

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

IPK – Počítačové komunikace a sítě

Implementační dokumentace k 2. projektu
Varianta ZETA - Sniffer paketů

Obsah

1	Úvod	2
2	Nastudované informace	2
2.1	Wireshark	2
2.2	TCP paket a UDP paket	2
2.3	SharpPcap	2
3	Popis implementace	3
3.1	Volba jazyka	3
3.2	Knihovny	3
3.3	Zpracování paketu	3
4	Testování	4
4.1	Snímky testování	4

1 Úvod

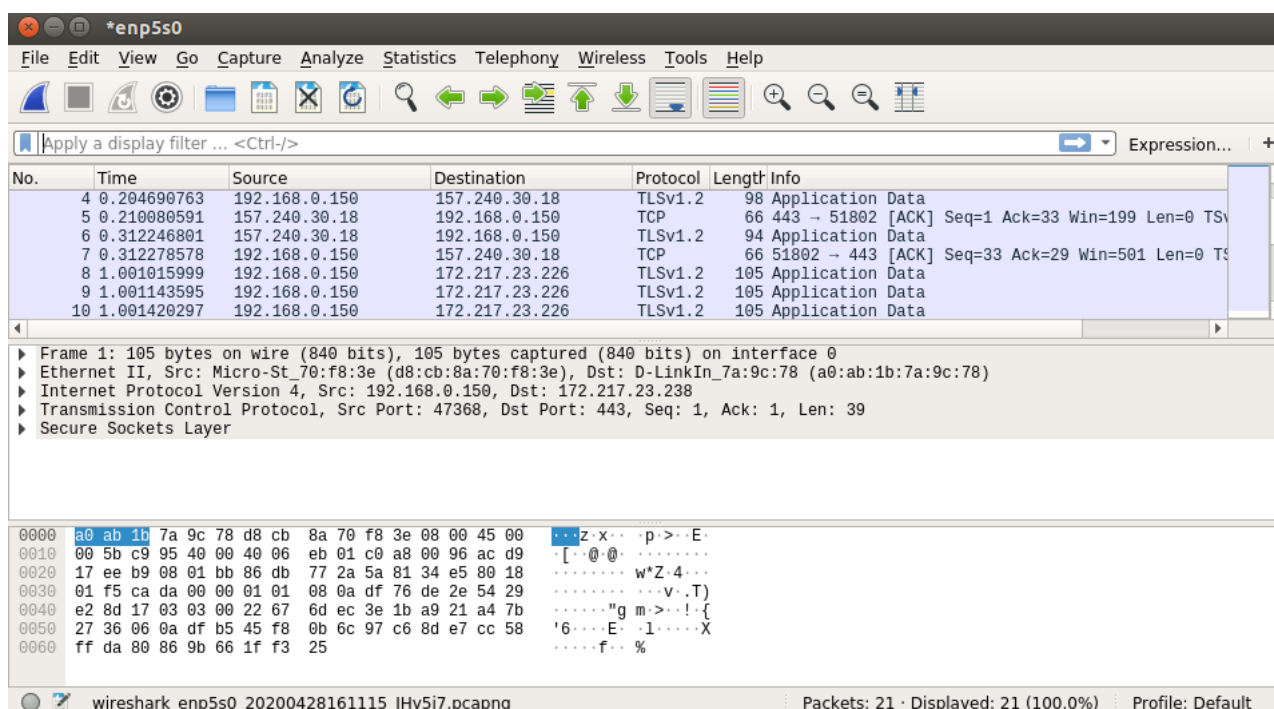
Cílem tohoto projektu bylo navrhnout a implementovat síťový analyzátor, který bude schopný na síťovém rozhraní zachytávat pakety. Tato aplikace je implementována pomocí jazyku c#. Jelikož program pracuje se síťovými rozhraními, je potřeba jej spouštět s administrátorskými právy.

