

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ  
Fakulta informačních technologií

POČÍTAČOVÉ KOMUNIKACE A SÍTĚ  
2019/2020

Projekt 2

**Sniffer paketů**

Tomáš Julina (xjulin08)

Zlín, 1. května 2020

# Obsah

1. Úvod .....	3
Obecné zadání .....	3
Zadání varianty ZETA .....	3
Spuštění programu.....	3
Co je packet sniffing?.....	3
Základní návrh a princip programu .....	4
2. Implementace .....	4
Zajímavé knihovny.....	4
Zpracování argumentů.....	4
Hlavní část programu .....	4
Callback funkce.....	5
Rozšíření 1: Odchytávání na více rozhraní, filtrování více portů zároveň.....	5
Rozšíření 2: Ukládání přeložených IP adres do cache .....	6
3. Testování .....	6
Popis testování.....	6
Ukázka testování .....	8
4. Zdroje .....	10

# 1. Úvod

## Obecné zadání

Vytvořte komunikující aplikaci podle konkrétní vybrané specifikace obvykle za použití libpcap a/nebo síťové knihovny BSD sockets (pokud není ve variantě zadání uvedeno jinak). Projekt bude vypracován v jazyce C/C++/C#. Individuální zadání specifikuje vlastní referenční systém, pod kterým musí být projekt přeložitelný a spustitelný. Program by však měl být přenositelný.

## Zadání varianty ZETA

Navrhněte a implementujte síťový analyzátor v C/C++/C#, který bude schopný na určitém síťovém rozhraní zachytávat a filtrovat pakety.

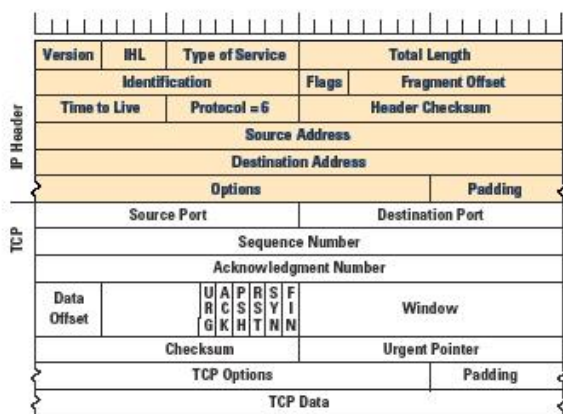
## Spuštění programu

`./ipk-sniffer -i rozhraní[,rozhraní2,...] [-p port[,port2,...]] [--tcp|-t] [--udp|-u] [-n num] [--help|-h]`

- `-i eth0` (rozhraní, na kterém se bude poslouchat. Nebude-li tento parametr uveden, vypíše se seznam aktivních rozhraní), rozšíření: `-i enp0s3,lo` (bude poslouchat na více rozhraních zároveň)
- `-p 23` (bude filtrování paketů na daném rozhraní podle portu; nebude-li tento parametr uveden, uvažují se všechny porty), rozšíření: `-p 23,53` (bude filtrovat na více portech zároveň)
- `-t` nebo `--tcp` (bude zobrazovat pouze tcp pakety)
- `-u` nebo `--udp` (bude zobrazovat pouze udp pakety), pokud nebude `-tcp` ani `-udp` specifikováno, uvažují se TCP a UDP pakety zároveň
- `-n 10` (určuje počet paketů, které se mají zobrazit; pokud není uvedeno, uvažujte zobrazení pouze 1 paket, pokud je uvedena hodnota -1, uvažuje se nekonečný výpis paketů – nutné ukončit program manuálně)
- `-h` nebo `--help` (vytiskne nápovědu a popis, poté ukončí program)

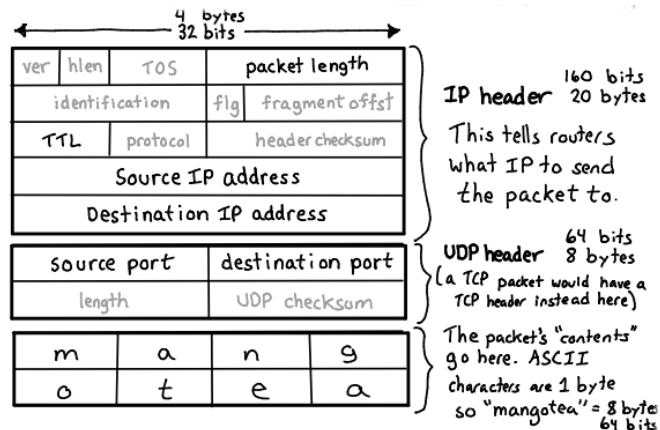
## Co je packet sniffing?

