Sem vložte zadání Vaší práce.

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra číslicového návrhu

Bakalářská práce

Programovatelný softwarový generátor ethernetových rámců

Miroslav Marek

Vedoucí práce: Ing. Pavel Kubalík Ph.D.

16. května 2013

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V Praze dne 16. května 2013 .....

Odkaz na tuto práci

Marek, Miroslav. *Programovatelný softwarový generátor ethernetových rámců*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2013.

Abstract

This bachelor thesis describes the design and implementation of network traffic generator for Ethernet frame level. Generator lets you configure the data sent by the configuration file. To send Ethernet frames is used pcap library services. Receiver is also included. It is used to verify the accurancy of a data, which was sended by generator.

Keywords frame, packet, generator, receiver, pcap

Abstrakt

Tato práce se zabývá návrhem a implementací generátoru síťového provozu na úrovni ethernetových rámců. Generátor umožňuje konfigurovat odesílaná data formou konfiguračního souboru. Pro odesílání ethernetových rámců je využito služeb knihovny pcap. Součástí práce je i přijímač sloužící k ověření správnosti obsahu generátorem odesílaných dat.

Klíčová slova rámec, paket, generátor, přijímač, pcap

Obsah

Seznam obrázků

Úvod

Znalost procesů odehrávajících se v prostředí počítačových sítí je považována za jednu ze základních kvalifikací profesionála v oblasti informačních technologií, zejména pak u studenta oboru Informační technologie, kterému by měly sítě býti denním chlebem.

K výběru tématu závěrečné práce mě vedla především výše zmíněná skutečnost. Vzhledem k dnešní všudypřítomnosti počítačových sítí nepředpokládám, že bych snad přišel na něco nového či implementoval něčím vyjímečný program. Generátorů síťového provozu dnes existuje nespočet a analyzátorů ještě více. Jimi nabízená funkcionalita je na špičkové úrovni. Ambice překonat již hotová řešení na poli generátorů/analyzátorů síťového provozu nemám.

Hlavní přínos mé práce spatřuji v osvojení si technologií používaných v prostředí počítačových sítí, neb nejlepší cestou, jak detailně porozumět problému, je naimplementovat jeho řešení. Touto cestou bych se chtěl seznámit s protokoly Ethernet, IP, ICMP, UDP a TCP. Tak se zdá, že největší profit z práce budu mít já, nicméně ani vy s prázdnou neodejdete.

Můžete být svědky celého průběhu tvorby programu. Začít nelze jinak, než specifikací cíle. Následovat bude zjištění, že konkurence je kapánek napřed. Pro zdárné snížení jejího náskoku bude třeba analýzy, kde si popíšeme základní vlastnosti protokolů, které budou implementovány. Následovat budou části věnované návrhu a implementaci. Na závěr nesmí být zapomenuto na kapitolu věnovanou testování.

Kdo má zájem a přiložené CD, může osobně funkčnost programu vyzkoušet v praxi. Uživatelskou příručku hledejte mezi přílohami.

Kapitola 1

Analýza zadání

Před analýzou samotného zadání bakalářské práce je potřeba si nejprve ujasnit několik základních pojmů, které mají tendenci splývat v pojem jeden. Je nutné mezi těmito pojmy cítit jasný rozdíl.

1.0.1 Rámec

Rámec je úsek dat, který jako jediný skutečně putuje sítí. Rámce jsou záležitostí vstvy síťového rozhraní TCP/IP, na jejíž úrovni vznikají i zanikají. Rámec tvoří hlavička, data a zakončení. Podrobněji si formát rámce probereme později. Do dat je možné ukládat cokoliv, ale především je to prostor, jenž je vyplňován shlukem bajtů zvaným „paket“. Nyní je pro nás podstatné zejména to, že délka datové části rámce je omezená, díky čemuž rámec nemusí pojmout celý paket. Pokud k dané situaci dojde a paket je větší nežli datová část rámce, dochází k tzv. fragmentaci paketu na více menších částí.

1.0.2 Paket

Paket je úsek dat, jenž se vkládá do datové části rámce. Paket obsahuje hlavičku a tělo. Patička se na rozdíl od rámce u paketu nevyskytuje. V následujícím textu bude pojmem „paket“ vždy myšlen paket protokolu IP verze 4. Označením „TCP/UDP/ICMP paket“ je myšlen také paket typu IP, jen je navíc specifikován jeho obsah.

1.0.3 Fragment paketu

Fragment paketu je paket, jež vzešel z jiného paketu při procesu zvaném „fragmentace“. Jeho velikost je přizpůsobena maximální velikosti datové

části rámce. Nese částečnou informaci o původním paketu.

1.0.4 Datagram

Datagram je označení vžité pro paket, jehož přijetí příjemce nepotvrzuje. Jedná se o paket, který ve své datové části obsahuje segment protokolu UDP či jakéhokoliv jiného protokolu, jenž neposkytuje záruku doručení.

1.0.5 Segment

Úsek dat sestávající z hlavičky protokolu transportní vrstvy (např. TCP, UDP) či protokolu síťové vrstvy (např. ICMP) a datové části. Celý se vkládá do datové části paketu.

1.1 Rozbor zadání

Cílem této práce je navrhnout a implementovat aplikaci, jenž umožní odesílat skrze síťové rozhraní pakety/rámce předem zadaných parametrů. Výsledek vzešlý z této práce by měl sloužit jako nástroj pro testování schopnosti k síti připojených zařízení vypořádat se s různým nepředvídatelným síťovým provozem k nim směřujícím. Při psaní programu je doporučeno využít služeb knihovny pcap.

1.1.1 Funkční požadavky

- konfigurace požadovaných rámců/paketů ve formě textového konfiguračního souboru
- možnost generovat prosté rámce
- možnost generovat prosté pakety
- možnost generovat pakety obsahující UDP segment
- možnost generovat pakety obsahující TCP segment
- možnost generovat pakety obsahující ICMP segment
- možnost nechat si přehledně vypisovat informace o příchozích paketech/rámcích

1.1.2 Nefunkční požadavky

- funkčnost programu v OS Windows
- funkčnost programu v OS Linux
- implementace v jazyce C++
- využití knihovních funkcí knihovny pcap

1.2 Použité nástroje

Při tvorbě této bakalářské práce bylo použito následující softwarové vybavení.

1.2.1 Prostředí OS Linux

`Geany` textový editor

`G++` kompilátor zdrojových kódů

`Make` utilitka usnadňující překlad programu

`Doxygen` nástroj pro generování dokumentace přímo ze zdrojového kódu

`Debian Wheezy` zástupce z řad operačních systémů OS Linux

`Libpcap` knihovna pcap ve své verzi pro OS Linux

`Graphviz` generátor grafů ze zadaných hodnot

`Wireshark` program pro odposlouchávání síťového provozu (použitý při testování)

1.2.2 Prostředí OS Windows

`Visual Studio Express` vývojové prostředí

`Windows 7` zástupce z řad operačních systémů OS Windows

`Winpcap` knihovna pcap ve své verzi pro OS Windows

Kapitola 2

Již existující řešení

Aplikací vytvářejících datový provoz na přání uživatele je velké množství. Obecně se dá říci, že naprostá většina z nich a generuje síťový provoz za použití protokolů Ethernet, IPv4, IPv6, TCP, UDP, ICMP a ICMPv6. Na samotném obsahu paketů zpravidla nezáleží. Hojně se využívá knihovny `libpcap` a její obdoby pro operační systém Windows – `winpcap`.

2.1 Packeth

Domovská stránka: `http://packeth.sourceforge.net/sourceforge/Home.html`

Program Packeth je multiplatformní – funguje pod operačními systémy Windows, Mac a Linux. Lze ho ovláfat jak skrze grafické uživatelské rozhraní, tak i přes příkazovou řádku. Umožňuje nastavit celkový počet odesílaných paketů. Umožňuje nastavit zpoždění mezi odesílanými pakety. Dovoluje změnit parametry odesílaných paketů v průběhu odesílání. Ukládá si konfiguraci do souboru pro pozdější využití (podporuje formát *.pcap).

2.2 Mausezahn

Domovská stránka: `http://www.perihel.at/sec/mz/`

Generátor paketů optimalizovaný pro rychlost psaný v jazyce C. Používá se mimojiné k měření kolísání zpoždění paketů (jitteru) posílaných mezi dvěma počítači nebo pro simulaci DoS útoků. Ovládá se pouze z příkazového řádku. Je k dispozici pouze pro operační systém Linux. Autor tohoto programu si nepřeje, aby vznikl port tohoto programu na Windows (z ideologických důvodů). Multiplatformní tedy tento progarm není.

2.3 Pacgen

Domovská stránka: `http://pacgen.sourceforge.net/`

Utilita pro generování ethernetových rámců. Umožňuje generovat IPv4 pakety obsahující UDP či TCP segment. Funguje pouze na operačním systému Linux. Umožňuje definovat časový interval mezi odesláním jednotlivých paketů.