2 Nastudované informace

K implementaci tohoto analyzátoru bylo potřeba předem nastudovat několik důležitých informací, které vedly k úspěšnému výsledku.

2.1 Wireshark

Před návrhem samotného projektu jsem se důkladně seznámil s nástrojem Wireshark, který mi byl po celou dobu programování oporou. Po instalaci tohoto nástroje jsem studoval dokumentaci, která je volně dostupná a můžete ji nalézt třeba zde: [6].



Obrázek 1: Odchycení paketu v aplikaci Wireshark

2.2 TCP paket a UDP paket

Další důležitou součástí projektu bylo nastudovat strukturu jednotlivých paketů. UDP paket je jednoduchý, vypovídá to o něm už samotný popis jeho standardu viz [4]. TCP paket už je o dost složitější a k jeho pochopení je potřeba se nad ním i trochu zamyslet, díky čemuž je jeho standard lehce složitější viz [5].

2.3 SharpPcap

Na závěr literatury uvádím knihovnu, kterou jsem studoval od začátku implementace mého projektu až do jeho konce. Jedná se o knihovnu pro jazyk c#, pomocí které jsem implementoval zachycení paketů viz [2].

3 Popis implementace

Program je implementován v jedinném souboru, jelikož nebylo nutné rozdělovat tuto aplikaci do modulů. Po zpracování argumentů příkazové řádky začne provádění samotného programu, které na daném síťovém rozhraní filtruje a zachytává pakety. Zachytávání paketů probíhá v nekonečné smyčce, kterou je možno ukončit SIGINT signálem.

3.1 Volba jazyka

Jazyk jsem zvolil c# z důvodu toho, že je nejjednodušší na implementaci složitějších struktur a pracování s nimi. Program není psaný objektově orientovaně. Jeho rozsáhlost mi nepřišla pro tuto volbu dostačující.

3.2 Knihovny

K implementaci projektu jsem se rozhodl použít volně dostupnou knihovnu SharpPcap viz [2], která je kompatibilní s knihovnou libpcap. Inspiroval jsem se z příkladu v SharpPcap GIT repozitáři, kde je ukázka jednoduchého programu na odchycení paketu a následné vypsání jeho času a délky viz [3].

```
// otevře zarizeni pro naslouchani
int readTimeoutMilliseconds = 1000;
device.Open(DeviceMode.Promiscuous, readTimeoutMilliseconds);

/* promenna pro paket */
RawCapture packet;

/*promenna pro vypis pozadovaneho poctu paketu */
int paketky = 0;

/*cyklem prochazime paketu po paketu ze zvoleneho zarizeni */
while ((packet = device.GetNextPacket()) != null)
{
```

Obrázek 2: Ukázka kódu použitého z příkladu.

3.3 Zpracování paketu

Velice důležitou součástí projektu je analýza paketu na jeho části. Analýza paketu není prováděna ručně, ale jsou zde použité struktury z knihovny PacketDotNet, které většinu těžké práce udělají za nás. Tato součást je důležitá z důvodu rozhodnutí, zda se jedná o pakety typu UDP nebo TCP, protože jiné pakety tato aplikace nepodporuje. Dále je tento zpracovaný paket použit k zjištění portů. Viz například [1].

```
var tcp = packet_parse.Extract<PacketDotNet.TcpPacket>();
var udp = packet_parse.Extract<PacketDotNet.UdpPacket>();
/*ziskani ipv4 paketu */
var ipv4pak = packet_parse.Extract<PacketDotNet.Ipv4Packet>();
```

Obrázek 3: Ukázka struktury použité z knihovny PacketDotNet

4 Testování

Testování aplikace probíhalo spuštěním mnou implementované aplikace současně s volně dostupným nástrojem (Wireshark)[6]. Nejprve jsem spustil filtrování paketů pomocí programu ipk-sniffer a hned na to filtrování v nástroji Wireshark. Poté jsem hledal první stejný paket pro obě aplikace. Když se mi podařilo takový najít, ručně jsem kontroloval zda jednotlivé výpisy sedí. Nalezení stejného paketu byl většinou oříšek, protože nelze oba programy spustit a vypnout v jeden okamžik. Přikládám snímky obrazovky z testování.