Pokaždé, kdy jsou přes internet přenášena nějaká data, jsou tato data u odesílatele (sender's node) rozdělena na mnoho malých částí = *data packets* a u příjemce (receiver's node) jsou opět poskládány do původního formátu. *Paket* je nejmenší jednotkou komunikace v počítačových sítích a právě odchyťování, následná analýza, filtrace apod. se nazývá *packet sniffing*.



1. TCP/IP packet

<https://www.potaroo.net/ispcol/2004-07/fig1.jpg>



2. UDP packet

<https://jvns.ca/images/packet-headers.png>

## Základní návrh a princip programu

Program se po spuštění pokusí otevřít určité rozhraní k naslouchání, v případě úspěchu je pokaždé, když se podaří odchytnout nějaký paket, volána funkce, která tento paket zpracuje. Paket je struktura, jejíž velikost je možné snadno zjistit, skládá se z více částí a každá část obsahuje určitá data. Na začátku zpracování paketu se zjistí, zda se jedná o TCP či o UDP paket (tato informace je zjistitelná z IP hlavičky) a dále se zjistí, zda odpovídá zadaný port a zda se ještě má další paket vůbec vypsát, v případě, že ano, program paket dále rozdělí a níže popsaným způsobem vypíše na standardní výstup.

## 2. Implementace

Program byl implementován v jazyce C++, zejména kvůli implementaci rozšíření, ve kterých bylo s výhodou využito datového typu *string*, vektorů z knihovny *vector* a hashovací tabulky z knihovny *map.h*. Pro samotné odchyťování paketů byla využita zejména knihovna *pcap*, při následné zpracovávání paketu pro výpis dle zadaných parametrů byly využity struktury z níže zmíněných zajímavých knihoven.

### Zajímavé knihovny

1. *pcap.h* – knihovna byla použita pro odchyťování paketů na vybraném rozhraní [\[4\]](#)
2. *udp.h*, *tcp.h*, *ip.h*, *ip6.h* – knihovny obsahují deklarace hlaviček pro jednotlivé protokoly
3. *getopt.h* – knihovna umožňuje pohodlné zpracování vstupních argumentů
4. *string*, *map*, *vector* – knihovny využité k implementaci rozšíření

### Zpracování argumentů

Kontrolu a zpracování vstupních argumentů má na starosti funkce *parse\_arguments*, argumenty jsou interně zpracovány pomocí funkce *getopt\_long*, která umožňuje „dlouhý“ formát přepínačů v kombinaci s klasickým i s krátkým formátem, jednotlivé možnosti jsou vybírány pomocí konstrukce *switch* a v ní jsou nastavovány pomocné proměnné pro řízení dalšího chodu programu, funkce také kontroluje správný tvar, formát a rozsah argumentů. O zpracování portů a rozhraní se starají pomocné funkce *processInterface* a *processPorts*, které pomocí regulárních výrazů zkontrolují formát vstupního řetězce a poté uloží data do vektoru (v případě rozšíření ještě vstupní formát rozdělí podle doporučeného formátu viz. sekce Rozšíření).

### Hlavní část programu

V případě, že nebyl zadán argument *-i* s názvem aktivního rozhraní, je využito funkce *pcap\_findalldevs*, která nalezne všechny aktivní a zároveň kompatibilní zařízení (s knihovnou *pcap*) a uloží je do listu rozhraní, každá položka má uložený název, případně popis apod. Následně je uživateli vypsán seznam těchto aktivních zařízení a program je úspěšně ukončen.

V případě, že byl zadán argument *-i* s existujícím, aktivním rozhraním, je volána funkce *pcap\_open\_live* (zajímavou implementační volbou je zejména argument s hodnotou 200[ms], který zajistí, že pakety nebudou zpracovávány po jednu, ihned po zachycení, ale v uvedené době se

akumuluje více paketů, které poté budou předány a zpracovány naráz – kdyby tato hodnota chyběla, aplikace by se volala pro každý paket, což by mohlo mít za následek zbytečné vytížení operačního systému), která zadané rozhraní otevře pro odchyťování a umožní získat *packet capture descriptor*. Následně je pomocí funkce *prepareFilter* sestaven filtr, který s využitím funkce *pcap\_compile* a *pcap\_setfilter* pomůže ignorovat nevyhovující pakety. Po aplikování filtrů již zbývá pouze odchyťovat pakety v rámci funkce *pcap\_dispatch*, která poté, co je odchytnut nějaký paket, umožní uživateli daný paket jakkoliv zpracovat pomocí uživatelsky definované *callback* funkce. Program podporuje rozhraní, která vytvářejí ethernetové či linux „cooked“ caption hlavičky (typ hlaviček linkové vrstvy).