2.4 Scapy

Domovská stránka: `http://www.secdev.org/projects/scapy/`

Nástroj pro interaktivní zpracování packetů. Umožňuje pakety podvrhovat, dekódovat či zachycovat. Z naměřených výsledků umí vytvářet grafické výstupy (grafy sítí, strukury packetů).

2.5 Iperf

Domovská stránka: `http://dast.nlanr.net/Projects/Iperf/`

Diagnostický nástroj sloužící k měření propustnosti sítě založené na IP protokolu. Funguje jak pod OS Windows, tak i pod OS Linux.

Kapitola 3

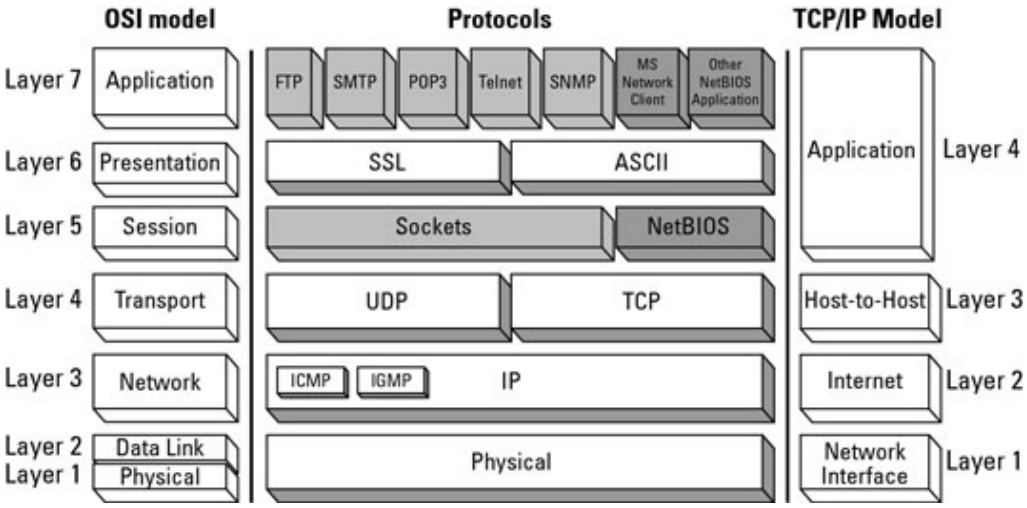
Analýza

Celá bakalářská práce se točí kolem implementace internetových protokolů, které by měl program podporovat. Nyní bychom si měli alespoň zhruba povědět, jak protokoly, jež budeme implementovat, vlastně fungují. K protokolům vede cesta přes pojem zvaný „síťová architektura“.

3.1 Síťová architektura

Síťová architektura popisuje fungování počítačové sítě. Počítačové sítě jsou natolik komplexní problém, že se jejich fungování definuje v několika vzájemně oddělených „vrstvách“. Děje se tak proto, aby se celý systém zjednodušil a zmodularizoval. Aby se probém komunikace v síti dekomponoval do problémů menších, na sobě nezávislých. Díky této „vrstevnatosti“ lze nahradit/upravit jakoukoliv z vrstev, aniž by to vyžadovalo zásah do vrstev ostatních. Úkolem každé vrstvy je poskytovat služby následující vyšší vrstvě a nezatěžovat ji zbytečně detaily o tom jak je služba ve skutečnosti realizována. Vrstva interaguje vždy s nejbližšími sousedními vrstvami – poskytuje služby vrstvám vyšším a využívá služeb vrstev nižších.

Jednotlivé vrstvy jsou realizovány protokoly, tudíž vrstvám také odpovídají hlavičky jednotlivých protokolů ve struktuře paketu. Základem paketu je blok aplikačních dat, který může mít klidně i nulovou délku. Před tento blok postupně jednotlivé protokoly přidávají (v pořadí vrstev shora dolů) hlavičky se svými informacemi. Tyto hlavičky jsou pak moduly příslušných protokolů v opačném pořadí odebírány na straně příjemce. Dochází tak k opakovanému zapouzdřování dat. Nižší vrstva vždy hlavičku s daty vyšší vrstvy pokládá jednoduše za data, v nichž nerozlišuje hlavičku vyšší vrstvy a ani nic jiného. Nejznámějšími definicemi síťové architektury popisujícími



Obrázek 3.1: Srovnání síťových architektur

tyto vrstvy jsou ISO/OSI a TCP/IP. Srovnání těchto definic můžete vidět na obrázku 3.1 převzatého z [2]

3.2 Protokol

Protokol je psanou definicí pravidel komunikace dvou stran. Definuje minimálně pravidla pro syntaxi, sémantiku a synchronizaci vzájemné komunikace. Usnadňuje vzájemnou spolupráci různých platforem. Protokoly týkající se počítačových sítí jsou publikovány formou RFC dokumentů 6.3. Protokoly se rozdělují do několika úrovní (vrstev). Každý protokol se vztahuje vždy k určité vrstvě modelu, ve které operuje a jejíž podstatu svou činností vlastně naplňuje.

3.3 Model ISO/OSI

Model ISO/OSI je abstraktním popisem síťové architektury. Dekomponuje síťovou architekturu do 7 vrstev. Nebyl nikdy realizován. Z modelu ISO/OSI vychází TCP/IP. Hodí se zejména pro teoretický popis sítí.

3.3.1 Fyzická vrstva

Fyzická vrstva, označovaná někdy také pořadovým číslem 1 je nijnižší vrstvou celého ISO/OSI modelu. Definuje fyzické, elektrické, mechanické a

funkční parametry týkající se fyzického propojení jednotlivých zařízení. Definuje 0 a 1, respektive způsob modulace. Umožňuje přenos bitů.

3.3.2 Linková (spojová) vrstva

Data z fyzické vrstvy uspořádává do celků zvaných „rámce“. Detekuje chyby vzniklé při přenosu dat. Funguje na základě fyzické adresace, kdy každé zařízení v síti má jedinečnou adresu zvanou MAC. Poskytuje spojení dvěma zařízením v rámci lokální sítě.

3.3.3 Síťová vrstva

Zavádí logické adresy, díky nimž je umožněna komunikace stanic z různých sítí. Ke svému provozu vyžaduje existenci zařízení zvaných směrovače. Nejznámějším protokolem pracujícím na této vrstvě je bezpochyby IP protokol.

3.3.4 Transportní vrstva

Zajišťuje přenos dat mezi koncovými zařízeními. Je raalizována spojově (TCP) a nespojově (UDP) orientovanými protokoly. Je zodpovědná za spolehlivé odeslání dat.

3.3.5 Relační vrstva

Má za úkol udržovat spojení/relaci mezi dvěma zařízeními tak dlouho, dokud je potřeba.

3.3.6 Prezentační vrstva

Specifikuje způsob, jakým jsou data formátována, prezentována, transformována a kódována. Řeší rozdíly v reprezentaci dat mezi aplikací a síťovým formátem.

3.3.7 Aplikační vrstva

Nejvyšší vrstva. Definuje způsob, jakým komunikují se sítí aplikace. Používá služby nižších vrstev a díky tomu je izolována od mnoha detailů jejich realizace. Zahrnuje protokoly jako jsou například FTP, DNS, POP3, SMTP, SSH atd.

3.4 Model TCP/IP

Rodina protokolů TCP/IP jejíž vznik se datuje na přelom 70. a 80. let se dodnes používá k realizaci většiny síťové komunikace. Navržena byla s cílem vytvořit robustnější model síťové komunikace, pro něhož by nepředstavovaly výpadky částí sítě vážný problém. Základní myšlenkou celé rodiny TCP/IP je technika přepínání paketů, což znamená to, že data nejsou posílána formou souvislých proudů, nýbrž formou samostatných na sobě nezávislých blocích (paketech). Jednotlivé uzly sítě se samy rozhodují, kudy pakety potečou. Rodina TCP/IP sestává ze 4 vrstev.

3.4.1 Aplikační vrstva

Nejvyšší a na protokoly nejbohatší vrstva. Zahrnuje protokoly jako jsou například HTTP, SMTP, FTP, NTP. Hlavička protokolu a jí přenášená data dohromady představují pro nižší vrstvy jednolitá data, která je potřeba přenést. O přenos se starají nižší vrstvy.

3.4.2 Transportní vrstva

Transportní vrstva zajišťuje přenos dat aplikační vrstvy. Je realizována spojově a nespojově orientovanými protokoly TCP respektive UDP. Tyto dva protokoly v současné době transportní vrstvě výrazně dominují.

