VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Počítačové komunikace a sítě – 2. projekt

Sniffer paketů

24. dubna 2022

Vojtěch Dvořák (xdvora3o)

Obsah

1 Úvod

Hlavní motivací pro výběr zadání, které řešiteli ukládá vytvořit sniffer paketů, byla možnost komplexního ověření znalostí o síťových protokolech, jež jsou dnes neodmyslitelnou součástí našich životů. Dalším důvodem bylo také zúročení instalace programu *Wireshark*, který mi poskytl nedocenitelné služby při testování programové části řešení.

Tato dokumentace stručně shrnuje teorii potřebnou pro pochopení fungování vytvořené aplikace. Dále je pak její velká část věnována popisu implementace a testování. Nakonec je zde také přítomen krátký uživatelský manuál. Zatímco je tento text psán v jazyce českém, pro komentáře a soubor *README.md* jsem zvolil angličtinu z důvodu zachování konzistence s názvy funkcí ve zdrojovém kódu.

2 Přehled teorie

Jak je již zmíněno v úvodu dokumentace, úkolem bylo vytvořit sniffer paketů, tedy software pro zachytávání síťové komunikace. Pro implementaci takovéto aplikace je samozřejmě nezbytná znalost způsobu jakým tato komunikace probíhá.

Tato kapitola si dává za cíl stručně shrnout nejnutnější teoretické poznatky důležité pro porozumění řešení projektu. Ačkoli se jedná o základy, mnohdy navíc značně osekané o detaily a speciální případy, předpokládá se alespoň základní znalost informačních technologií.

2.1 Forma komunikace v síti

Každá informace, jež má být skrze síťové rozhraní přenesena, je odesílatelem rozdělena na části. Odesílatelem rozumíme koncové zařízení, které odesílá danou informaci jinému zařízení. Fragmenty informace jsou poté obaleny dodatečnými údaji. Takto získané balíčky, někdy souhrnně označované jako **pakety**, se pak již mohou vydat na strastiplnou cestu skrze síťovou infrastrukturu k jinému zařízení - příjemci, které se postará o rekonstrukci odeslané informace [13, str. 4].

Aplikace by měla být schopna tyto komunikační jednotky byte po bytu vypsat. Kromě toho je však také vyžadováno zobrazit některé informace o zachyceném paketu. V tom nám pomůže znalost internetových protokolů.

2.2 Protokoly a vrstvy

Aby bylo možné mezi zařízeními vyměňovat informace, bylo zapotřebí zavést pravidla komunikace. K tomuto účelu nám slouží tzv. **protokoly**. *„Protokoly definují formát a pořadí zpráv předávaných mezi komunikačními entitami“* [13, str. 9].

Jelikož je přenesení informace mezi dvěma zařízeními poměrně komplexní úkol, je rozdělen na méně komplexní zodpovědnosti, jež jsou rozdistribuovány mezi několik vrstev. V referenčním ISO OSI modelu (viz [12, str. 32]) je rozlišováno vrstev sedm.

Pro účely tohoto projektu však bude postačující se zabývat pouze vrstvami zahrnující rodinu internetových protokolů. Jedná se o aplikační, transportní, síťovou a linkovou vrstvu [7, str. 8]. Každé z těchto vrstev náleží množina konkrétních protokolů, které mohou být pro danou vrstvu použity. Díky vrstevnaté architektuře se sada těchto protokolů někdy označuje jako **zásobník protokolů** [13, str. 50].

V jedné zprávě je tedy typicky obsaženo více protokolů, každý v jiné vrstvě. Každý z nich také přidává dodatečné informace ve formě tzv. **hlaviček**. Pakety označujeme podle vrstvy, na jejíž úrovni o nich hovoříme (a tedy i podle obalení hlavičkami), následujícím způsobem (převzato z [13, str. 51]):

Označení paketu	Vrstva
Zpráva	Aplikační vrstva
Segment	Transportní vrstva
Datagram	Síťová vrstva
Rámec	Linková vrstva

Tato terminologie je používána jak v této dokumentaci, tak i v komentářích zdrojového kódu. Pro řešení projektu je podstatné zejména uspořádání hlaviček za sebou a také jejich formát. Analýzou obsahu hlaviček totiž můžeme získat některé důležité informace.

2.2.1 Linková vrstva

Protože nejnižší vrstva, se kterou síťová rozhraní našich zařízení pracují, je vrstva linková (viz [13, str. 437]), začne analýza komunikace právě zde. Zodpovědnosti této vrstvy jsou (kromě jiného) vytváření rámců, adresování a detekce chyb [1, str. 19]. Jelikož musí program podporovat alespoň ethernetové rámce, musíme znát jejich formát. Ethernetový rámec se skládá po řadě z polí:

Název pole	Délka (B)
MAC adresa příjemce	6
MAC adresa zdroje	6
Typ (specifikuje vyšší vrstvy)	2
Data	46 – 1500
Kontrolní součet rámce	4

Tyto údaje byly převzaty z [1, str. 20]. Na fyzické vrstvě je rámec navíc doplněn o tzv. preambuli [1, str. 75]. Z údajů v tabulce vidíme, že maximální velikost rámce je něco přes 1500 B.