## Callback funkce

Tato funkce je volána pokaždé, když se podaří odchyťit nějaký paket. Na začátku funkce se dle protokolu zachyceného paketu rozhodne, zda se jedná o TCP či UDP paket (bylo by možné další dělení, ovšem to se po nás v zadání nežádalo). Hlavička (*header*) paketu se skládá z IP hlavičky (obsahuje například informaci o verzi IP, délce této hlavičky, protokolu – pro nás zajímavý TCP a UDP, zdrojovou a cílovou IP adresu apod.), ethernetové hlavičky (popřípadě linux „cooked“ capture hlavičky) a TCP/UDP hlavičky (obsahují například zdrojový a cílový port). Z části IP hlavičky je získána zdrojová a cílová adresa, tyto adresy se následně program pomocí funkce *getnameinfo* pokusí přeložit a získat pro ně odpovídající doménové jména, tato jména jsou spolu s odpovídající IP adresou uložena do hashovací tabulky, která je využívána jako *cache* proto, aby se opakovaně nepřekládaly stejné adresy neustále dokola pro každý paket (šetří vytížení sítě a nevytváří zbytečné pakety, které by běžely v cyklu dokola, rozšíření základního zadání), z části UDP respektive TCP hlavičky je naopak získán zdrojový a cílový port a v poslední části paketu lze nalézt datový obsah paketu (pokud nějaký paket obsahuje). Takto získaná data jsou již dle zadání vypsaná na standardní výstup pomocí funkce *print\_packet* (princip funkce inspirován ze zdroje [\[11\]](#)), lze toho docílit poměrně jednoduchým způsobem – na začátku každého řádku je vytisknut hexadecimálně počet vypsaných bajtů, poté se vypíše šestnáct bajtů obsahu z datového bufferu hexadecimálně a následně ty samé bajty v klasické ASCII podobě, pokud se jedná o netisknutelné znaky, vypíše se tečka (v případě, že by řádek nebyl celý se zbytek doplní mezerami), takto se vypíše počet paketů, dle vstupních argumentů (když není uvedeno, vypíše se jeden paket). Formát výpisu je následující: na prvním řádku je čas, IP, port zdroje a cíle, následuje prázdný řádek, poté se vypíše zvlášť obsah IP hlavičky, ethernetové hlavičky a TCP (resp. UDP) hlavičky, následuje opět prázdný řádek a v poslední části se vypíše datový obsah paketu (již bez obsahu hlaviček, viz. zadání), takto vypsaný paket je oddělen od následujícího prázdným řádkem.

## Rozšíření 1: Odchyťování na více rozhraní, filtrování více portů zároveň

V projektu jsem se rozhodl implementovat možnost odchyťování paketů na více zařízeních současně a také filtrování paketů z více portů současně.

Ukázka formátu rozšíření:

```
sudo ./ipk-sniffer -i lo,ens3 -p 80,53
```

Jednotlivá rozhraní i porty od sebe musí být odděleny čárkou, zvažoval jsem, zda u portů nepodporovat i možnost „rozpětí od, do“, ovšem vzhledem k povaze paketů jsem se rozhodl, že hledat na více konkrétních portech bude rozumnější.

Aby byla implementace co možná nejčistší s ohledem na již vypracovanou základní verzi, jsem se rozhodl využít vektory a bohatou nabídku práce s řetězci, kterou C++ poskytuje. Při zpracovávání argumentů využívám regulární výrazy (oddělení jednotlivých částí bylo inspirováno [14]), které mi umožní podporovat jak klasickou verzi ze zadání, tak i toto rozšíření, jednotlivé rozhraní a porty jsou ukládány do vektorů a ty jsou dále využívány.

### Více rozhraní

Rozhraní v zadaném formátu se zpracuje funkcí *processInterface* (získání jednotlivých názvů z řetězce inspirováno [14]). Každé zadané rozhraní je otevřeno pro odchytávání paketů, poté se využívá funkce *pcap\_dispatch*, která vždy zpracovává pakety po „dávkách“, tudíž je možné nerušeně v cyklu while neustále přepínat mezi jednotlivými zařízeními a zpracovávat pakety tam, kde právě nějaké jsou, je využíváno také funkce *pcap\_setnonblock*, která umožní tzv. neblokující mód (každá funkce se hned vrátí, tzn. nečeká na příchod paketů a umožní zpracování na dalším rozhraní, díky tomu není nutné využívat více procesů zároveň). Po zpracování a vypsání zadaného množství paketů program skončí.

