = ntohs(udph->uh_sport);
header.dest_port = ntohs(udph->uh_dport);
```

Keď sčítame celkové veľkosti hlavičiek získame celkovú veľkosť hlavičky paketu, ktorá sa uloží do *header.header_size*. Funkcia vráti celú premennú *header*. V prípade chyby sa do premennej header.header_size uloží hodnota -1. Funkcie *callback, udp_packet, tcp_packet* sú modifikáciami kódu od Faraz Fallahi [5].

4.3 Výpis výstupu

Výstup dát zabezpečuje funkcia,

```
void print_data(char *time, long microsec, struct pckt_info packet_info,
        const unsigned data_len, const u_char *data)
```

ktorá je volaná z funkcie *callback*. Dáta vypisuje na štandardný výstup podobne ako je v zadaní a obdobne s opensource softvérom *Wireshark*. Výpis je uskutočňovaný po riadkoch v cykle `for`.

```
_____________________________________UDP_____________________________________
21:32:08.461717 192.168.0.103 : 54719 > resolver3.opendns.com : 53

0x0010 98 de d0 ad 9e d4 04 0c  ce e1 33 f4 08 00 45 00  ._Q..U.. Ob.u....
0x0020 00 48 63 00 00 00 ff 11  e8 74 c0 a8 00 67 d0 43  ........ i.A)..Q.
0x002a de dc d5 bf 00 35 00 34  96 ec                    _]V@.... .m

0x003a 00 00 00 00 00 00 03 31  30 33 01 30 03 31 36 38  ........ ........
0x004a 03 31 39 32 07 69 6e 2d  61 64 64 72 04 61 72 70  ........ ........
0x0056 61 00 00 0c 00 01                                 ......
------------------------------------------------------------------------------
```

Prvý riadok je orámovanie paketu s informáciou o aký paket sa jedná. Na ďalšom riadku je timestamp paketu s presnosťou na mikro sekundy, nasleduje IP adresa alebo FQDN zdroja, za dvojbodkou je port zdroja, za znakom > sa nachádza IP adresa alebo FQDN cieľa, za dvojbodkou je port cieľa.

Nasleduje prázdny riadok a následne sa uskutočňuje výpis jednotlivých bajtov paketu v hexadecimálnej sústave. To znamená, že dvojica takýchto čísel sú 2 bajty.

Prvý stĺpec značí počet dovtedy vypísaných bajtov + počet, ktorý sa vypíše na danom riadku v hexadecimálnom formáte a správne číslo sa stará hodnota uložená v `end_case`.

V druhom stĺpci sa nachádza maximálne 16 bajtov, ktoré sa píšu po dvojiciach a pre lepšiu prehladnosť je stĺpec po ôsmich bajtoch obohatený o jednu medzeru naviac.

V treťom stĺpci je výpis tých istých bajtov, čo v druhom stĺpci avšak sú vypísané tlačiteľnými znakmi ASCII, netlačiteľné sú pomocou funkcie *isprint* nahradené bodkou. O doplnenie medzier pre správne zarovnanie sa stará funkcia `void add_space(int count)`.

Hlavička paketu je od zvyšných dát oddelná medzerou. Posledný riadok slúži opäť ako orámovanie a oddelenie od ostatných zachytených paketov.

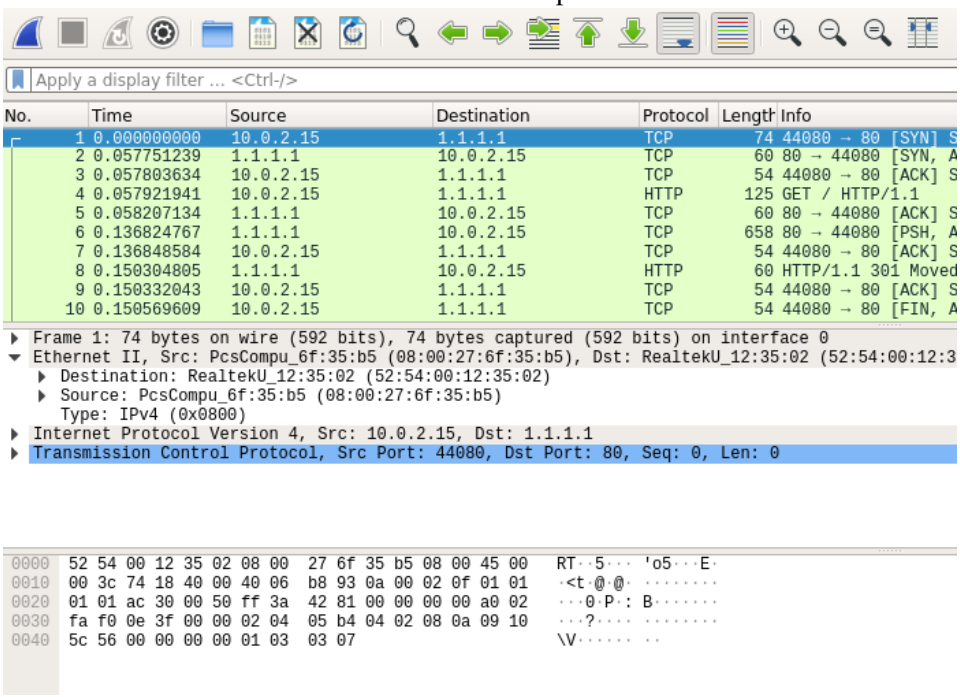
5 Testovanie

Testovanie prebiehalo na referenčnom stroji `PDS-VM` so systémom `Ubuntu 18.1` a taktiež aj na systéme macOS High Sierra 10.13 . Testovanie prebiehalo pomocou súbežne spusteného open source softvéru **`Wireshark`** a projektu **`ipk-sniffer`**. Následne sa buď spustil internetový prehliadač a porovnal sa rovnaký paket, alebo sa zachytával paket odoslaný programom `Curl`.

Paket odoslamý programom Curl:

```
student@student-vm:~$ curl 1.1.1.1
<html>
<head><title>301 Moved Permanently</title></head>
<body bgcolor="white">
<center><h1>301 Moved Permanently</h1></center>
<hr><center>cloudflare-lb</center>
</body>
</html>
student@student-vm:~$
```

Obrázek 1: Paket s IPv4 protokolom



Obrázek 2: Paket vygenerovaný programom Curl zachytený vo Wireshark

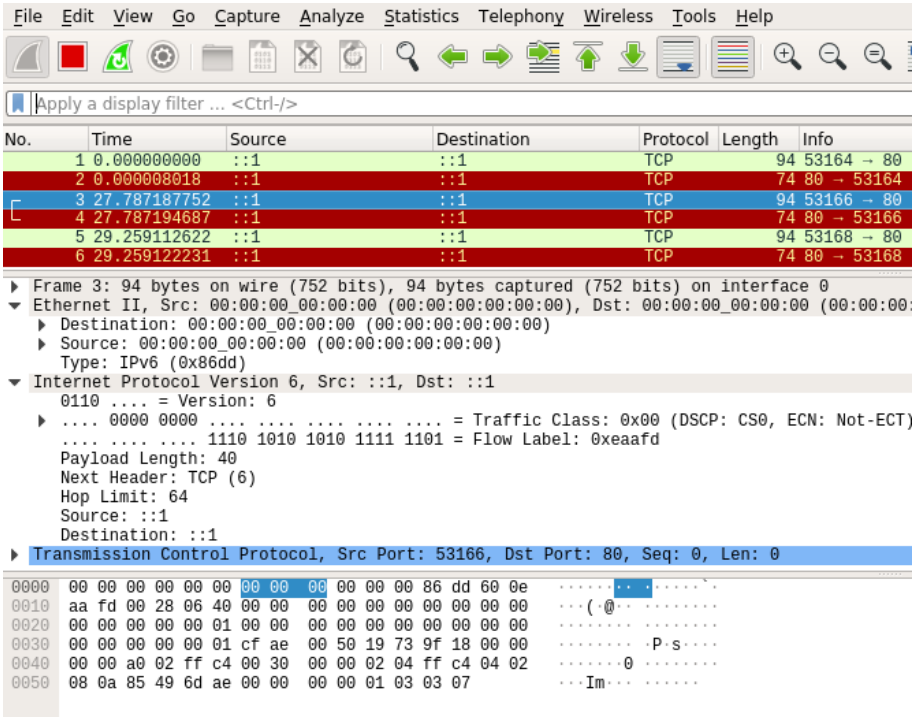
```
student@student-vm:~/Desktop/VUT-FIT-IPK2$ sudo ./ipk-sniffer -i enp0s3 -n 3
                                        TCP
22:58:02.581972 student-vm : 44080 > one.one.one.one : 80