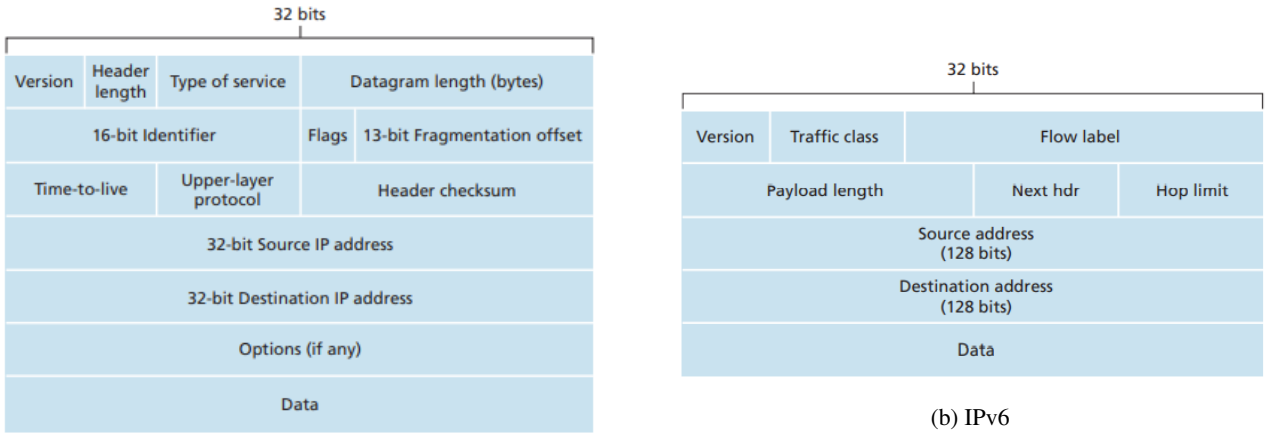
Situaci v tomto směru mohou komplikovat tzv. **jumbo rámce**, jejichž velikost může přesahovat až 9 kB [11]. Při implementaci ale raději budeme počítat s velikostí blízkou maximální velikosti IP datagramu - přibližně 64 kB (neuvažujeme jumbogramy, viz 2.2.2). To nám zajistí bezpečné zachytávání naprosté většiny rámců.

Z polí je z pohledu analýzy paketu nejdůležitější Ethertype, pomocí kterého můžeme zjistit protokol použitý na vyšší vrstvě [5].

S linkovou vrstvou také souvisí jeden z protokolů, jenž má být podporován - **ARP** neboli Address resolution protocol. Jeho úkolem je překlad adres používaných ve vyšších vrstvách na fyzické adresy používané v této vrstvě [4]. Výpis specifických informací obsažený v ARP paketu však není zadáním vyžadován, a proto se jeho formátem nebudeme blíže zabývat.

2.2.2 Síťová vrstva

Z protokolů na síťové vrstvě je vyžadována podpora IPv4 a IPv6 (souhrnně jsou označované jako IP - Internet protocol). Formát datagramů těchto protokolů je následující:



(a) IPv4

(b) IPv6

Obrázek 2: Formát IP datagramů

Tato schemata byla převzata z [13, str. 333, 357]. V záhlaví obou typů datagramů jsou zásadní adresy příjemce a odesílatele, které musí sniffer ve správné formě vypsat.

Z pohledu dalšího zpracování jsou v IPv4 hlavičce důležitá pole **Header length** a **Type of Service** (dále jako ToS). První z nich udává délku IPv4 záhlaví[1], která díky volitelnému poli **Options** může být naneštěstí proměnná. Druhé z nich nám pomůže zjistit transportní protokol a tedy i následující hlavičku. [3].

U IPv6 nás bude z tohoto pohledu zajímat pole označené jako **Next hdr**, jehož sémantika je v podstatě shodná s polem TOS. U tohoto typu datagramu nám však situaci komplikuje existence tzv. **extension headers**. Jedná se o možnost vložení přídavných informací (opět ve formě hlaviček) mezi IPv6 hlavičku a hlavičku vyšších vrstev [8, str. 6].

Jednou z jejich praktických aplikací je tvorba tzv. **jumbogramů**, jež mohou dosahovat výrazně větší velikosti než je teoretický limit daný velikostí pole **Payload length** - 64 kB [6]. Zadání však naštěstí podporu extension headers nevyžaduje.

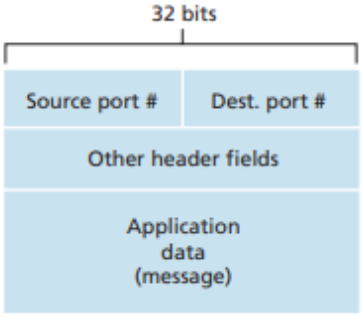
Součástí pole **Data** může být protokol vyšší vrstvy (transportní) nebo ICMP zpráva. **ICMP** (Internet Control Message Protocol) je protokol, jehož prostřednictvím si zařízení v síťové infrastruktuře předávají informace související se síťovou vrstvou (typicky se jedná o hlášení chybových stavů) [2]. *„Přestože využívá IP stejně jako každý jiný protokol vrstvy vyšší, je ve skutečnosti součástí IP a musí být implementován každým IP modulem“*[2, str. 1]. Z tohoto důvodu ho uvádím právě v této sekci.