### Více portů

Zpracování z více portů zároveň je umožněno pomocí filtrování paketů s využitím funkcí *pcap\_compile* a *pcap\_setfilter*, porty jsou po zpracování vstupních dat funkcí *processPorts* (získání jednotlivých portů z řetězce inspirováno [14]) uloženy ve vektoru celých čísel a poté jsou z nich poskládány filtry datového typu *string*, tento filtr potom zařídí to, že pakety s jinými porty než s těmi, které uživatel požadoval, budou ignorovány.

## Rozšíření 2: Ukládání přeložených IP adres do cache

Jako druhé rozšíření jsem implementoval ukládání IP adres do cache, což zamezí zbytečnému neustálému posílání DNS paketů s žádostí o *resolving IP* na doménové jméno (což by způsobilo nekonečný cyklus).

### Implementace

Po získání IP adres z paketu jsou tyto adresy vyhledány v hashovací tabulce obsahující IP adresy s již přeloženými doménovými jmény, v případě, že se najde shoda, vytiskne se již uložená hodnota, jinak se požádá o překlad pomocí funkce *getnameinfo*. Ke *kešování* (caching) je využita struktura *std::map* dostupná v jazyce C++, tato struktura slouží jako hashovací tabulka a umožní ukládat data ve formátu *klíč, hodnota*, klíč i hodnota jsou v mém případě datového typu *string*, v klíči se nachází původní IP adresa a hodnota je doménové jméno (popřípadě IP adresa, pokud se překlad nepodařilo provést).

## 3. Testování

### Popis testování

Pro testování jsem se rozhodl využít velice známý (a hlavně neplacený) program Wireshark a TCPdump. Zajímal mě zejména formát informací, které tento program o paketech zobrazí, formát je velice podobný naší projektové variantě, samozřejmě obsahuje více informací (a možností filtrování), ale to je pochopitelné, výsledky byly naprosto srovnatelné s mou implementací. Další testování jsem prováděl již s hotovou implementací na referenčním virtuálním stroji. Pro vytváření

*network traffic* jsem používal program *curl*, *ssh* a *Mozilla Firefox*. Při testování jsem se pokoušel odchyťovat zejména pakety na portu 80, jelikož se často jedná o protokol http, bylo to pro mě „nejčitelnější“. V průběhu jsem narazil na (pro mě) velice zajímavý případ – a sice, po delším snažení odchyťit UDP paket na portu 80 jsem se dočetl, že se tento port ve vztahu „http-over-UDP“ téměř nikdy nepoužívá (v některých případech tohoto portu využívají viry a ostatní malware), ale byl zde i pokus od společnosti Google o vytvoření protokolu (QUIC = Quick UDP Internet Connections), který by zlepšil výkon (čti rychlost, odezvu) webových aplikací na bázi připojení, které doteď využívají protokol TCP.