3.4.3 Síťová vrstva

Stará se o adresování a směrování. V praxi je realizovaná téměř výhradně protokolem IP. Zahrnuje protokoly IP a ICMP. Implementována je jak v koncových stanicích, tak i ve směrovačích. Adresy protokolu IP jsou přidělovány hierarchicky tak, že delegace jednotlivých rozsahů odpovídá topologii sítě. Z cílové IP lze určit, kudy máme paket poslat. Kromě této své zásadní funkce řeší IP protokol ještě některé další problémy, například fragmentaci (rozdělení příliš dlouhých paketů na pakety kratší). Některé problémy ale IP neřeší. Adresovat dovolí konkrétní uzel sítě, nikoliv však konkrétní proces běžící na stanici. Dalším problémem je, že dva pakety běží sítí zcela samostatně a nelze poznat, zda-li dva pakety patří do téhož datového toku. Také neřeší otázku ztracených paketů či změnu jejich pořadí během jejich cesty k cíli. U některých datových toků to nemusí být zcela na závadu, tam kde ano to řeší vrstva transportní.

3.4.4 Vrstva síťového rozhraní

Slouží k přenášení informací po přenosovém médiu. Tato vrstva používá k adresaci adresy zvané MAC. Tyto adresy nelze používat ke směrování, neb z nenesou informaci o tom, kde cíl hledat.

3.5 Protokol IP

V modelu TCP/IP protokol IP zastává funkci síťové vrstvy. Stávající verze číslo 4 byla vytvořena na University of Southern California v roce 1980. Jeho přesnou definici lze nalézt v dokumentu RFC791 [4]. Tento obecně nejznámější protokol používaný v počítačových sítích zajišťuje hned několik funkcí. Umožňuje přenos dat mezi dvěma zařízeními v celosvětové síti Internet, což jinými slovy znamená, že umožňuje komunikaci zařízením, která neleží ve stejné síti. Stará se o adresování a směrování, na což má dnes již prakticky monopol. Jedinou vážnější konkurenci má IP protokol sám v sobě – svou vyšší verzi (v současnosti verzi 6).

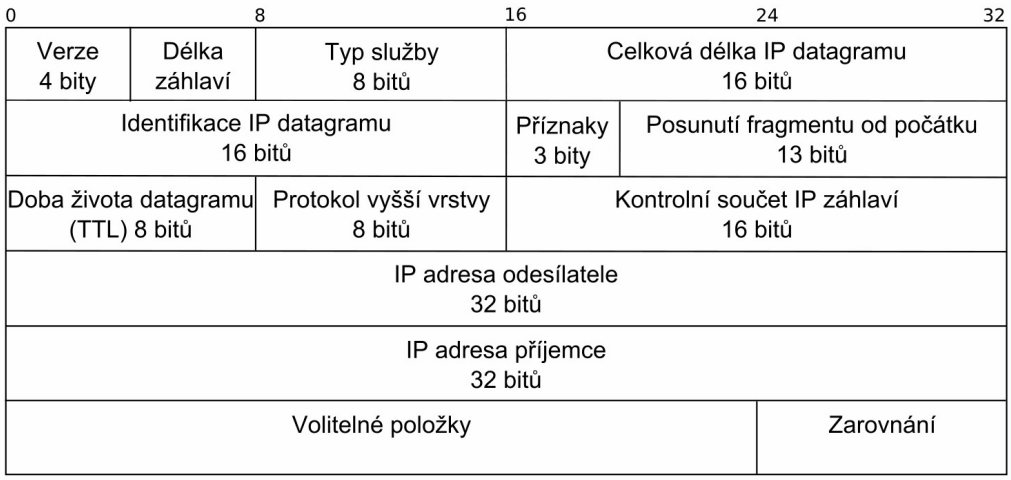
Směrování je realizováno na zařízeních zvaných směrovače, jenž leží na hranicích sítí a směrují provoz z jedné sítě do druhé. Obě komunikující strany musí být jednoznačně určeny adresou, kterou v rámci protokolu IP tvoří 32 bitů dlouhé číslo. Každá veřejná IP adresa je unikátní v rámci celého internetu. Směrovače si data vzájemně předávají do doby, než paket dorazí do sítě příjemce. Obsah paketu není v těchto zařízeních měněn s výjimkou položek hlavičky týkajících se fragmentace paketu a době životnosti paketu. IP paket sestává z hlavičky a těla. Obsah hlavičky je znázorněn na obrázku 3.2.

3.5.1 Položky hlavičky

V následujících kapitolách bude `takto zvýrazněným textem` vždy myšlena položka obsažená v hlavičce IP paketu.

3.5.1.1 Verze protokolu

Číslo verze použitého IP protokolu. Od hodnoty této položky se odvíjí složení položek následujících. V celé této práci se zabýváme pouze protokolem IP verze 4.



Obrázek 3.2: Struktura hlavičky IP paketu

3.5.1.2 Délka záhlaví

Číslo udávající délku záhlaví IP protokolu. Jednotkou zde však není obvyklý 1 oktet, ale 4 oktety, respektive 32 bitů namísto 8. Jelikož se zde ukládá délka záhlaví v násobcích 32 bitů, musí být záhlaví logicky vždy zarovnáno na tuto hodnotu. Maximální délka záhlaví je $(2^4 - 1) \cdot 32$ bitů, čili v přepočtu 60 oktetů.

3.5.1.3 Typ služby

Označení, jaký typ přepravní služby je pro paket nejvhodnější vzhledem k charakteru přenášených dat. Umožňuje nastavit, zda-li požadujeme vyšší spolehlivost za cenu vyšší odezvy, či vyšší rychlost. Dnešní směrovače na tuto položku neberou zřetel.

3.5.1.4 Celková délka IP datagramu

Číslo udávající maximální velikost IP paketu. Maximální velikost IP paketu je $2^{16} - 1 = 65535$ oktetů.

3.5.1.5 Identifikace IP datagramu

Identifikační číslo přiřazené operačním systémem odesílatele. Je potřeba při případné defragmentaci paketu, aby bylo počítači jasné, které fragmenty patří k sobě a tvoří dohromady jeden celistvý paket.

3.5.1.6 Příznaky

Rezervovaný bit Tento bit je vždy nastaven na hodnotu *0*. Jeho význam mi v rámci hlavičky IP paketu zatím úspěšně uniká.

Don't fragment Je-li nastaven na hodnotu *1*, znamená to, že daný paket nesmí být kýmkoliv fragmentován.

More fragments Je-li nastaven na hodnotu *1*, znamená to, že daný fragment není posledním fragmentem.

3.5.1.7 Posunutí fragmentu od počátku (neboli Offset)

Daný parametr říká, o kolik jsou data přenášená v daném fragmentu paketu posunuta oproti začátku datové části původního nefragmentovaného paketu. Pozor ale na jednotku! Za jednotku se v případě tohoto parametru považuje *8* bajtů, čili `offset` = *1* znamená, že data přenášená daným fragmentem jsou od skutečného počátku dat vzdálená *8* bajtů. Pro daný parametr je v hlavičce IP paketu vyhrazeno *13* bitů.

Co z daných informací vyplývá pro nás? Paket lze rozložit na maximálně $2^{13} = 8192$ fragmentů. To však platí jen za předpokladu, že paket dosahuje své maximální délky 65535 bajtů, jinak bychom neměli co rozkládat. Délka dat fragmentu paketu (mimo toho úplně posledního) musí být násobkem 8 bajtů, neboť offset nám posun o číslo nedělitelné 8 nedovolí vyjádřit. Nefragmentovaný paket má pole `offset` nastaveno na hodnotu 0.

3.5.1.8 Zbývající doba života datagramu (TTL)

Brání nekonečnému přeposílání paketu v síti Internet. Pokud by paket nebyl schopen dosáhnout svého cíle, putoval by sítí donekonečna. Tomuto stavu tato položka hlavičky brání tak, že při každém průchodu směrovačem je daný směrovač povinen snížit tuto hodnotu o nejméně o jedna.

Klesne-li hodnota `ttl` na *0*, směrovač paket zahodí. Následně pošle zprávu o této události odesílateli paketu formou zprávy přenášené ICMP protokolem.

3.5.1.9 Protokol vyšší vrstvy

Číselná identifikace protokolu vyšší vrstvy, jehož hlavička je umístěna na začátku datové části IP paketu. Kódy protokolů, které nás budou zajímat jsou *1* pro ICMP, *6* pro TCP a *17* pro UDP.

3.5.1.10 Kontrolní součet záhlaví

Kontrolní součet záhlaví je nutné spočítat při každé změně záhlaví. A jelikož se záhlaví mění při každém průchodu směrovačem, je počítán takřka neustále. Algoritmus jeho výpočtu je popsán v RFC 1071 [1] a RFC 1141 [3].

3.5.1.11 IP adresa odesílatele

Číslo reprezentující logickou adresu odesílatele.

3.5.1.12 IP adresa příjemce

Číslo reprezentující logickou adresu příjemce.

3.5.1.13 Volitelné položky zarovnání

Rozšiřují záhlaví IP datagramu o řadu dalších voleb. Nejsou povinné. K tomu, aby měly nějaký význam, musí být jejich použití podporováno všemi zařízeními, skrze která datagram při své cestě proteče. Maximální velikost záhlaví IP datagramu je *60* oktetů, z čehož *20* oktetů zabírají povinnné položky. Na volitelné položky zbývá tedy *40* oktetů prostoru.