Tento protokol je nutné dle zadání podporovat včetně jeho IPv6 verze - ICMPv6.

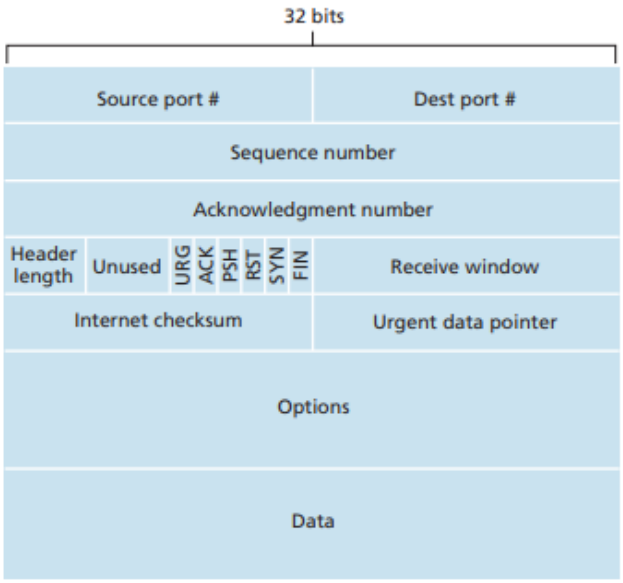
[1] Délka je uvedena v 32-bitových slovech. Pro délku hlavičky v B musíme proto tento údaj vynásobit čtyřmi.

2.2.3 Transportní vrstva

Na této vrstvě je zapotřebí podporovat protokoly UDP a TCP. Na rozdíl od UDP (User datagram protocol), jež slouží pro snadnou komunikaci aplikací bez ustavení spojení, TCP (Transmission control protocol) poskytuje spolehlivý komunikační kanál, jehož zavedení je však relativně komplikované [13, str. 186]. Tato skutečnost se odráží pochopitelně i ve formátech segmentů těchto protokolů:



(a) Formát UDP segmentu



(b) Formát TCP segmentu

Obrázek 4: Formáty TCP a UDP segmentů

Výše uvedená schemata byla opět převzata z knihy [13] ze stránek 202 (UDP) a 234 (TCP). Během analýzy se zaměříme zejména na portová čísla (**Source port** a **Dest port**). Jsou to čísla z rozsahu 0-65535, pomocí kterých jsme schopni identifikovat cílovou nebo zdrojovou aplikaci operující nad transportní vrstvou [13, str. 234].

3 Použité knihovny a technologie

Projekt je implementován v jazyce C++. Tento výběr jazyka umožnil snadnou práci s řetězci a snazší formátování výstupu nežli při výběru klasického C. Zároveň jsem s tímto jazykem měl dostatek zkušeností oproti třetí možné variantě, C#.

Z použitých knihoven bych vyzdvihl hlavně *pcap* [16]. Její funkcionalita je naprosto zásadní pro samotné zachytávání rámců a jejich filtrování. Její použití značně zjednodušilo části projektu, které se těmito úkoly zabývají.

Další knihovny a důvody pro jejich použití jsou uvedeny v kapitole 4.

4 Popis implementace a fungování programu

Při vývoji bylo použito hned několik zdrojů informací. Ve zdrojovém kódu jsou využité materiály uvedeny vždy u funkce, kde se nachází dané know-how. Pokud není uvedeno jinak, jedná se hlavně o myšlenky vedoucí k vyřešení nějakého problému, nikoliv o zdrojový kód jako celek.

Během implementace mi byly nápomocné zejména zdroje [10], [15], manuálové stránky (hlavně pro knihovnu *pcap*, viz [16]) a také v nich přítomné příklady. Základním pramenem byly RFC dokumenty.

4.1 Příprava

Ihned po spuštění programu je zaregistrována obslužná funkce pro signál `SIGINT` prostřednictvím funkce `signal` ze stejnojmenné knihovny. Jedním z požadavků bylo korektní uvolňování prostředků při ukončení klávesami CTRL-C a tímto způsobem můžeme před samotných ukončením zavolat funkci, jež se o ono uvolnění postará. Konkrétně se jedná o funkci `termination_handler`. Ta volá funkci `free_resources`, do níž jsou pomocí speciální kombinace parametrů zaregistrovány prostředky, které mají být uvolněny. Tohoto efektu je docíleno použitím statických proměnných uvnitř této funkce.

4.2 Zpracování přepínačů

Protože přepínačů není málo, lze uvádět v libovolném pořadí a různě je kombinovat (vyjma přepínače `--interface` nebo `-i`), je pro tento účel použito funkcí z knihovny *getopt*. Během analýzy přepínačů a kontroly jejich platnosti je inicializována struktura typu `settings_t`, ze které je možné snadno získat veškerá potřebná nastavení po celý běh programu (na rozdíl od dvourozměrného pole `argv`). Navíc ji můžeme také poměrně snadno propagovat všem funkcím skrze parametr. Pro vytvoření funkcí zpracovávající přepínače mi jako šablona posloužil příklad z [9].