# Ukázka testování

student@student-vm: ~/Desktop/xjulin08

```
student@student-vm:~/Desktop/xjulin08$ sudo ./ipk-sniffer -i enp0s3 --tcp -n 5 -p 80
16:00:43.90086 student-vm: 49694 > ec2-18-217-80-105.us-east-2.compute.amazonaws.com : 80
0x0000: 52 54 00 12 35 02 08 00 27 6f 35 b5 08 00 45 00 RT..S... 'o5...E.
0x0010: 00 3c 7f 0a 40 00 40 06 4c 61 0a 00 02 0f 12 d9 <...@...La...
0x0020: 50 69 c2 1e 00 50 95 d5 99 7c 00 00 00 00 a0 02 PL...P... [.....
0x0030: fa f0 6f 7f 00 00 02 04 05 b4 04 02 00 0a 87 80 P...P... [.....
0x0040: 77 2d 00 00 00 00 01 03 03 07 W.....

16:00:44.03950 ec2-18-217-80-105.us-east-2.compute.amazonaws.com : 80 > student-vm : 49694
0x0000: 00 00 27 6f 35 b5 52 54 00 12 35 02 08 00 45 00 ..'o5..RT..S...E.
0x0010: 00 2c 02 0b 00 00 40 06 08 f1 12 d9 50 69 a0 00 .....@.....Pl..
0x0020: 02 0f 00 50 c2 1e 01 1a 3a 01 95 d5 99 7d 06 12 ...P..... [.....
0x0030: ff fb e8 00 00 02 04 05 b4 00 00 .....

16:00:44.04017 student-vm : 49694 > ec2-18-217-80-105.us-east-2.compute.amazonaws.com : 80

student@student-vm: ~
File Edit View Search Terminal Help

student@student-vm:~$ sudo tcpdump -i enp0s3 tcp port 80 -vv -X
tcpdump: listening on enp0s3, link-type EN10MB (Ethernet), capture size 262144 bytes
16:00:43.90086 IP (tos 0x0, ttl 64, id 32522, offset 0, flags [DF], proto TCP (6), length 60)
  student-vm.49694 > ec2-18-217-80-105.us-east-2.compute.amazonaws.com.http: Flags [S], cksum 0x6f7f (incorrect
  t -> 0xed4f), seq 2513803644, win 64240, options [mss 1460,sackOK,TS val 2273343277 ecr 0,nop,wscale 7], length
  0
  0x0000: 4500 003c 7f0a 4000 4006 4c61 0a00 020f E...@.La...
  0x0010: 12d9 5069 c21e 0050 95d5 997c 0000 0000 ..PL..P... [.....
  0x0020: a002 faf0 6f7f 0000 0204 05b4 0402 080a .....@.....Pl..
  0x0030: 0700 772d 0000 0000 0103 0307 W.....
  16:00:44.03950 IP (tos 0x0, ttl 64, id 651, offset 0, flags [none], proto TCP (6), length 44)
    ec2-18-217-80-105.us-east-2.compute.amazonaws.com.http > student-vm.49694: Flags [S.], cksum 0xbfe8 (correct
    ), seq 18496001, ack 2513803645, win 65535, options [mss 1460], length 0
    0x0000: 4500 002c 020b 0000 4006 08f1 12d9 5069 E.....@.....Pl..
    0x0010: 0a00 020f 0050 c21e 011a 3a01 95d5 997d .....P..... [.....
    0x0020: 6012 ffff f8e8 0000 0204 05b4 0000 .....@.....Pl..
  16:00:44.04017 IP (tos 0x0, ttl 64, id 32523, offset 0, flags [DF], proto TCP (6), length 40)
    student-vm.49694 > ec2-18-217-80-105.us-east-2.compute.amazonaws.com.http: Flags [F.], cksum 0xf6fb (incorrect
    t -> 0x18b5), seq 1, ack 1, win 64240, length 0
    0x0000: 4500 0028 7f0b 4000 4006 4c74 0a00 020f E...@.Lt....
    0x0010: 12d9 5069 c21e 0050 95d5 997d 011a 3a02 ..PL..P... [.....
    0x0020: 5010 faf0 6f0b 0000 P...ok...
  16:00:44.04022 IP (tos 0x0, ttl 64, id 32524, offset 0, flags [DF], proto TCP (6), length 113)
```

enp0s3

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

tcp.port == 80

No.	Time	Source	Destination	Protocol	Length	Info
1	0.131920092	10.0.2.15	10.0.2.15	TCP	60	
2	0.271194418	10.0.2.15	10.0.2.15	TCP	60	
3	0.271251481	10.0.2.15	10.0.2.15	TCP	54	
4	0.271516224	10.0.2.15	10.0.2.15	HTTP	127	
5	0.271861241	10.0.2.15	10.0.2.15	TCP	60	
6	0.419768223	10.0.2.15	10.0.2.15	HTTP	706	
7	0.419804464	10.0.2.15	10.0.2.15	TCP	54	
8	0.420551429	10.0.2.15	10.0.2.15	TCP	54	
9	0.421062266	10.0.2.15	10.0.2.15	TCP	60	
10	0.559905487	10.0.2.15	10.0.2.15	TCP	60	
11	0.559928864	10.0.2.15	10.0.2.15	TCP	54	
12	0.170265802	10.0.2.15	35.224.99.156	TCP	74	
13	0.307361028	35.224.99.156	10.0.2.15	TCP	60	

Frame 5: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface  
Ethernet II, Src: PcsCompu, 8f:35:b5 (08:00:27:0f:35:b5), Dst: RealtekU, 12:35:0  
Internet Protocol Version 4, Src: 10.0.2.15, Dst: 10.0.2.15  
Transmission Control Protocol, Src Port: 49694, Dst Port: 80, Seq: 0, Len: 0

student@student-vm: ~

```
student@student-vm:~$ curl http://blank.org
<html>
<head>
<title>
blank
</title>
</head>
<body bgcolor=#ffffff text=#2222ff link=#ff0000 vlink=#880000 al
link=#00ff00>
<br>
</body>
```