4.1 Snímky testování

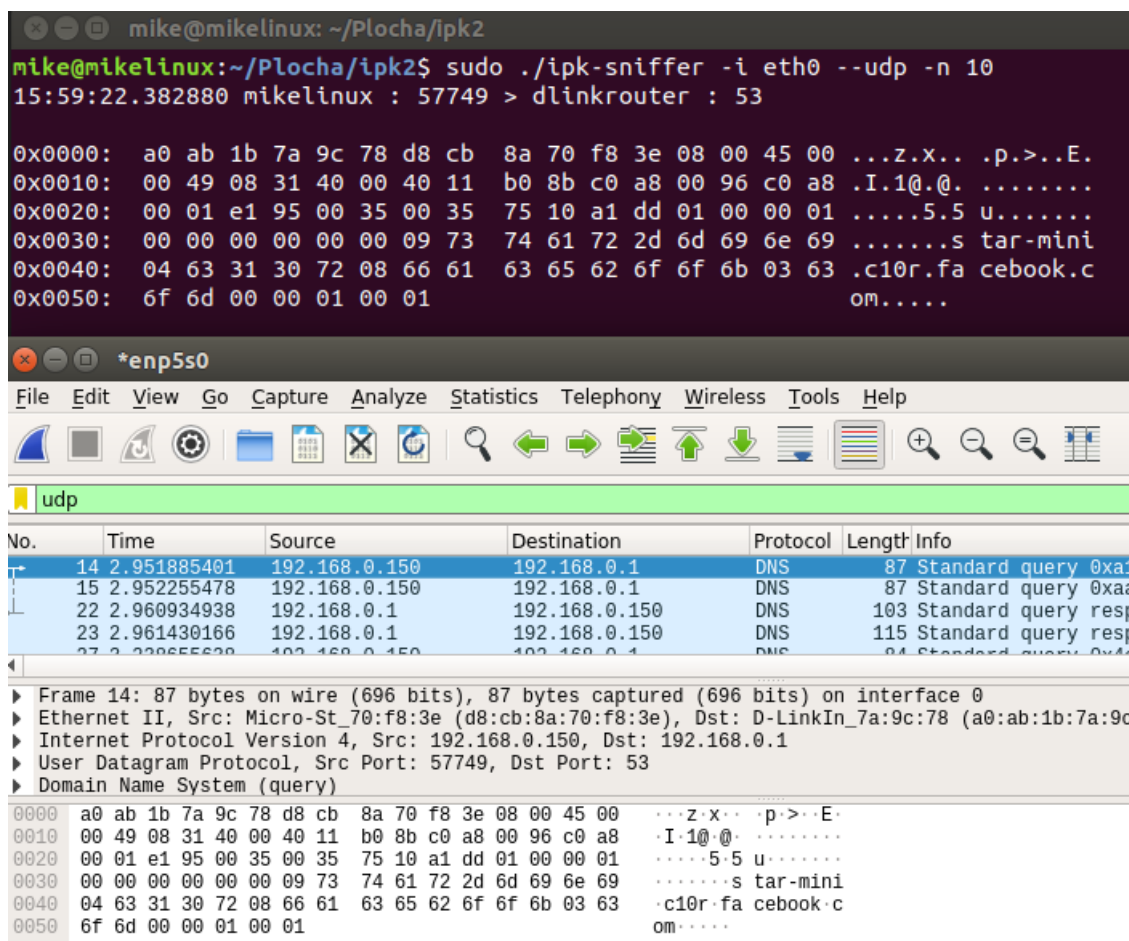
```
mike@mikelinux:~/Plocha/ipk2$ sudo ./ipk-sniffer -i eth0 --tcp -n 10
15:53:33.743083 162.159.130.234 : 443 > mikelinux : 35210

0x0000: d8 cb 8a 70 f8 3e a0 ab 1b 7a 9c 78 08 00 45 00 ...p.>.. .z.x..E.
0x0010: 00 e2 58 cd 40 00 38 06 02 81 a2 9f 82 ea c0 a8 ..X.@.8. ....
0x0020: 00 96 01 bb 89 8a 11 f3 5b ea 08 96 7a cd 50 18 ..... [..z.P.
0x0030: 00 46 01 a5 00 00 17 03 03 00 b5 39 9d 05 ba a5 .F..... ..9....
0x0040: 8f 8c 94 02 2a b5 e1 f2 bb d6 95 ff e0 a9 09 56 ....*... .....V
0x0050: d9 5c c9 4f ce 8f bd 80 4a ef be e7 64 4f b5 b6 .\..0.... J...d0..
0x0060: 84 65 0a 27 4c 19 66 7a 28 01 fc 77 22 9a ec 65 .e.'L.fz (..w"..e
0x0070: 14 6d b8 17 37 62 db bb 44 0c 86 90 db a3 46 55 .m..7b.. D.....FU
0x0080: 49 83 4f e8 4a 7a 08 28 cd b9 b8 2f ae 44 29 b9 I.O.Jz.( .../.D).
0x0090: cc 92 01 8e 9f 34 fd c8 21 89 97 eb 8c 08 08 a4 .....4.. !.....
0x00a0: 86 f6 fb 7c 16 a7 6a 3a 16 0a 12 b4 fa 93 b4 04 ...|..j: .....
0x00b0: fe 70 92 ad a5 8d dd c3 7b df 07 68 ea 0c d8 41 .p..... {..h...A
0x00c0: dc 07 74 47 2e f1 40 61 c3 86 8c 3b dc 09 a2 66 ..tG..@a ...;...f
0x00d0: cd 96 fe 4a 54 38 b2 2b 0d c6 f0 f4 d9 7f bd e1 ...JT8.+ .....
0x00e0: 49 f3 3b be b2 63 85 6e 91 78 0b 82 fc 84 9e 55 I.;..c.n .x.....U

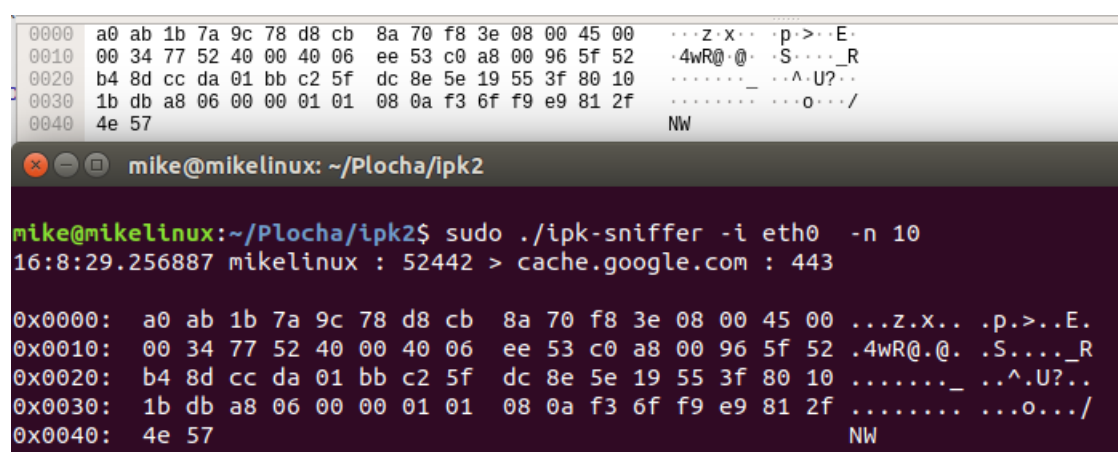
▶ Frame 1: 240 bytes on wire (1920 bits), 240 bytes captured (1920 bits) on interface 0