Zvláštním, povinným přepínačem je `-i` (`--interface`) s parametrem specifikující síťové rozhraní, na němž má zachytávání probíhat. Platnost názvu rozhraní je ověřena pomocí knihovny *pcap*. V případě, že byl zadán nevalidní název rozhraní, je pomocí funkce `pcap_findalldevs` získán seznam všech rozhraní a ten je následně vypsán.

4.3 Inicializace struktur

Následuje inicializace struktury `pcap_t` z výše uvedené knihovny *pcap*, která nese všechna potřebná data pro zachytávání rámců na linkové vrstvě. Způsob, jakým se provádí inicializace, byl převzat z článku [10] a patřičně modifikován pro tento projekt. Zajímavá je tvorba filtru, který uživatel specifikuje pomocí přepínačů. Zejména zde jsou zúročeny výhody jazyka C++ pro práci s řetězci.

Vychází se z prázdného řetězce, na jehož konec se během procházení struktury `settings_t` doplňují výrazy popisující filtr. Jedná se např. o řetězec "icmp", který říká, že filtrem mají projít pouze ICMP pakety. Je-li použito více přepínačů, jsou spojeny příslušnými logickými operátory "and" a "or". Využity jsou také závorky pro úpravu jejich precedence.

Kromě výrazů pro filtrování podle protokolů může být přidáno také klíčové slovo "port", pokud je zapotřebí filtrovat pakety podle portových čísel. Takto vytvořený řetězec specifikující filtr je pomocí knihovní funkce `pcap_compile` zkompilován do interní reprezentace stravitelné dalšími funkcemi z knihovny *pcap*. Tento proces probíhá v rámci funkce `create_filter_str`. Formát řetězce charakterizujícího filtr byl nastudován z manuálových stránek a článku [10]. Implicitně není použit žádný filtr, a tudíž sniffer zobrazuje všechny podporované pakety. Pokud je zadáno více přepínačů určujících filtrování, je mezi ně vložena spojka "or". Některé příklady použití těchto přepínačů jsou uvedeny v kapitole 7.

4.4 Zachytávání síťové komunikace

Rámce jsou zachytávány funkcí `pcap_next`, jež je volaná v `sniff_packet`. Tato funkce vrací ukazatel na buffer s obsahem rámce a jako výstupní parametr je navrácen ukazatel na inicializovanou strukturu `pcap_pkthdr`, která obsahuje informace o zachyceném rámci. Další fází je pak analýza jeho obsahu.

Zachytávání rámců je prováděno v cyklu, který je ukončen při dosažení jejich určitého počtu. Ten je získán opět při zpracování argumentů programu (jedná se o uživatelské nastavení) a je předáván strukturou `settings_t` nebo je ponecháno výchozí nastavení na počet jedna.

4.5 Analýza paketu

Zde je využíváno ukazatelů na struktury reprezentující hlavičky protokolů a explicitní přetypování ukazatelů v jazyce C++. Opakovaně (pro každou hlavičku) je kopírována adresa bytu a ta je přetypována na typ ukazatele na strukturu, jež je velikostí shodná s očekávanou hlavičkou. Díky jejímu spojitému uložení v paměti se pak můžeme dívat na pole bytů v paketu sémantikou této struktury. Můžeme se tak jednoduše odkazovat na její členy, které jsou významově ekvivalentní s poli hlaviček, a zjišťovat potřebné údaje.

Původně byl proces analýzy hlaviček implementován jako průchod polem bytů hlavičky v cyklu, což bylo značně neefektivní. Díky protokolům se totiž můžeme spolehnout na to, že hlavička bude mít příslušný formát a nemusíme ji tak dynamicky analyzovat byte po bytu. Navíc je použitím struktur možné poměrně snadno přidat podporu dalších protokolů. K tomuto způsobu analýzy paketu mě inspiroval návod [15].

Ačkoli jsou struktury odpovídající hlavičkám již implementovány v některých knihovnách (např. *netinet*), rozhodl jsem se vytvořit vlastní verze. Použití jejich knihovních ekvivalentů by totiž zbytečně zvyšovalo knihovní závislosti programu. Kromě toho, některé informace obsažené v hlavičkách nejsou zadáním vyžadovány, a proto je postačující jejich zjednodušená verze.

Funkce `get_hdr` zajišťuje jak zkopírování adresy do výstupního parametru, tak i její přetypování na typ ukazatele na příslušnou strukturu. Zároveň je uchováván také index, na němž je očekáván začátek následující hlavičky a také zbývající délka paketu. Na základě těchto hodnot funkce nejdříve ověří, zda je v paketu přítomen požadovaný počet bytů pro strukturu, aby se zamezilo neoprávněným přístupům do paměti.

Při implementaci této funkce byly využity šablony v C++. Získávání ukazatele na strukturu záhlaví je totiž vždy stejný proces. Liší se pouze v použité struktuře. Nevýhodou tohoto přístupu je, že proměnná délka hlavičky (např. při výskytu IPv4 options) musí být ošetřena nad rámec této funkce.