3. Ukázka testování na referenčním stroji, http stránka

student@student-vm: ~/Desktop/xjulin08

```
student@student-vm:~/Desktop/xjulin08$ sudo ./ipk-sniffer -i lo -n 2
15:05:08.111746 ip6-localhost : 47742 > ip6-localhost : 80
0x0000: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0x0010: d5 6b 00 28 06 40 00 00 00 00 00 00 00 00 00 00 ..k.(. @.....
0x0020: 00 00 00 00 01 00 00 00 00 00 00 00 00 00 00 00 .....
0x0030: 00 00 00 00 01 ba 7e 00 50 8b c3 35 0b 00 00 .....P..S...
0x0040: 00 0a 02 ff c4 00 30 00 02 04 ff c4 04 02 .....@.....
0x0050: 08 0a 3e 53 93 29 00 00 00 00 01 03 03 07 -->S)...

15:05:08.111755 ip6-localhost : 80 > ip6-localhost : 47742
0x0000: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0x0010: 93 55 00 14 06 40 00 00 00 00 00 00 00 00 00 00 .....
0x0020: 00 00 00 00 01 00 00 00 00 00 00 00 00 00 00 00 .....
0x0030: 00 00 00 00 01 50 50 ba 7e 00 00 00 00 8b c3 .....P.....
0x0040: 35 8c 50 14 00 00 00 01 0c 00 00 00 .....S.P.....

student@student-vm:~/Desktop/xjulin08$

student@student-vm: ~
File Edit View Search Terminal Help

student@student-vm:~$ sudo tcpdump -i lo -vv -X
tcpdump: listening on lo, link-type EN10MB (Ethernet), capture size 262144 bytes
15:05:08.111746 IP6 (flowlabel 0xbdb50b, hlim 64, next-header TCP (6) payload length: 40)
  ip6-localhost.47742 > ip6-localhost.http: Flags [S], cksum 0x0030 (incorrect -> 0x008f),
  seq 2344826251, win 65476, options [mss 65476,sackOK,TS val 1045664553 ecr 0,nop,wscale
  7], length 0
  0x0000: 6000 d56b 0028 0640 0000 0000 0000 0000 ..k.(. @.....
  0x0010: 0000 0000 0000 0001 0000 0000 0000 0000 .....U...@.....
  0x0020: 0000 0000 0000 0001 ba7e 0050 8bc3 358b .....P..S...
  0x0030: 0000 0000 0000 ffca 0030 0000 0204 ffc4 .....@.....
  0x0040: 0402 080a 3e53 9329 0000 0000 0103 0307 -->S)...
  15:05:08.111755 IP6 (flowlabel 0x29355, hlim 64, next-header TCP (6) payload length: 20)
    ip6-localhost.http > ip6-localhost.47742: Flags [R.], cksum 0x001c (incorrect -> 0x3b1
    ), seq 0, ack 2344826252, win 0, length 0
    0x0000: 6002 9355 0014 0640 0000 0000 0000 0000 ..U...@.....
    0x0010: 0000 0000 0000 0001 0000 0000 0000 0000 .....
    0x0020: 0000 0000 0000 0001 0050 ba7e 0000 0000 .....P.....
    0x0030: 8bc3 358c 5014 0000 001c 0000 .....S.P.....
```

Capturing from Loopback: lo

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter ... <Ctrl-F>

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	:::1	:::1	TCP	94	47742 -> 80 [SYN] Seq=0 Win=65476
2	0.000000491	:::1	:::1	TCP	74	80 -> 47742 [RST, ACK] Seq=1 Ack=

Frame 1: 94 bytes on wire (752 bits), 94 bytes captured (752 bits) on interface 0  
Ethernet II, Src: 00:00:00:00:00:00 (00:00:00:00:00:00), Dst: 00:00:00:00:00:00 (00:00:00:00:00:00)  
Internet Protocol Version 6, Src: ::1, Dst: ::1  
Transmission Control Protocol, Src Port: 47742, Dst Port: 80, Seq: 0, Len: 0