▶ Ethernet II, Src: D-LinkIn_7a:9c:78 (a0:ab:1b:7a:9c:78), Dst: Micro-St_70:f8:3e (d8:cb:8a:70
▶ Internet Protocol Version 4, Src: 162.159.130.234, Dst: 192.168.0.150
▶ Transmission Control Protocol, Src Port: 443, Dst Port: 35210, Seq: 1, Ack: 1, Len: 186
▶ Secure Sockets Layer

0000 d8 cb 8a 70 f8 3e a0 ab 1b 7a 9c 78 08 00 45 00 ...p.>.. .z.x..E.
0010 00 e2 58 cd 40 00 38 06 02 81 a2 9f 82 ea c0 a8 ..X.@.8. ....
0020 00 96 01 bb 89 8a 11 f3 5b ea 08 96 7a cd 50 18 ..... [..z.P.
0030 00 46 01 a5 00 00 17 03 03 00 b5 39 9d 05 ba a5 .F..... ..9....
0040 8f 8c 94 02 2a b5 e1 f2 bb d6 95 ff e0 a9 09 56 ....*... .....V
0050 d9 5c c9 4f ce 8f bd 80 4a ef be e7 64 4f b5 b6 .\..0.... J...d0..
0060 84 65 0a 27 4c 19 66 7a 28 01 fc 77 22 9a ec 65 .e.'L.fz (..w"..e
0070 14 6d b8 17 37 62 db bb 44 0c 86 90 db a3 46 55 .m..7b.. D.....FU
0080 49 83 4f e8 4a 7a 08 28 cd b9 b8 2f ae 44 29 b9 I.O.Jz.( .../.D).
0090 cc 92 01 8e 9f 34 fd c8 21 89 97 eb 8c 08 08 a4 .....4.. !.....
00a0 86 f6 fb 7c 16 a7 6a 3a 16 0a 12 b4 fa 93 b4 04 ...|..j: .....
00b0 fe 70 92 ad a5 8d dd c3 7b df 07 68 ea 0c d8 41 .p..... {..h...A
00c0 dc 07 74 47 2e f1 40 61 c3 86 8c 3b dc 09 a2 66 ..tG..@a ...;...f
00d0 cd 96 fe 4a 54 38 b2 2b 0d c6 f0 f4 d9 7f bd e1 ...JT8.+ .....
00e0 49 f3 3b be b2 63 85 6e 91 78 0b 82 fc 84 9e 55 I.;..c.n .x.....U
```

