VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Počítačové komunikace a sítě – 2. projekt

Sniffer paketů

2. května 2020 Alexandr Chalupnik (xchalu15)

Obsah

1 Úvod do problematiky

Cílem projektu bylo vytvořit síťový analyzátor, který bude schopný na určitém síťovém rozhraním zachytávat a filtrovat pakety.

2 Implementace

Program byl implementován v jazyce C++ za použití knihovny *PCAP*, konkrétně implementaci pro unixové systémy *libpcap*. Knihovna poskytuje vysoko úrovňové rozhraní pro systémy sledující provoz v síti[1], jednou z mnoha aplikací, která využívá rozhraní *PCAP* je aplikace Wireshark.

Program tvoří soubor `ipk-sniffer.cpp` s implementací a hlavičkový soubor `ipk-sniffer.h` s definicemi funkcí a struktur. Projekt je dokumentovaný tak, aby se dala případně vygenerovat Doxygen dokumentace.

2.1 Zpracování argumentů

Před samotným sledováním sítě proběhne kontrola parametrů zadaných na příkazovou řádku a jejich uložení do struktury `Params`. Každý parametr lze zadat pouze jednou, pokud se tak nestane, jde o chybu. Program v případě potřeby vypisuje nápovědu pomocí parametrů `-h` nebo `--help`, tento parametr nelze kombinovat s jinými parametry.

2.2 Nastavení programu

Po kontrole parametrů se volá funkce `sniff`, která přebírá parametry a zajišťuje nastavení relace pro sledování paketů. Funkce `pcap_open_live` zajišťuje otevření námi zadaného rozhraní, při snaze otevřít neplatné rozhraní se vypíše chybová hláška. Po zpřístupnění rozhraní dojde k nastavení filtru pomocí funkcí `pcap_setfilter`, naším filtrem je je právě kombinace parametrů `--tcp, --udp` a `-p` (port). Pokud vše proběhne správně může nastat samotné zachytávaní paketů, to zajišťuje funkce `pcap_loop`, které je předán ukazatel na na callback funkci `process_packet`, která má na starost zpracování paketu[3].

2.3 Zpracování paketů

Paket je ve funkci `process_packet` reprezentován jako ukazatel na `u_char`. V první řadě přetypujeme tento ukazatel na strukturu `ether_header`, která reprezentuje hlavičku ethernetového rámce.

```
1    eth_h = (ether_header*) (packet);
```

Díky tomu jsme schopni přes parametr `type` zjistit jaký protokol ze síťové vrstvy je zapouzdřen do rámce.

```
1    if(ntohs(eth_h->ether_type) == ETHERTYPE_IPV6){
2    ...
3    }
```

Pokud by se jednalo o protokol IPv4, byla by zde uložena hexadecimální hodnota `0x0800`, která je dostupná přes makro `ETHERTYPE_IP`. Pro protokol IPv6 by šlo o hexadecimální hodnotu `0x86DD` a makro `ETHERTYPE_IPV6`[8]. Pro zpřístupnění obsahu IP datagramu je potřeba ukazatel na paket opět přetypovat. Ukázka pro protokol IPv6:

```
1    ip6_h = (ip6_hdr*) (packet + ETH_HLEN);
```

Ukazatel `packet`, musíme nejdříve posunou o posunou o 14 bytů, tzn. o velikost hlavičky ethernetového rámce. Poté dojde k přetypování na strukturu `ip6_hdr`, která reprezentuje hlavičku IPv6 datagramu. Hlavička IPv6 datagramu obsahuje 8-bitovou položku `Next header`, v které je informace o transportním protokolu, který je v něm zapouzdřen (např.: 6 pro TCP, 17 pro UDP)[7]. V případě IPv4 hlavičky, je informace o transportním protokolu uložena v položce `Protocol`[5].

```
    if(ip4_h->protocol == IPPROTO_TCP){
    ...
    }else if(ip4_h->protocol == IPPROTO_UDP){
    ...
    }
```

Jakmile je jasné jaký protokol transportní vrstvy je zapouzdřen v paketu, můžeme k němu přistoupit opět pomocí přetypování ukazatele `packet`. Adresa ukazatele se posouvá o velikost ethernetové hlavičky v bytech, která je konstantní `ETH_HLEN` a o velikost hlavičky IPv4 která je variabilní, a dá se zjistit z parametru `IHL`. V tomto parametru je uložen počet 4 bytových slov, nejmenším počtem slov je 5, tzn. že hlavička neobsahuje položku `Option`. Jelikož je proměnná `ihl` struktury `ip4_h` pouze počet slov, musí se vynásobit velikostí jednoho slova (32 bitů = 4 byty)[5]. Příklad přetypování na strukturu `tcp_h` reprezentující hlavičku TCP segmentu.

```
tcp_h = (tcphdr*) (packet + ETH_HLEN + ip4_h->ihl*4);
```