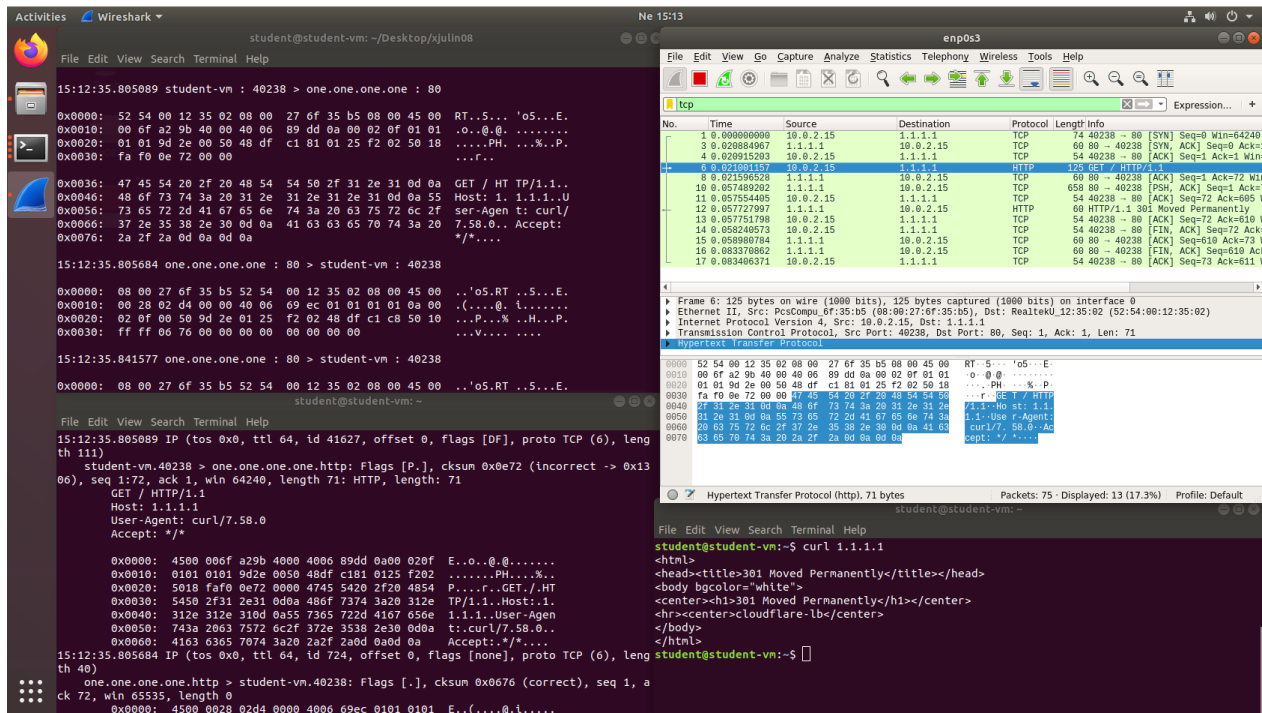
Loopback: lo: <live capture in progress> Packets: 2 - Displayed: 2 (100.0%) Profile: Default

student@student-vm: ~

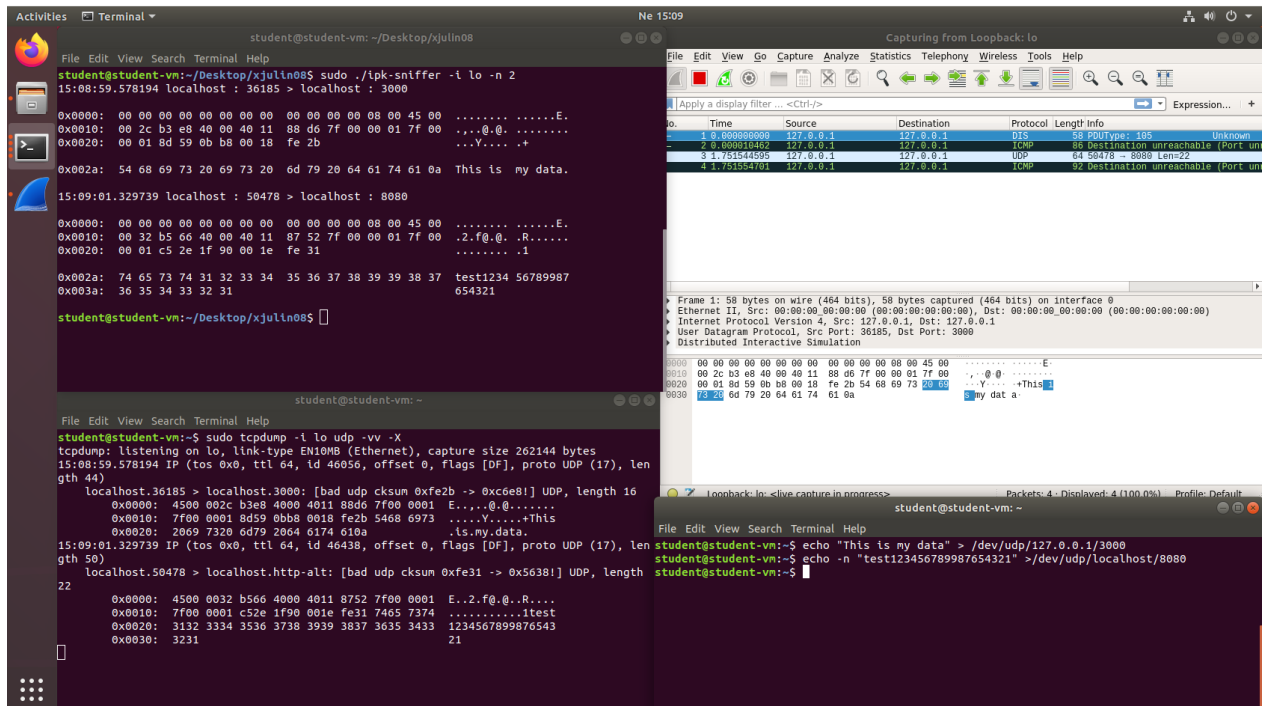
```
student@student-vm:~$ curl -g -6 "http://[::1]:80/"
url: (7) Failed to connect to ::1 port 80: Connection refused
student@student-vm:~$ curl -g -6 "http://[::1]:80/"
url: (7) Failed to connect to ::1 port 80: Connection refused
student@student-vm:~$
```