Tímto způsobem nejdříve získáme ethernetovou hlavičku (program podporuje pouze ethernetové rámce), poté hlavičku protokolu použitého na síťové vrstvě a nakonec záhlaví protokolu transportní vrstvy (pokud jsou zde přítomny). Z pohledu analýzy jako takové jsou v hlavičkách zásadní pole, které nám udávají protokol, jenž je použit na vyšší vrstvě (viz 2.2.1).

Informace z hlaviček jsou vypisovány v požadovaném formátu k tomu určenými funkcemi (např. `print_port`, `print_ipv4_addr`...). U portových čísel je vždy důležitá konverze endianity pomocí funkcí z *arpa/inet*, aby byla bez obtíží pro uživatele čitelná. Za zmínku stojí také výpis časového razítka ve formě udané RFC3339 ([14]), kde je zapotřebí uvést časový posun vzhledem k UTC. Tento údaj je zjišťován dynamicky pomocí `gmtime` z knihovny *ctime*. Po výpisu zásadních informací je vypsán obsah celého rámce tak, jak je získán funkcí `pcap_next`.

4.6 Ukončení programu

Zachytávání komunikace může být standardně přerušeno dvěma způsoby, které jsou již lehce nastíněny v předchozích kapitolách - signálem `SIGINT` nebo dosažením počtu zachycených rámců.

První situaci je možné vyvolat v terminálu, kde je program spuštěn, stiskem kláves CTRL-C. Druhý případ nastává automaticky. Je však možné ho oddálit pomocí parametru `--num`.

V obou situacích je ošetřeno uvolnění programem držených zdrojů, tudíž by nemělo docházet např. k úbytkům paměti. Jak v průběhu přípravy, tak i během samotného zachytávání paketů může dojít k chybě (v důsledku použití neplatných parametrů apod.). V takovém případě je vypsáno chybové hlášení na standardní chybový výstup a program je ukončen s kódem 1.

5 Testování

Výsledný program *ipk-sniffer* byl testován na referenčním virtuálním stroji se systémem *Ubuntu 20.04.02*. Překlad programu byl prováděn příkazem[2]:

```
g++ -std=gnu++11 -Wall -Wextra -pedantic ipk-sniffer.cpp -o ipk-sniffer -lpcap
```

Jelikož bylo při testování potřeba se zaměřit na konkrétní části programu, byl využity programy *ping*[3] a *nping*[4]. Druhý z nich však bylo nutné do referenčního stroje doinstalovat. Abych umožnil testování funkcionality spojené s IPv6, musel jsem také poupravit síťové nastavení virtuálního stroje z NAT na NAT network.

Tyto nástroje mi umožnily odesílat z referenčního stroje různé typy paketů (specifikované přepínači při jeho spuštění) a nebylo tak nutné zdlouhavě vyčkávat na paket s protokolem, jehož podpora byla zrovna testována. Příkazy, kterými jsem docílil odeslání konkrétních typů paketů a které tvořily základní sadu testovacích případů jsou následující:

Druh paketu	Příkaz programu *nping*/*ping*
ARP	`sudo nping --arp google.com`
ICMP	`ping google.com`
ICMPv6	`ping -6 google.com`
IPv4 TCP	`sudo nping --tcp google.com`
IPv4 UDP	`nping --udp google.com`
IPv6 TCP	`sudo nping --tcp -6 google.com`
IPv6 UDP	`nping --udp -6 google.com`
IPv4 TCP, dst port 80	`sudo nping --tcp -p 80 google.com`

Kromě těchto případů jsem ale experimentoval i s jinými konfiguracemi. Program *nping* dává uživateli velkou volnost a v nadsázce lze říci, že si tak může poskládat paket "na míru". Pro větší pohodlí jsem z příkazů z tabulky vytvořil cíl *Makefile*, jehož zavoláním (`make -Bi test`) je zajištěna pestrost paketů.

Jako referenční nástroj, se kterým jsem výstupy mnou vytvořeného programu porovnával, jsem vybral open source *Wireshark*[5]. Během testování řešení jsem tuto aplikaci spouštěl z jiného terminálu.

Po provedení testů jsem výstup mého programu porovnával s výstupem z Wiresharku manuálně a na základě rozdílů určil, zda byl test úspěšný nebo ne. Testování, které probíhalo, mělo tři po sobě jdoucí fáze:

1. Průběžné testování během vývoje - po implementaci určité funkcionality byla vždy otestována prostřednictvím nástroje *nping*.

2. Souhrnné testování po dokončení většího celku (např. podpora IPv6, filtrace) - probíhalo pomocí *nping* a výstupy byly porovnávány s programem *Wireshark* (viz 5).