3.5.2 Fragmentace

Maximální délka IP paketu je $2^{16} = 65535$. Toto omezení je de dáno samotným IP. Platné je všude tam, kde se IP na úrovni síťové vrstvy používá. Samotný přenos dat je však v režii vrstvy síťového rozhraní, která takto velkorysá, co se týče velikosti přenášených dat, nebývá. Navíc protokoly, jenž realizují její činnost se mohou lišit podle toho, o jaký druh sítě se jedná. Kabelové sítě mohou být realizovány protokolem Ethernet a WiFi spoje protokolem 802.11. Tyto protokoly mohou mít odlišné MTU – Ethernet má *1500B*, zatímco 802.11 *2272B*.

Může tedy nastat sitace, kdy odesílatel z IP paketu sestaví rámec s délkou dat, na kterou je zvyklý. To se stává zcela běžně. Problém může přijít ve chvíli, je-li na nějakém úseku cesty k cíli nasazen protokol, jenž definuje menší MTU. Je-li zakázána fragmentace paketu, paket nelze poslat danou cestou a buďto se pošle cestou, která tak velké MTU dovoluje nebo se odešle odesílateli zpráva formou ICMP paketu, že je nutné zmenšit velikost rámce na požadovanou hodnotu, jinak že cíle nebude možné dosáhnout. Není-li zakázána fragmentace, směrovač paket rozdělí na fragmenty, jejich

délka bude rovna nejnižšímu MTU na cestě k cíli. Vysvětlíme si fragmentaci na příkladě.

3.5.2.1 Přišel paket

Dejme tomu, že jsme směrovač, a že nám právě přišel paket o celkové délce 3313 bajtů, z čehož nám vyplývá fakt, že putoval sítí, na které linkovou vrstvou není Ethernet, nýbrž nějaký jiný protokol, který má hodnotu MTU nastavenou minimálně na 3313 bajtů. Paket bychom rádi směrovali do sítě, která má však jako protokol na linkové vrstvě nastaven Ethernet. Daná síť tedy dokáže přenést maximálně 1500 bajtů dlouhý blok dat (+ hlavičku a CRC rámce). Chceme-li paket fragmentovat, musíme se podívat do jeho hlavičky na příznak zvaný `don't_fragment`. Pro nás důležité položky hlavičky jsou vypsány v tabulce 3.1 v prvním sloupci.

3.5.2.2 Vytváříme první fragment

Do nového fragmentu nakopírujeme celou hlavičku, neb se jedná pouze o základních 20 bajtů, jejichž přítomnost je povinná v každém fragmentu. Obsahovala-li by hlavička paketu nějaké volitelné položky, podívali bychom se na to, zda-li mají nastavený příznak kopiruj a zachovali se podle toho. Zbývající délku vyplníme daty tak, abychom dosáhli limitu MTU, konkrétně tedy 1480 bajty z celkových 3293 bajtů dat. Kopírujeme data, kterými datová část původního paketu začínala – bajty na pozicích $< 0, 1480)$, počítáme-li za adresu prvního datového bajtu hodnotu 0. To bychom měli, nyní musíme upravit hodnoty několika položek hlavičky tohoto paketu. Výslednou podobu úprav můžete spatřit ve druhém sloupci tabulky 3.1.

Parametru `identifikace` vždy necháváme jeho původní hodnotu. Slouží k rozlišení paketů, které patří k sobě při následné defragmentaci. Parametr `delka` nastavíme na maximální hodnotu MTU, jež nám linková vrstva sítě, kam bude paket vpuštěn, nabízí – 1500 bajtů. Parametr `delka_hlavicky` nastavíme na základních 20 bajtů. Jedná se o povinnou část hlavičky, která se kopíruje do každého fragmentu za každých okolností. Parametr `offset` ponecháme na jeho stávající hodnotě 0, neb sestavujeme teprve první fragment, jehož data na žádná předchozí data nenavazují. Parametr `don't_fragment` nastavíme na true, neb už se jedná o fragmentovaný paket a co je již jednou fragmentováno vícekrát fragmentovat nelze, nebylo by kam zapsat nové hodnoty parametrů vztahujících se k fragmentaci. Parametr `more_fragments` nastavíme na true, neboť nám zbývá ještě odeslat $3313 - (20 + 1480) = 1813$ bajtů dat. Jinými slovy: „Budou potřeba ještě další pakety.“

	paket	první frag.	druhý frag.	třetí frag.
identifikace	0xABCD	0xABCD	0xABCD	0xABCD
delka	3313	1500	1500	353
delka_hlavicky	20	20	20	20
delka_dat	3293	1480	1480	333
offset	0	0	185	370
don't_fragment	false	true	true	true
more_fragments	false	true	true	false

Tabulka 3.1: Tabulka s některými parametry hlaviček paketů navíc s položkou `delka_dat`, která se v žádné hlavičce nenachází a slouží zde jen pro „lepší vizualizaci“ řešeného problému

3.5.2.3 Vytváříme druhý fragment

Opět nakopírujeme celou hlavičku a dorovnáme daty. Tentokrát již ale daty „druhými v pořadí“, respektive bajty sídlícími na pozicích v intervalu $< 1480, 2960)$. Položky jako jsou `identifikace`, `delka`, `delka_hlavicky`, `don't_fragment` a `more_fragments` zůstanou stejné jako v případě prvního fragmentu. Změna však nastane u položky `offset`. Pozor bychom si měli dát na nezvyklou jednotku použitou u této položky. Jednotkou není očekávaných 8 bitů, čili 1 bajt, ale bitů hned 64, čili bajtů 8. Datům nakopírovým do tohoto fragmentu předchází přesně 1480 bajtů dat obsažených v předchozím fragmentu. `offset` tedy nastavíme na hodnotu $1480/8 = 185$. Po tomto fragmentu nám zbývá odeslat ještě $3293 - 1480 - 1480 = 333$ bajtů dat.

3.5.2.4 Vytváříme třetí fragment

Do třetího fragmentu nakopírujeme hlavičku původního paketu a data. Tento fragment již nebudeme plnit až po okraj stanovený hodnotou MTU, ale pouze zbývajícími daty. Zkopírovány budou bajty na pozicích $< 2960, 3293)$. Hodnotu parametru `offset` nastavíme na $(1480 + 1480)/8 = 370$. Parametry `identifikace` a `delka_hlavicky` ponecháme opět stejné. Parametr `delka` nastavíme na součet délky dat a hlavičky $333+20 = 353$. Parametr `delka_dat` nastavíme na počet bajtů zkopírovaných z původního paketu 333. `more_fragments` nastavíme na false, neboť žádný další fragment již po tomto nebude následovat – neměli bychom do něj co dát. `don't_fragment` necháme nastavený na true. Stále se totiž jedná o již jednou fragmentovaný paket, který není možné dále fragmentovat.

3.5.2.5 Stále však nemáme nafragmentováno

Mohlo by se zdát, že jsme s prací hotovi a že bychom mohli fragmenty paketu již odeslat. Nesmíme ale zapomenout na to, že jsme směrovač. A každý správný směrovač paketu sníží v hlavičce hodnotu `time_to_live` o jedničku. My jsme jí měli snížit již při přijetí paketu, abychom vyloučili situaci, kdy celý paket hezky zfragmentujeme, fragmentům snížíme `time_to_live` o jedničku a následně je zahodíme protože jejich `time_to_live` dosáhl nuly. A taky by nám nemělo uniknout, že sme se paketům hrabali v hlavičkách. Jsme tedy povinni jim přepočítat jejich konstrolní součet s již novými hodnotami.

3.5.3 Defragmentace

Zatímco fragmentovat může kdokoliv na cestě od odesílatele k příjemci, defragmentovat paket smí pouze jeho příjemce. Má to dobrý důvod, a to skutečnost, že jednotlivé fragmenty IP paketu mohou putovat sítí nezávisle na sobě. Pokud by kdokoli jiný než příjemce chtěl paket deftagmentovat, neměl by zaručeno, že bude mít k dispozici všechny fragmenty daného paketu. Příjemce s příchodem všech fragmentů počítat naopak může. Jak ale změť fragmentů IP paketů defragmentovat do celistvých IP paketů? Opřít se můžeme o informace, jež si každý fragment IP paketu musí nést ve své hlavičce. Zatímco volitelné části hlavičky IP paketu nemusí být rozkopírovány do všech fragmentů, ty povinné naopak musí být přítomny v každém z nich. Informace poskytnuté povinou částí hlavičky IP paketu nám zcela pro defragmentaci dostačují.

Stejně tak, jako proces fragmentace, tak i proces defragmentace není nutné rozjíždět při každém přijatém paketu, neb ne každý přijatý paket je fragmentovaný paket. Napoví parametry `offset` a `more_fragments`. Jsou-li oba tyto parametry nulové, jedná se o nefragmentovaný paket, jež není potřeba nijak dál upravovat a jeho obsah (datová část) se může obratem předat vyšší vrstvě. Nastane-li situace opačná, bude nutné defragmentovat.