Obrázek 4: Testování paketu TCP



Obrázek 5: Testování paketu UDP



Obrázek 6: Testování obou paketů

Reference

- [1] chmorgan: packetnet. [online], rev. 1. duben 2020, [vid. 2020-04-28].
URL <https://github.com/chmorgan/packetnet>
- [2] kayoub5: SharpPcap. [online], rev. 26. duben 2020, [vid. 2020-04-28].
URL <https://github.com/chmorgan/sharppcap/>
- [3] kayoub5: SharpPcap. [online], rev. 29. prosinec 2019, [vid. 2020-04-28].
URL <https://github.com/chmorgan/sharppcap/blob/master/Examples/Example4.BasicCapNoCallback/Example4.BasicCapNoCallback.cs>
- [4] Postel, J.: User Datagram Protocol. [online], rev. 28. srpen 1980, [vid. 2020-04-28].
URL <https://tools.ietf.org/html/rfc768>
- [5] Wikipedia: Transmission Datagram Protocol. [online], rev. 20. duben 2020, [vid. 2020-04-28].
URL https://cs.wikipedia.org/wiki/Transmission_Control_Protocol
- [6] Wireshark: Wireshark Documentation. [online], rev. 20. duben 2020, [vid. 2020-04-28].
URL <https://www.wireshark.org/docs/>