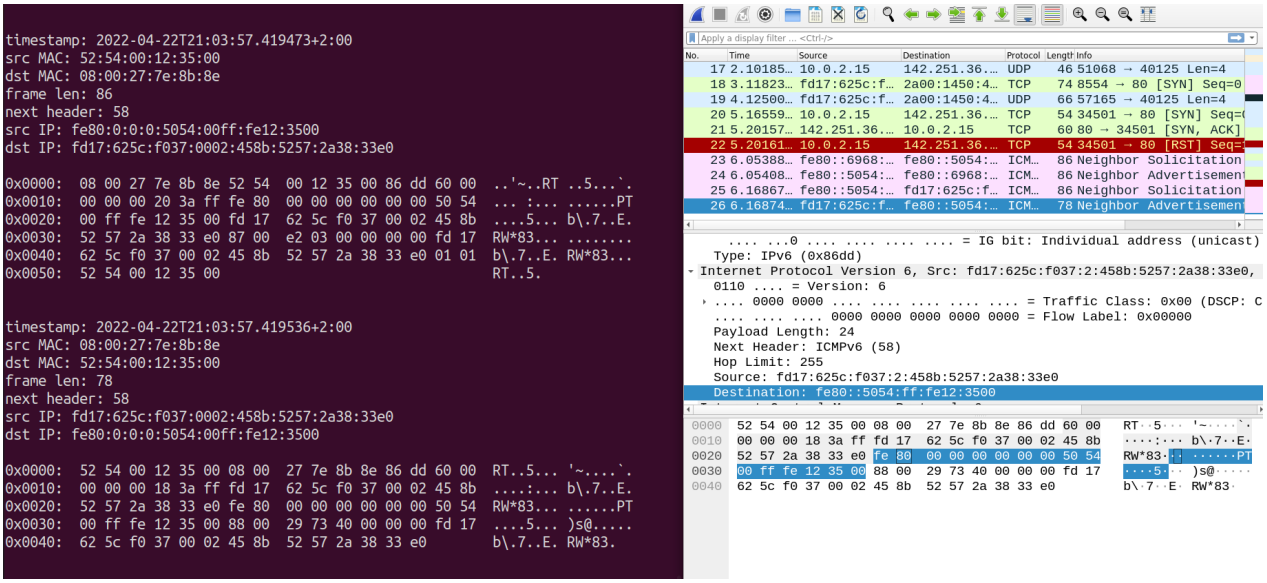
3. Závěrečné testování - programem *ipk-sniffer* byl sledován síťový provoz při běžném používání internetu (např. během prohlížení webových stránek, sledování videa nebo stahování většího souboru) a výstupy byly opět porovnány s referečním programem (viz 6).

[2] Při vývoji a testování byl samozřejmě použit i program *make*, pro nějž byl vytvořen *Makefile*, který je součástí archivu s řešením. Překlad programu je tedy možné provést prostým příkazem `make`

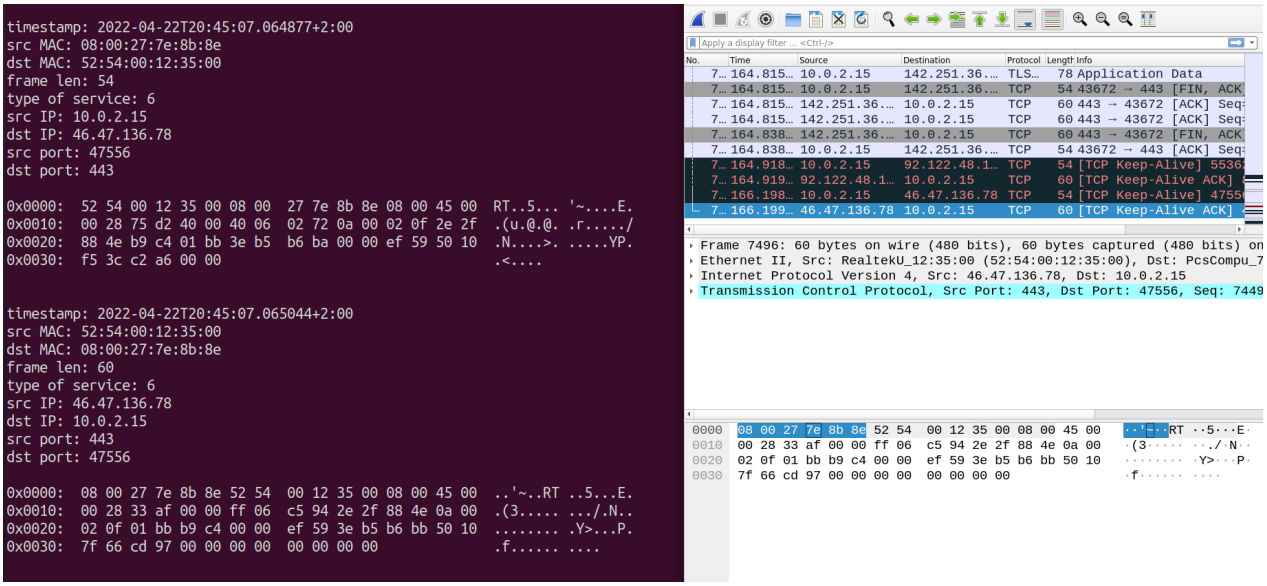
[3] Manuálová stránka programu *ping*: https://linux.die.net/man/8/ping

[4] Manuálová stránka programu *nping*: https://man7.org/linux/man-pages/man1/nping.1.html