Fragmenty jednoho paketu patřící k sobě rozpoznáme podle obsahu položky `identifikace` v jejich hlavičce. Fragmenty téhož paketu mají danou položku stejnou. Nastavuje odesílatel a po celou dobu života paketu či jeho fragmentů je neměnná. Pozor si musíme dát na skutečnost, že první příchozí fragment obecně nemusí nést první bajty z datové části. Během své cesty k příjemci paketu může být jakýmkoli svým sourozencem (čti „fragmentem vzešlým ze stejného paketu“) předběhnut. Bylo by vhodné si ukázat proces defragmentace na nějakém příkladě. Budiž nám nápomocen příklad z části o fragmentaci. Řekněme, že nám pakety přijdou v pořadí 2. 3. a 1.

účel	velikost [B]
prostor pro uložení hlavičky paketu	60
prostor pro uložení datové části paketu	65515 (65535-20)
prostor pro uložení evidence již dorazivších paketů	1024
prostor pro uložení parametru „identifikace“	2 (16b)

Tabulka 3.2: Prostor, jenž je potřeba pro defragmentaci příchozího paketu

3.5.3.1 První paket přichází

Přívlastkem „první“ zde není myšleno pořadí jím nesených dat v konečném IP paketu, ale pouze to, že daný fragment k nám dorazil jako první z časového hlediska. Nejprve si zjistíme, zda-li již defragmentace paketu s parametrem `identifikace` = *0xABCD* již neprobíhá. Pokud by probíhala, budeme se k danému fragmentu chovat podle scénáře v sekci 3.5.3.2. V našem případě neprobíhá.

Je tedy nutné začít defragmentaci zcela od počátku. Abychom mohli defragmentovat, musíme si v paměti alokovat prostor, do nějž si budeme ukládat informace o průběžně přicházejících fragmentech výsledného paketu. Co všechno budeme potřebovat je uvedeno v tabulce 3.2.

Pro hlavičku naalokujeme obecně maximální možný prostor *60* bajtů. Uděláme tak i přesto, že námi přijatý fragment má parametr `delka_zahlavi` nastaven na *20*. Víme totiž, že nepovinná část hlavičky nemusí být přítomna v každém z fragmentů. Jediný fragment, kde hlavička musí být ve své plné podobě, je fragment nesoucí první bajty dat. Pro datovou část paketu si naalokujeme maximální možnou hodnotu *65515* bajtů. Stejně jako v případě hlavičky, ani u datové části paketu nevíme, jak dlouhá nakonec tato část ve výsledném paketu bude. Dále si uložíme hodnotu parametru `identifikace` právě přišlého fragmentu. Podle ní pak budeme moci zjistit při příchodu dalších fragmentů, je-li proces defragmentace výsledného paketu s danou hodnotou `identifikace` již nastartován či nikoliv.

K možnosti defragmentovat paket nám stále něco chybí. V průběhu defragmentace si potřebujeme nějak zaznamenávat, které fragmenty již dorazily (a my si je nakopírovali do paměti) a na které se ještě čeká. Jinak bychom nevěděli, na čem jsme a zda-li již není práce hotova. K tomuto poznamenávání si můžeme sestrojit jakoukoliv strukturu. Jelikož je parametr `offset` dlouhý *13* bitů, dovolí nám rozlišit $2^{13} = 8192$ příchozích fragmentů. Na evidenci, zda-li daný fragment již přišel či nikoliv, nám postačí *1* bit. Celkem tedy postačí naalokovat prostor o délce $8192/8 = 1024$ bajtů. Tento prostor si celý vynulujeme.

Podíváme-li se na položku `offset`, zjistíme, že její hodnota není rovna

0, nejedná se tedy o první paket. Hlavičku fragmentu tedy kopírovat do paměti nebudeme. Co naopak do paměti zkopírujeme, je datová část, jejíž délku vyčteme z parametrů `celkova_delka` a `delka_hlavicky`. Zkopírujeme jí do prostoru „pro uložení datové části paketu“ na přesně definované místo. Obecně do intervalu adres:

$$\langle offset \cdot 8, offset \cdot 8 + (celkovadelka - delkahlavicky)]$$

V naší konkrétní situaci, kdy nám přišel druhý paket ze tří:

$$\langle 185 \cdot 8, 185 \cdot 8 + 1500 - 20] = \langle 1480, 2960]$$

Nyní si musíme poznačit, kterou část paketu jsme již přijali. Při příchodu fragmentu se podíváme na hodnoty, jenž si s sebou přinesl v položkách `offset`, `delka_hlavicky` a `celkova_delka`. V poli bitů, jenž eviduje již přišlé fragmenty nastavíme na jedničku bity z následujícího intervalu:

$$\langle offset, offset + (celkovadelka - delkahlavicky)/8]$$

V naší konkétní situaci nastavíme na jedničku bity z intervalu:

$$\langle 185, 185 + (1500 - 20)/8] = \langle 185, 370]$$

3.5.3.2 Druhý paket přichází

Nyní nám přijde paket, jenž byl v příkladu o fragmentaci označován jako třetí. Podle parametru `identifikace` obsaženého v jeho hlavičce zjistíme, že defragmentace paketu s danou identifikací tu již probíhá. Stejně jako u předcházejícího fragmentu, zkopírujeme datovou část současného fragmentu podle obsahu parametrů `offset`, `delka_hlavicky` a `celkova_delka`. Datová část daného paketu má délku 333 bajtů. V poli pro data tedy zabere pozice v intervalu:

$$\langle 370 \cdot 8, 370 \cdot 8 + (353 - 20)] = \langle 2960, 3293]$$

V tabulce bitů označujících, které fragmenty již přišly nastavíme na jedničku bity:

$$\langle 370, 370 + (353 - 20)/8] = \langle 370, 412]$$

Jelikož je parametr `more_fragments` nastaven tentokrát na nulu, jedná se o fragment poslední. Díky tomu může ten, kdo defragmentuje zjistit celkovou velikost výsledného paketu:

$$celkovadelkavyslednehopaketu = offset \cdot 8 + (celkovadelka - delkazahlavi)$$

Konkrétně v našem případě:

$$celkovadelkavyslednehopaketu = 370 \cdot 8 + (353 - 20) = 3293$$

Po příjmu paketu, jenž nese poslední data z původního paketu dostáváme možnost zkontrolovat, zda-li je paket již kompletně přijat. Tuto informaci zjistíme tak, že všechny bity v poli, jenž nám uchovává informaci o tom, které fragmenty již přišly budou od adresy *0* do adresy *celkovadelkavyslednehopaketu/8* nastavené na jedničku. Zatím tomu tak ale není. Čekáme na přijetí posledního chybějícího fragmentu.

3.5.3.3 Třetí paket přichází

Zbývá nám již jen příchod původně prvního fragmentu z původního paketu. Skutečnost, že je byl daný parametr první můžeme zjistit z hodnoty parametru `offset`, jenž je u prvního fragmentu vždy *0*. Tento fragment jako jediný je povinen nést s sebou kompletní hlavičku, kterou my zkopírujeme do předem vytvořeného místa. Kopírovat se bude samozřejmě skutečná velikost hlavičky – `delka_hlavicky` bajtů. V našem případě *20* bajtů. Taktéž nastavíme na jedničku bity značící příchod fragmentu s odpovídajícími indexy:

$$\langle 0, 0 + (1500 - 20)/8] = \langle 0, 185]$$

Nyní máme v poli bitů signalizujících příchod všech fragmentů všechy potřebné bity nastaveny na jedničku. Paket tedy dorazil vcelku. Už nám zbývá jen mu v hlavičce upravit parametr `delka` na celkovou skutečnou délku. Nahradíme tedy *1500* hodnotou $3293 + 20 = 3313$.

3.6 Protokol ICMP

ICMP je protokol internetové vrstvy komunikačního modelu TCP/IP. Slouží k diagnostickým a servisním účelům na úrovni protokolu IP. Jeho prostřednictvím se oznamují nejrůznější chybové stavy.

Protokol ICMP se vymyká zavedeným pořádkům. Jelikož jsou jeho data přenášena v těle IP paketu, mohlo by se zdát, že jde o protokol transporní vrstvy. Nicméně není tomu tak. Jako protokol transportní vrstvy by musel být schopen zajišťovat přenos („transport“) dat aplikační vrstvy. Jelikož jeho hlavička nenese žádná čísla portů, která by mohla sloužit k adresaci procesů běžících na komunikujících zařízeních, přenosu dat aplikační vrstvy není schopen. Je tedy řazen do vrstvy internetové/síťové.