Pokud je například TCP segment zapouzdřen v hlavičce IPv6 datagramu, stačí ukazatel `packet` posunout o velikost ethernetové hlavičky a velikost IPv6 hlavičky, která je konstantní (40 Bytů).

```
tcp_h = (tcphdr*) (packet + ETH_HLEN + IPV6_HLEN);
```

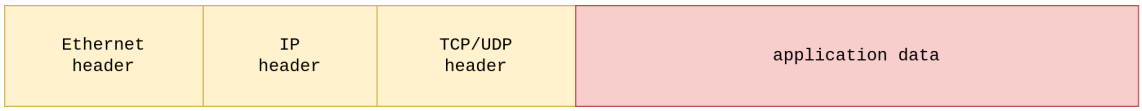
Z hlaviček TCP/UDP protokolu se následně zjistí cílový a zdrojový port (z položek `Source port` a `Destination Port`[6][4]). Ukázka získání cílového a zdrojového portu z hlavičky TCP segmentu zapouzdřené do IPv4 datagramu:

```
    if(ip4_h->protocol == IPPROTO_TCP){
        tcp_h = (tcphdr*) (packet + ETH_HLEN + ip4_h->ihl*4);
        sport = ntohs(tcp_h->th_sport);
        dport = ntohs(tcp_h->th_dport);
    }else if(ip4_h->protocol == IPPROTO_UDP){
        udp_h = (udphdr*) (packet + ETH_HLEN + ip4_h->ihl*4);
        sport = ntohs(udp_h->uh_sport);
        dport = ntohs(udp_h->uh_dport);
    }
```

2.4 Výpis paketů

Paket je při výpisu rozdělen na dvě části:

- hlavička (Ethernetová hlavička + IP hlavička + TCP/UDP hlavička)
- data (datová část TCP/UDP, tzn. data z aplikační vrstvy)



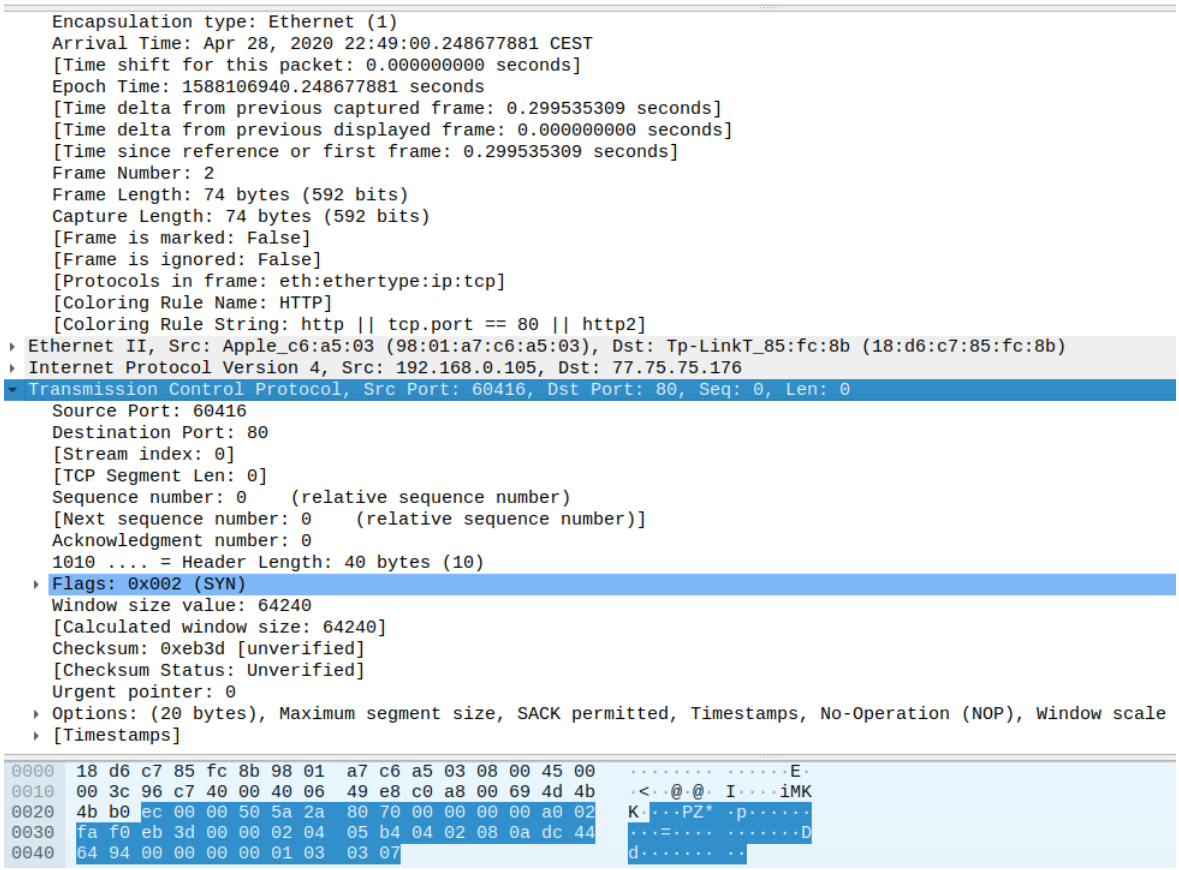
Obrázek 1: Ukázka rozdělení dat při výpisu (převzato z: [2])