[5] Stránky programu *Wireshark*: https://www.wireshark.org/



Obrázek 5: Porovnávání výstupů z programů *ipk-sniffer* a *Wireshark* pro vybrané různorodé testovací případy



Obrázek 6: Porovnávání výstupů z programů *ipk-sniffer* a *Wireshark* při sledování videa na YouTube

6 Výsledek a jeho omezení

Výsledek implementace je po otestovaných stránkách plně funkční a měl by splňovat všechny základní požadavky na funkcionalitu. Vytvořený síťový analyzátor je schopný zachytávat a filtrovat pakety na uvedeném rozhraní a vypisovat všechny požadované informace v zadaném formátu. Podporuje protokoly IPv4, IPv6, ICMP, ICMPv6, UDP, TCP, ARP. Program je ale implementován tak, aby byl tolerantní vůči výskytu také jiných protokolů. Nedojde však k analýze příslušné hlavičky a hlaviček následujících.

Oproti informacím uvedeným v zadání, program vypisuje také IPv4 Type of Service a IPv6 Next Header. Pokud tyto (a jiné volitelné informace) v paketu přítomny nejsou, je jejich výpis zcela přeskočen (není vypsáno tedy ani záhlaví řádku bez hodnoty za dvojtečkou).

Sniffer bohužel podporuje pouze Ethernetové rámce. Dále nepodporuje IPv6 extension headers. Předpokládá se, že výsledný program bude spouštěn s právy uživatele root a také, že síťové rozhraní, nad kterým je zachytávání paketů prováděno, je v promiskuitním módu.

7 Uživatelský manuál

Manuál slouží pro potenciální uživatele programu *ipk-sniffer* a nastiňuje konkrétní způsoby, jakým může být použit. Většina informací vychází ze zadání tohoto projektu.

7.1 O programu

Program *ipk-sniffer* slouží k zachytání a filtraci síťové komunikace. Data obsažená v zachycených rámcích vypisuje na standardní výstup. V případě chyby je ukončen s návratovou hodnotou 1, jinak je navrácena 0. Program podporuje IPv4, IPv6 (bez extension headers), ICMP, ICMPv6, UDP, TCP, ARP a Ethernetové rámce.

7.2 Překlad a spuštění programu

Překlad programu může být proveden příkazem `make` (je-li ve složce se zdrojovým kódem přiložen také *Makefile*) nebo manuálně, tímto příkazem:

```
g++ -std=gnu++11 -Wall -Wextra -pedantic ipk-sniffer -o ipk-sniffer.cpp -lpcap
```

Program ipk-sniffer může být může být spuštěn příkazem ve formátu ({} značí volitelný argument, [] značí povinný argument):

```
sudo ./ipk-sniffer [-i rozhrani | --interface rozhrani] {-p port} {[--tcp|-t]
[--udp|-u] [--arp] [--icmp] } {-n num} {--help|-h}
```

7.3 Přepínače programu

- `interface` (`i`) `rozhrani` parametr tohoto přepínače specifikuje rozhraní, na němž má být komunikace zachytávána
- `n num` parametr tohoto přepínače udává počet paketů, jenž má být zachycen před ukončení programu (implicitně nastaveno na 1)

Přepínače specifikující filtrování paketů (více v 7.4):

- `p port` parametr udává filtrování dle daného portového čísla, vypisovány jsou pouze pakety mířící na daný port/jdou z daného portu
- `tcp` (`t`) vypisovány jsou pouze TCP pakety
- `udp` (`u`) vypisovány jsou pouze UDP pakety
- `arp` vypisovány jsou pouze ARP pakety
- `icmp` vypisovány jsou pouze ICMP a ICMPv6 pakety

7.4 Filtrování paketů

Komunikaci, která má být analyzována, lze blíže specifikovat pomocí přepínačů: `p`, `tcp` (`t`), `udp` (`u`), `arp`, `icmp`. Při použití více přepínačů je mezi nimi uvažován OR operátor (např. při použití kombinace `--udp --tcp` jsou vypisovány jak UDP pakety, tak TCP pakety). V případě, že je zadáno pouze filtrování dle portového čísla a paket portová čísla nevyužívá (tzn. jedná se např. o ICMP), tak zachycen není.

7.5 Interpretace výstupu

Výstup programu může vypadat například takto (bez popisků za < a pod _):

```
timestamp: 2022-04-22T22:07:45.779990+2:00    < čas
src MAC: 08:00:27:7e:8b:8e                    < zdrojová MAC adresa
dst MAC: 52:54:00:12:35:00                    < cílová MAC adresa
frame len: 54                                 < délka rámce
type of service: 6                            < IPv4 typ služby (pokud je)
src IP: 10.0.2.15                             < zdrojová IP adresa (pokud je)
dst IP: 93.184.220.29                         < cílová IP adresa (pokud je)
src port: 58174                               < zdrojový port (pokud je)
dst port: 80                                  < cílový port (pokud je)

0x0000:  52 54 00 12 35 00 08 00  27 7e 8b 8e 08 00 45 00  RT..5... '~....E.
0x0010:  00 28 30 b4 40 00 40 06  c4 37 0a 00 02 0f 5d b8  .(0.@.@. .7....].
0x0020:  dc 1d e3 3e 00 50 54 a4  7b 22 00 07 35 d1 50 10  ...>.PT. {"..5.P.
0x0030:  f9 b0 45 ff 00 00                                 ..E...


^____^   ^_____________________________________________^  ^_________________^
offset       kompletní obsah paketu v HEX formátu           obsah paketu
                                                          (netisknutené znaky
                                                           jsou nahrazeny '.')
```

Z výše uvedeného výstupu tedy můžeme říci např., že se jedná o rámec s délkou 54 obsahující protokoly IPv4 a TCP a který putuje na IP adresu 93.184.220.29 do portu 80. Výpisy jednotlivých paketů jsou odděleny dvěma prázdnými řádky.