3.6.1 Položky hlavičky

3.6.1.1 Typ ICMP paketu

Prvních *8* bitů hlavičky je vyhrazeno pro definici typu ICMP paketu. Každý typ se užívá k jinému účelu. Jejich výčet je uveden v kapitole

3.6.1.2 Kód

Dalších *8* bitů pro bližší specifikaci typu ICMP paketu.

3.6.1.3 Kontrolní součet

Položka umožňující detekci případného poškození paketu.

3.6.1.4 Proměnné atributy

Celé *4* bajty vyhrazené pro parametry, jenž může mít každý typ ICMP paketu různé. Jejich „složení“ je odvislé od specifikace typu 3.6.1.1 a podtypu 3.6.1.2.

3.6.2 Typy ICMP paketů

Zcela vyčerpávající popis je uveden v RFC 792 [5]. Kdo se nechce nechat zbytečně vyčerpat, tomu bude muset stačit následujících pár příkladů.

3.6.2.1 Odpověď na žádost (*0*)

ICMP paketem tohoto typu odpovídá zařízení, jemuž se dostalo žádosti o odpověď (typ *8*). Položku `kod` nastavuje vždy na hodnotu *0*.

3.6.2.2 Zpráva o nemožnosti doručení paketu (*3*)

Zjistí-li nějaký směrovač, že není možné doručit určitý paket, paket zahodí a pošle odesílateli ICMP paketem zprávu o tom, že se doručení nezdařilo a proč. K tomu je používán právě ICMP paket typu *3*. Přesnější určení nastalého problému se přitom uloží do položky `kod`.

Důvodem bývá často buďto neexistence adresáta v cílové síti nebo to, že cestou k adresátovi je na některé z sítí MTU nižší než byla velikost zahozeného paketu. Pokud má paket ve své hlavičce nastaveno `don't_fragment` na false, směrovač může paket místo zahazování fragmentovat a pokusit se ho doručit.

3.6.2.3 Žádost o snížení rychlosti odesílání (*4*)

Tento typ ICMP paketu obvykle zasílá stanice nemající dostatek místa ve vstupním bufferu a nestíhající všechny příchozí pakety zpracovávat.

3.6.2.4 Žádost o změnu směrování (*5*)

Zasílá směrovač odesílateli paketu, zjistí-li, že existuje jiná rychlejší cesta k cíli. Odesilatel by se po přijetí této zprávy měl zařídit podle jejího pokynu a odesílat další pakety doporučeným směrem.

3.6.2.5 Žádost o odpověď (*8*)

Někdy také označováno jako „Žádost o echo“. Daný typ použijeme, chceme-li zjistit, zda-li zařízení s určitou IP adresou v síti existuje či nikoliv. Každé zařízení, které obdrží tuto žádost, by na ní mělo odpovědět ICMP paketem typu *0*. V případě tohoto typu ICMP paketu jsou *4* bajty proměnných atributů nositeli dvou následujících položek:

Identifikátor ICMP paketu zabírající prvních *16* bitů by měl být nastaven na zcela unikátní hodnotu, aby bylo možné rozlišit mezi více případnými odpověďmi přicházejícími od dotazované stanice. To, abychom rozlišili, která odpověď patří ke kterému požadavku, neb dané číslo odpovídající stanice vloží do téhož parametru ve své odpovědi. Aby byla zaručena absolutní unikátnost vyplňovaného čísla, nastavuje se tato položka zpravidla na číslo identifikující onen žádající proces v rámci počítače.

Pořadové číslo žádosti rozlišuje více žádostí o odpověď. To nám umožňuje jich zaslat více za sebou. Odpovídající stanice toto číslo zapracuje do své odpovědi (umístí ho téhož volitelného parametru).

3.6.2.6 Čas vypršel (*11*)

Informuje odesílatele, že jím odeslaný paket se nedožil svého doručení. Tuto zprávu posílá odesílateli paketu směrovač, u něhož se čítač `ttl` v hlavičce paketu sníží na hodnotu *0*. Takový paket je jednoduše zahozen, protože se předpokládá, že již cestuje sítí bez možnosti dosáhnout svého cíle.

3.6.2.7 Signalizace chybně vyplněné hlavičky IP paketu (*12*)

Oznámí odesílateli paketu, že jím odeslaný paket měl chybně vyplněnou hlavičku. Přesnější určení chyby má na starost parametr `kod`.

pojmenování hodnoty	bitů	příklad
adresa síťové vrstvy	32	0xC0A80001
protokol transportní vrstvy	16	0x0011
adresa transporní vrstvy (port)	16	0x0050

Tabulka 3.3: Položky obsažené v adrese procesu

3.7 Protokoly transportní vrstvy

Nejprve si připomeneme úlohu transportní vrstvy v síťové architektuře a následně se budeme podrobněji zajímat o dva význačné protokoly, jež funkce této vrstvy realizují, – TCP a UDP.

3.7.1 Úloha transportní vrstvy

Transportní vrstva tak jako každá vrstva užívá služeb vrstev nižších a poskytuje služby vrstvám vyšším. Zatímco na síťové vrstvě dnes dominuje protokol jediný, i když v různých verzích, tak pole transportní vrstvy obsadily protokoly dva – TCP a UDP, z nichž si aplikační vrstva má možnost vybrat. Oba dva nabízí řešení jejího problému – přenosu dat, každý ale po svém.

Jaký problém dané protokoly vlastně řeší? Jejich úkolem je přenos dat mezi aplikacemi (přesněji řečeno mezi procesy). Zatímco protokol IP zajišťuje adresaci komunikujících stran na úrovni koncových stanic („počítačů“), protokoly transportní vrstvy hrají svojí roli při adresaci jednotlivých procesů běžících na daných stanicích. Jednotlivé po síti komunikující procesy jsou v rámci stanic rozlišeny číslem, zvaným „port“ – adresou transportní vrstvy.

3.7.2 Adresa transportní vrstvy

Adresa transportní vrstvy – port – může nabývat hodnot z intervalu

$$\langle 0, 2^{16}] = \langle 0, 65536]$$

Každý protokol transportní vrstvy má k dispozici celý rozsah pro sebe, z čehož plyne, že adresa konkrétního procesu na konkrétní stanici má celkem tři části, jejichž příklad můžete vidět v tabulce 3.3.

Na jednom rozhraní s jednou IP adresou může teoreticky naslouchat $2^{16} \cdot 2^{16}$ procesů. Proces je tedy jednoznačně identifikován celkem 8 bajty. V rámci jedné stanice je identifikován 4 bajty.

3.7.3 Jednotka transporní vrstvy

Transportní vrstva pracuje s jednotkou nazývanou „segment“. Toto seskupení dat sestává z hlavičky a datové části, kam jsou vkládána data vyšší vrstvy. TCP i UDP protokol připouští maximální délku segmentu rovnu $2^{16} - 1 = 65535$ bajtů. Z této maximální délky segmentu si svoje vezme také hlavička segmentu, jejíž délka je v případě UDP vždy *8* bajtů, v případě TCP v rozmezí *20* až *60* bajtů. Pro data aplikační vrstvy tak zbývá prostor $\langle 65535 - 8, 65535 - 60] = \langle 65527, 65475]$. Strukturou hlaviček segmentů se budeme zabývat v následujících kapitolách. Datová část je pro nás nezajímavá.

3.7.4 Protokol UDP

Zajišťuje přenos dat aplikační vrstvy bez nutnosti navazování spojení. To se může hodit v situaci, kdy by zdržení způsobené čekáním na každý cestou ztracený jednotlivý paket bylo na obtíž. Tento protokol se využívá například při audio/video přenosech (VoIP), kde ztráta pár paketů nebolí a řeší se prostým vynecháním paketu s tím, že informace jím nesené se dopočítají. Dále se protokol UDP používá například v systému zajišťujícím překlad doménových jmen – DNS. V případě systému DNS by navazování spojení znamenalo vzhledem k délce přenášených dat neúměrnou režii navíc. Nepřišlou odpověď si v tomto případě však nelze „dopočítat z prstu“ a situaci řešíme opětovným zasláním dotazu.

Jelikož protokol UDP nezná pojem „navazování spojení“ jako třeba protokol TCP, je považován za bezestavový. Jeho jediným posláním je odeslat data. To, zda-li daná data dorazila ve správném pořadí či zda-li vůbec dorazila protokol UDP neřeší, z čehož vyplývá jeho nižší režie, nežli je tomu u protokolu TCP. To jej činí velmi jednoduchým na implementaci.

Pojďme si nyní popsat účel jednotlivých parametrů obsažených v hlavičce UDP segmentu.

vložit obrázek!

3.7.4.1 Zdrojový port

Určení portu, na němž sídlí odesílatel – proces, jenž inicioval odeslání daného UDP segmentu. Zabírá *16* bitů hlavičky.

3.7.4.2 Cílový port

Určení portu, na němž by měl sídlit příjemce daného UDP paketu. Zabírá *16* bitů hlavičky.

3.7.4.3 Celková délka segmentu

16 bitů vyhrazených pro uložení informace o délce segmentu včetně obou jeho částí – hlavičky i prostoru pro data vyšší vrstvy.