2.5 Ukončení programu

Pokud v jakémkoliv místě programu dojde k chybě, vypisuje se chybová hláška (např. „invalid port“, pokud bylo zadáno neplatné číslo portu) a návratový kód programu **1**, jinak se vypisuje požadovaný počet paketů a program končí s návratovým kódem **0**.

3 Testování

Testování spočívalo v porovnávání výstupů programu s výstupem programu Wireshark. Pro testování určitého případu se oba programy nastavily stejnými parametry. Například pro analyzování paketů TCP na portu 80, se program `ipk-sniffer` spustil s nepovinnými parametry `-p 80 --tcp` a programu Wireshark byl nastaven filtr `tcp.port == 80`[1].



Obrázek 2: Výstup programu Wireshark

```
22:49:00.248677 alex-MacBookAir : 60416 > www.seznam.cz : 80
0x0000: 18 d6 c7 85 fc 8b 98 01  a7 c6 a5 03 08 00 45 00  ........ ......E.
0x0010: 00 3c 96 c7 40 00 40 06  49 e8 c0 a8 00 69 4d 4b  .<..@.@. I....iMK
0x0020: 4b b0 ec 00 00 50 5a 2a  80 70 00 00 00 00 a0 02  K....PZ* .p......
0x0030: fa f0 eb 3d 00 00 02 04  05 b4 04 02 08 0a dc 44  ...=.... .......D
0x0040: 64 94 00 00 00 00 01 03  03 07                    d....... ..
```

Obrázek 3: Výstup programu ipk-sniffer

[1] Tyto výstupy jsou dostupné na `https://github.com/alxndrch/ipk-output`

Literatura

[1] Manpage of PCAP. [Online], [rev. 2020-01-29], [cit. 2020-04-20]. Dostupné z: https://www.tcpdump.org/manpages/pcap.3pcap.html

[2] Bouška, P.: Rozumíme počítačovým sítím. [Online], [rev. 2010-05-09], [cit. 2020-04-20]. Dostupné z: https://www.samuraj-cz.com/clanek/rozumime-pocitacovym-sitim

[3] Carstens, T.: Programming with pcap. [Online], [cit. 2020-04-20]. Dostupné z: https://www.tcpdump.org/pcap.html

[4] Postel, J.: User Datagram Protocol [Online]. RFC 768, RFC Editor, Srpen 1980, [cit. 2020-04-20]. Dostupné z: https://tools.ietf.org/html/rfc768

[5] Postel, J.: Internet Protocol [Online]. RFC 791, RFC Editor, Září 1981, [cit. 2020-04-20]. Dostupné z: https://tools.ietf.org/html/rfc791#section-3.1

[6] Postel, J.: Transmission Control Protocol [Online]. RFC 793, RFC Editor, Září 1981, [cit. 2020-04-20]. Dostupné z: https://tools.ietf.org/html/rfc793#section-3.1

[7] S. Deering, R. H.: Internet Protocol, Version 6 (IPv6) Specification [Online]. RFC 2460, RFC Editor, Prosinec 1998, [cit. 2020-04-20]. Dostupné z: https://tools.ietf.org/html/rfc2460#section-3

[8] Wikipedia: EtherType. [Online], [rev. 2020-05-01], [cit. 2020-05-02]. Dostupné z: https://en.wikipedia.org/wiki/EtherType