0x0010 52 54 00 12 35 02 08 00  27 6f 35 b5 08 00 45 00  RT..5... 'o5...E.
0x0020 00 3c 74 18 40 00 40 06  b8 93 0a 00 02 0f 01 01  .<t.@.@. ........
0x0030 01 01 ac 30 00 50 ff 3a  42 81 00 00 00 00 a0 02  ...0.P.: B.......
0x0040 fa f0 0e 3f 00 00 02 04  05 b4 04 02 08 0a 09 10  ...?.... ........
0x004a 5c 56 00 00 00 00 01 03  03 07                    \V...... ..
-------------------------------------------------------------------------------
```

Obrázek 3: Paket vygenerovaný programom Curl zachytený ipk-sniffer

Paket s IPv6 vygenerovaný programom Curl, bohužiaľ jediný spôsob ako bolo možné otestovať podporu IPv6:

```
student@student-vm:~$ curl -g -6 "http://[::1]:80/"
curl: (7) Failed to connect to ::1 port 80: Connection refused
student@student-vm:~$
```

Obrázek 4: Paket s IPv6 protokolom



Obrázek 5: Paket vygenerovaný programom Curl zachytený vo Wireshark

```
student@student-vm:~/Desktop/VUT-FIT-IPK2$ sudo ./ipk-sniffer -i lo -n 3
_____________________________________TCP_____________________________________
23:04:15.895876 ip6-localhost : 53166 > ip6-localhost : 80

0x0010 00 00 00 00 00 00 00 00  00 00 00 00 86 dd 60 0e  ........ ......`.
0x0020 aa fd 00 28 06 40 00 00  00 00 00 00 00 00 00 00  ...(.@.. ........
0x0030 00 00 00 00 00 01 00 00  00 00 00 00 00 00 00 00  ........ ........
0x0040 00 00 00 00 00 01 cf ae  00 50 19 73 9f 18 00 00  ........ .P.s....
0x0050 00 00 a0 02 ff c4 00 30  00 00 02 04 ff c4 04 02  .......0 ........
0x005e 08 0a 85 49 6d ae 00 00  00 00 01 03 03 07        ...Im... ......
------------------------------------------------------------------------------
```

Obrázek 6: Paket vygenerovaný programom Curl zachytený ipk-sniffer

6 Použité zdroje

Použitá literatúra

[1] AG, P.: IT Explained: Packet Sniffing. [online], rev. 2020, [vid. 2020-04-25]. Dostupné z: https://www.paessler.com/it-explained/packet-sniffing

[2] ALGORISM: getnameinfo() example problem. [online], rev. 3. júl 2016, [vid. 2020-04-23]. Dostupné z: https://cboard.cprogramming.com/c-programming/169902-getnameinfo-example-problem.html

[3] ARORA, H.: How to Perform Packet Sniffing Using Libpcap with C Example Code. [online], 2002, [vid. 2020-04-25]. Dostupné z: https://www.thegeekstuff.com/2012/10/packet-sniffing-using-libpcap/

[4] CARSTENS, T.: Programming with pcap. [online], rev. 25. október 2012, [vid. 2020-04-25]. Dostupné z: https://www.tcpdump.org/pcap.html

[5] Fallahi, F.: lsniffer.c. [online], rev. 6. apríla 2020, [vid. 2020-04-20]. Dostupné z: https://gist.github.com/fffaraz/7f9971463558e9ea9545

[6] VESELÝ, V.: IPv6 Sťová vrstva. [Univerzitná prednáška], 2017.

[7] VESELÝ, V.: Sťová vrstva. [Univerzitná prednáška], 2018.

[8] VESELÝ, V.: Transportní vrstva. [Univerzitná prednáška], 2018.

[9] Wikipedia: Promiscuous mode. [online], rev. 23. november 2019, [vid. 2020-04-25]. Dostupné z: https://en.wikipedia.org/wiki/Promiscuous_mode

[10] Wikipedia: Fully qualified domain name. [online], rev. 28. január 2020, [vid. 2020-04-25]. Dostupné z: https://en.wikipedia.org/wiki/Fully_qualified_domain_name