3.7.4.4 Kontrolní součet

Kontrolní součet celého segmentu včetně hlavičky i datové části. Je-li tato položka nastavena na hodnotu *0*, má se za to, že kontrolní součet nebyl spočítán. Komunikujícím stranám by tato skutečnost neměla vadit.

3.7.5 Protokol TCP

Protokol nabízí aplikační vrstvě zcela odlišný přístup. Od přístupu reprezentovaného protokolem UDP se odlišuje v následujících bodech:

TCP je spojovanou službou. Pro přenos dat vyžaduje navázání spojení. To si vyžádá jistou režii. Po dobu, kdy je spojení navázáno, se mohou data přenášet duplexně – oběma směry současně a nezávisle na sobě.

TCP ručí vyšší vrstvě za úplnost přenášených dat. Nedorazila-li nějaká část dat, protokol TCP o ně žádá opakovaně do doby, kdy přijdou nebo do doby, kdy vyprší timeout spojení.

TCP ručí vyšší vrstvě za správné pořadí dat. To je zajištěno tím, že odesílatel data vždy čísluje.

Z výše uvedených vlastností vyplývají jednoznačně vyšší nároky na výpočetní výkon a vyšší složitost jeho implementace ve srovnání s protokolem UDP. Používá se všude tam, kde je bezpodmínečně požadována spolehlivost datového přenosu. Nyní se podívejme na jednotlivé položky hlavičky TCP segmentu.

3.7.5.1 Zdrojový port (*16b*)

Port odesílatele segmentu.

3.7.5.2 Cílový port (*16b*)

Port adresáta segmentu.

3.7.5.3 Pořadové číslo odesílaných dat (*32b*)

Číslo prvního bajtu obsaženého v datové části relativně vůči prvnímu bajtu všech odesílaných dat. Číslování začíná od náhodně zvoleného čísla.

3.7.5.4 Pořadové číslo přijatých dat(*32b*)

Číslo bajtu, který je příjemce připraven přijmout. Udáním tohoto čísla příjemce potvrzuje, že přijal všechny bajty předcházející danému bajtu.

3.7.5.5 Délka záhlaví (*4b*)

Číslo určující délku záhlaví TCP segmentu. Jednotkou jsou zde *4* bajty. To znamená, že délka *20* bajtů (délka povinné části hlavičky TCP segmentu) se v této položce vyjádří jako *0x0005*.

3.7.5.6 Rezerva (*6b*)

Zbytečně promrhaných *6* bitů.

3.7.5.7 Příznaky

URG segment nese naléhavá data

ACK hodnota nesená v `pořadové číslo přijatých dat` je platná

PSH segment nese aplikační data

RST spojení odmítnuto

SYN nové spojení

FIN ukončení spojení

3.7.5.8 Délka okna (*16b*)

3.7.5.9 Kontrolní součet (*16b*)

Hodnota kontrolního součtu se počítá pouze z přenášených dat.

3.7.5.10 Ukazatel naléhavých dat (*16b*)

3.7.6 TCP spojení

Jak je již uvedeno výše – protokol TCP zřizuje spojovou službu. Nese odpovědnost za následující činnosti:

- navázání spojení
- potvrzování přijatých dat
- vyžádání ztracených dat
- ukončení spojení

3.7.6.1 Navázání spojení

Iniciátor spojení pošle paket s následujícími hodnotami v TCP segmentu.

```
ACK = 0
SYN = 1
FIN = 0
pořadové číslo odeslaného bajtu = x (libovolná hodnota)
pořadové číslo přijatého bajtu = 0
```

Příznak `SYN` příjemci sdělí, že se jedná o paket navazující doposud nezapočaté spojení. Příjemce si poznamená hodnotu `pořadového čísla odeslaného bajtu`. Do své odpovědi vloží následující hodnoty.

```
ACK = 1
SYN = 1
FIN = 0
pořadové číslo odeslaného bajtu = y (libovolná hodnota)
pořadové číslo přijatého bajtu = x+1
```

Tato kombinace hodnot iniciátorovi spojení sdělí, že jeho žádost o navázání spojení byla akceptována. Parametr `pořadové číslo přijatého bajtu` říká adresu další očekávané hodnoty $x + 1$. Původní iniciátor spojení potvrdí svůj úmysl založit spojení ještě odesláním..

```
ACK = 1
SYN = 0
FIN = 0
pořadové číslo odeslaného bajtu = x+1
pořadové číslo přijatého bajtu = y+1
```

Kapitola 4

Návrh aplikace

V této kapitole se rozhodneme, jakou podobu by měla naše výsledná aplikace mít, jak si představujeme její komunikaci s uživatelem, čím jí napíšeme a jaký všechen cizí kód v našem díle využijeme.

4.1 Uživatelské rozhraní

Cesty k uživateli máme celkem tři. Mohli bychom vytvořit aplikaci s grafickým rozhraním, kde si uživatel vybere, co chce dělat především pomocí myši – výběrem z rozbalovacích nabídek a radio buttonků. Tato cesta nám slibuje snadné a rychlé ovládání, absolutní nemožnost použití programu ve skriptech, nefunkčnost v negrafickém prostředí.

Druhou možností je udělat aplikaci ovladatelnou skrze parametry příkazové řádky. Unixu neholdujícímu uživateli se to nebude líbit, bude nutné znát spoustu přepínačů či syntaxi konfiguračního souboru. Narozdíl od varianty s GUI umožní tato cesta použití programu ve skriptech.

Jelikož jsme nikdy ve své historii nic grafického nevytvořili a vůbec si neumíme představit, co všechno to obnáší, nebudeme riskovat a zvolíme variantu číslo dvě. Aplikace bude s uživatelem komunikovat skrze konzoli. Vzhledem k tomu, že aplikace by měla umět přijímat a vypisovat pakety či je generovat dle zadání, nebude interakce s uživatelem za běhu aplikace v naprosté většině času nutná. Uživatel totiž aplikaci předá všechny pokyny při jejím spuštění formou konfiguračního souboru.

4.2 Volba programovacího jazyka

Program jsem původně chtěl napsat v jazyce Python, jehož možnostmi a elegancí jsem byl okouzlen v předmětu BI-PYT. Lákala mě především představa jednoduchého rozparsování a kontroly syntaxe konfiguračního souboru. Tato část práce by byla v Pythonu rozhodně elegantní a ne příliš složitá. Druhou přípustnou alternativou byl jazyk C++, ke kterému jsem měl zcela neutrální vztah. Vedoucí mé práce mě přesvědčil k použití C++, dnes už ani nevím jak. Nicméně dobře tomu tak. Python by sice vypadal „esteticky lépe“, ale vykonával by se déle a co nejhůř, zdaleka by mi nedovoloval tak jednoduchý nízkoúrovňový přístup k jednotlivým bajtům pomocí ukazatelů, jako nabízí C++. Nakonec jsem ani pro rozparsování konfiguračního souboru nezvolil Python, přestože jsem to měl dlouho v plánu. Neb možnosti C++ které jsem časem objevil v rámci parsování konfiguračního souboru nejsou vůbec špatné.

4.3 Užité knihovny třetích stran

4.3.1 pcap

Program je zcela závislý na knihovně zvané pcap. Tato knihovna je napsána v jazyce C a je spravována tvůrci programu tcpdump. Implementuje funkce a datové struktury, které slouží k nízkoúrovňovému přístupu na síťové rozhraní. Na unixových platformách je pcap implementována knihovnou libpcap. Na platformě Windows pak knihovnou winpcap.

Ačkoliv toho tato knihovna poskytuje mnoho funkcionality, aplikace, která je předmětem této bakalářské práce jí využívá jen z velmi malé části. Dokonce bych řekl, že tak malé, že je možné si použité funkce a datové objekty z dané knihovny rovnou lehce okomentovat, ať víme na čem budeme stavět.

4.3.1.1 `int pcap_findalldevs(pcap_if* ukazatel, char* buffer)`

Funkce poskytující spojový seznam dostupných zařízení, na která se lze „napojit“ funkcí `pcap_open_live()`. Funkce má dva vstupně-výstupní parametry – `ukazatel`, který ukazuje na začátek spojového seznamu zařízení. Pakliže hledání zařízení skončí nezdarem, návratová hodnota funkce bude nastavena na hodnotu *1* a parametr `buffer` bude naplněn informací o chybě. V ukazateli, který nám tato funkce nastaví, nás budou zajímat položky `(*ukazatel).name` (název zařízení) a `(*ukazatel).next` (ukazatel na další nalezené rozhraní).

4.3.1.2 `void pcap_freealldevs(pcap_if* ukazatel)`

Dealokuje spojový seznam vytvořený funkcí `pcap_findalldevs()`. Jediným parametrem je ukazatel na začátek daného spojového seznamu `ukazatel`.