7.6 Příklady

1. Zachytávání všech typů paketů na rozhraní *enp0s3* s ukončením po sto paketech:

   ```
   sudo ./ipk-sniffer -i enp0s3 -num 100
   ```

2. Zachycení jednoho paketu na rozhraní *enp0s3* jdoucího z/na port 80 (bude se tedy jednat pouze o UDP nebo TCP paket):

   ```
   sudo ./ipk-sniffer -i enp0s3 -p 80
   ```

3. Zachycení jednoho paketu na rozhraní *enp0s3* jdoucího z/na port 80 (TCP/UDP) nebo ARP paketu či ICMP paketu. Jiné pakety zachytávány nejsou (např. TCP/UDP s jinými portovými čísly):

   ```
   sudo ./ipk-sniffer -i enp0s3 --icmp --arp -p 80
   ```

Reference

[1] The Ethernet: A Local Area Network: Data Link Layer and Physical Layer Specifications. *SIGCOMM Comput. Commun. Rev.*, ročník 11, č. 3, jul 1981: str. 20–66, ISSN 0146-4833, doi:10.1145/1015591.1015594.
URL https://doi.org/10.1145/1015591.1015594

[2] Internet Control Message Protocol. RFC 792, Září 1981, doi:10.17487/RFC0792.
URL https://www.rfc-editor.org/info/rfc792

[3] Internet Protocol. RFC 791, Září 1981, doi:10.17487/RFC0791.
URL https://www.rfc-editor.org/info/rfc791

[4] An Ethernet Address Resolution Protocol: Or Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware. RFC 826, Listopad 1982, doi:10.17487/RFC0826.
URL https://www.rfc-editor.org/info/rfc826

[5] 3rd, D. E. E.; Abley, J.: IANA Considerations and IETF Protocol and Documentation Usage for IEEE 802 Parameters. RFC 7042, Říjen 2013, doi:10.17487/RFC7042.
URL https://www.rfc-editor.org/info/rfc7042

[6] Borman, D. A.; Deering, D. S. E.; Hinden, B.: IPv6 Jumbograms. RFC 2675, Srpen 1999, doi:10.17487/RFC2675.
URL https://www.rfc-editor.org/info/rfc2675

[7] Braden, R. T.: Requirements for Internet Hosts - Communication Layers. RFC 1122, Říjen 1989, doi:10.17487/RFC1122.
URL https://www.rfc-editor.org/info/rfc1122

[8] Deering, D. S. E.; Hinden, B.: Internet Protocol, Version 6 (IPv6) Specification. RFC 8200, Červenec 2017, doi:10.17487/RFC8200.
URL https://www.rfc-editor.org/info/rfc8200

[9] FSF: *The GNU C Library Reference Manual, for version 2.35*. [cit. 2022-04-20].
URL https://www.gnu.org/software/gnu-c-manual/gnu-c-manual.html

[10] Garcia, L. M.: Programming with Libcap - Sniffing the Network From Our Own Application. *Hakin9*, ročník 3, č. 2, 2008: s. 38–46, ISSN 1733-7186.
URL http://recursos.aldabaknocking.com/libpcapHakin9LuisMartinGarcia.pdf

[11] Garcia, N. M.; Freire, M. M.; Monteiro, P. P.: The Ethernet Frame Payload Size and Its Effect on IPv4 and IPv6 Traffic. In *2008 International Conference on Information Networking*, 2008, s. 1–5, doi:10.1109/ICOIN.2008.4472813.

[12] International Telecommunication Union: Data Networks and Open System Communications. [online], Červenec 1994, [cit. 2022-04-19].
URL https://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-X.200-199407-I!!PDF-E&type=items

[13] Kurose, J. F.; Ross, K. W.: *Computer Networking: A Top-Down Approach (6th Edition)*. Pearson, 6 vydání, 2012, ISBN 0132856204.

[14] Newman, C.; Klyne, G.: Date and Time on the Internet: Timestamps. RFC 3339, Červenec 2002, doi:10.17487/RFC3339.
URL https://www.rfc-editor.org/info/rfc3339

[15] Sarma, G.: Packet sniffer and parser in C. [online], Srpen 2019, [cit. 2022-04-20].
URL https://gauravsarma1992.medium.com/packet-sniffer-and-parser-in-c-c86070081c38

[16] The Tcpdump Group: Man page of PCAP. [online], Září 2020.
URL https://www.tcpdump.org/manpages/pcap.3pcap.html