4. Ukázka testování na referenčním stroji, IPv6 localhost





5. Ukázka testování na referenčním stroji, one.one.one.one



6. Ukázka testování na referenčním stroji, vlastní paket

## 4. Zdroje

- [1] Paessler.com. *What Is Packet Sniffing? Definition And Details*.  
[online] ©2020 Paessler AG [cit. 1.5.2020] Dostupné z: <https://www.paessler.com/it-explained/packet-sniffing>
- [2] Linux.die.net. *Linux Man Pages. PCAP, SOCKET, IP, TCP, UDP man pages*.  
[online] ©1996, die.net [cit. 1.5.2020] Dostupné z: <https://linux.die.net/man>
- [3] Wikipedia, the free encyclopedia. *Pcap*.  
[online] [cit. 1.5.2020] Dostupné z: <https://en.wikipedia.org/wiki/Pcap>
- [4] Tim Carstens. *Programming with pcap*.  
[online] ©2020, The Tcpdump Group [cit. 1.5.2020] Dostupné z: <http://www.tcpdump.org/pcap.html>
- [5] Evans, J. *How Big Can A Packet Get?*.  
[online] ©Julia Evans. [cit. 1.5.2020] Dostupné z: <https://jvns.ca/blog/2017/02/07/mtu/>
- [6] The Internet Society. *RFC 793 – Transmission Control Protocol Specification*.  
[online] ©1981 [cit. 1.5.2020] Dostupné z: <https://tools.ietf.org/html/rfc793>
- [7] The Internet Society. *RFC 791 - Internet Protocol Specification*.  
[online] ©1981 [cit. 1.5.2020] Dostupné z: <https://tools.ietf.org/html/rfc791>
- [8] The Internet Society. *RFC 768 - User Datagram Protocol*.  
[online] ©1980 [cit. 1.5.2020] Dostupné z: <https://tools.ietf.org/html/rfc768>
- [9] The Internet Society. *RFC 3542 - ADVANCED SOCKETS FOR IPv6*.  
[online] ©2003 [cit. 1.5.2020] Dostupné z: <https://www.ietf.org/rfc/rfc3542.txt>
- [10] The Internet Society. *RFC 2460 - Internet Protocol, Version 6 (Ipv6) Specification*.  
[online] ©1998 [cit. 1.5.2020] Dostupné z: <https://tools.ietf.org/html/rfc2460>
- [11] Moon, S., 2013. *C Packet Sniffer Code With Libpcap And Linux Sockets*.  
[online] ©2020 BinaryTides. Dostupné z: <https://www.binarytides.com/packet-sniffer-code-c-libpcap-linux-sockets/>
- [12] Mullins, M. *Exploring The Anatomy Of A Data Packet*.  
[online] ©2001 TechRepublic. [cit. 1.5.2020] Dostupné z: <https://www.techrepublic.com/article/exploring-the-anatomy-of-a-data-packet/>
- [13] Cplusplus.com. *Cplusplus.Com - The C++ Resources Network*.  
[online] ©cplusplus.com, 2000-2020 [cit. 1.5.2020] Dostupné z: <http://www.cplusplus.com/>
- [14] „hayk.mart“. *Parse (Split) A String In C++ Using String Delimiter (Standard C++)*.  
In: stackoverflow.com [online] ©2020 Stack Overflow Dostupné z: [https://stackoverflow.com/questions/14265581/parse-split-a-string-in-c-using-string-delimiter-standard-c#comment44856986\\_14266139](https://stackoverflow.com/questions/14265581/parse-split-a-string-in-c-using-string-delimiter-standard-c#comment44856986_14266139)