4.3.1.3 `pcap_t* pcap_open_live(char* a, int b, int c, int d, char* e)`

Funkce otevírající rozhraní, jehož název je předáván ve formě řetězce jako parametr `a`. Na OS Linux mívají tato rozhraní krátká pojmenování typu `eth0` či `wlan0`. Na platformě Windows naopak názvy tvoří dlouhé a těžko zapamatovatelné řetězce typu

```
\Device\Tcpip_{EF8BD219-1D00-4EE2-A755-DD343B646785}
```

Tento parametr je nastaven na hodnotu, jež zvolí uživatel při (nebo těsně po) startu programu.

Parametr `b` definuje maximální počet bajtů, které budou z přícozího objektu (rámce) zachyceny. Pohybujeme-li se na síti, jejíž linková vrstva je realizována protokolem Ethernet, postačí zde uvést hodnotu *1514*, což je délka ethernetového rámce včetně hlavičky a kontrolního součtu. Zadáme-li zde například číslo *34*, bude zachytávána pouze hlavička rámce a poviné položky hlavičky paketu. Tento parametr je v programu nastaven na hodnotu *1514*.

Nastavíme-li `c` na nenulovou hodnotu, přepneme tím síťovou kartu do promiskuitního módu. To znamená, že nám karta bude doručovat i pakety, jenž nejsou určeny přímo nám (jí). O tuto funkcionalitu nemáme zájem, nastavíme tedy na hodnotu *0*.

Parametrem `d` nastavujeme čas (v milisekundách). Po dobu, na jakou ho nastavíme bude funkce přijímající jednotlivé rámce `pcap_next()` čekat na jakýkoliv příchozí paket. Pakliže jí po celou tuto dobu nic nepřijde, oznámí programu, že nic nepřijala.

A konečně parametr `e` představuje ukazatel na pole znaků, kam se má uložit případný chybový výpis. Tomu nastane v případě, když funkce vrátí hodnotu *0*. Naopak v případě úspěšného otevření rozhraní funkce vrátí ukazatel na něj (jakýsi deskriptor rozhraní).

4.3.1.4 `int pcap_sendpacket(pcap_t* adapter, u_char* buffer, int delka)`

Hlavní funkce, na které je celý program postaven. Umožňuje mu totiž odesílat jím předpřipravené rámce v síť. Odesílá skrze rozhraní reprezentované

ukazatelem adapter rámec délky delka uložený v paměti na adrese buffer. Rámec je odesílán přesně v tom tvaru, v jakém je uložen v paměti. Jediné, co při neobsahuje je zakončení s CRC součtem. Ten rámci přidá ovladač síťové karty těsně před jeho odesláním. V případě zdaru funkce vrací hodnotu *0*, v opačném případě hodnotu *−1*.

4.3.1.5 u_char* pcap_next(pcap_t* adapter, struct pcap_pkthdr* info)

Funkce umožňující přijímat rámce. Vrací ukazatel na rámec zachycený ve formě řetězce. V případě nezdaru, respektive pakliže nic po celou dobu timeoutu (nastavovaném u funkce pcap_open_live() nepřišlo, vrací hodnotu *0*. Parametr adapter je ukazatelem na rozhraní, na němž se má naslouchat. Struktura info je vstupně-výstupním parametrem, do něhož jsou uloženy dodatečné informace vztahující se k přijateému paketu.

Zde bude nutné si podrobněji popsat onu strukturu pcap_pkthdr*, jež obsahuje klíčové informace o přijatém paketu.

```
struct pcap_pkthdr
{
    struct timeval  ts;
    bpf_u_int32     caplen;
    bpf_u_int32     len;
};
```

Položka caplen udává počet zachycených bajtů rámce. Maximální počet bajtů, které chceme z každého rámce zachytit se nastavuje při otevírání rozhraní funcí pcap_open_live(). V ethernetové síti by to mělo postačovat maximálně *1514*. CRC na konci rámce je ignorováno. Položka len udává skutečnou délku rámce. Pokud se položky caplen a len rovnají, tak jsme zachytili rámec celý. Pokud se nerovnají, máme pouze prvních caplen bajtů z celkových len bajtů.

Položka ts obsahuje čas, kdy byl paket zachycen. Tato informace je sice možná zajímavá, ale pro nás nepodstatná. Pokud by jsme však uznali, že by bylo vhodné uživatele informovat o času přijetí paketu, pak bychom se museli seznámit s další strukturou timeval.

4.3.1.6 pcap_close(pcap_t* a)

Funkce uzavře rozhraní, jehož identifikátorem je ukazatel na strukturu pcap_t* a, jenž byl vytvořen při otevírání daného rozhraní.

Kapitola 5

Implementace

V zadání práce je řečeno, že program by měl být funkční na obou dnes dominantních platformách – OS Linux a OS Windows. Tyto dva operační systémy se pochopitelně nechovají zcela stejně. Co nabízí jeden, nenabízí druhý. Zde by se možná slušelo dodat „a naopak“, ale myslím, že toto vystihuje situaci lépe. Jak danou situaci vyřešit? Naštěstí byli návrháři jazyka `C/C++` prozíraví a implementovali podmíněný překlad.

5.1 Podmíněný překlad

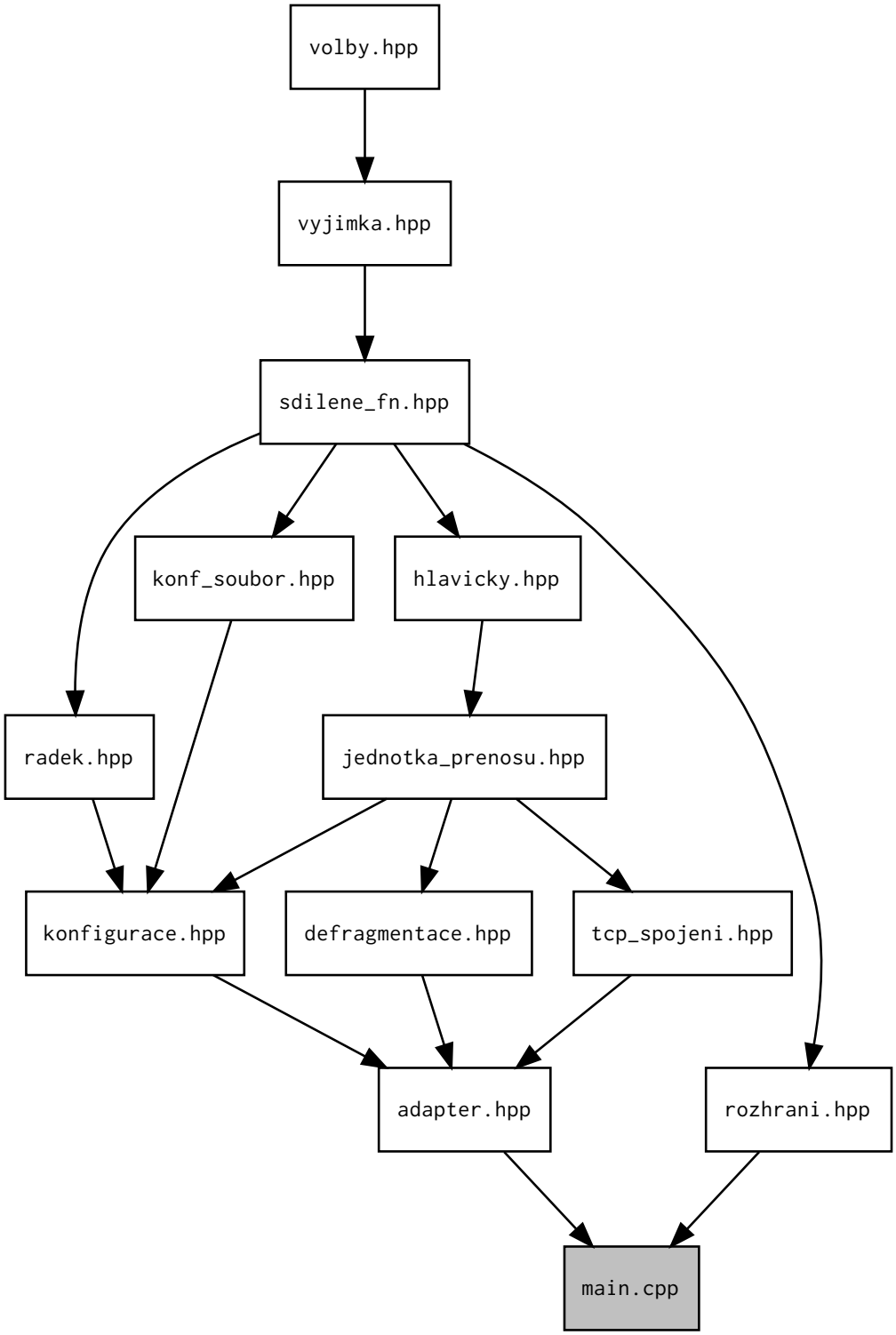
Podmíněný překlad umožňuje napsat multiplatformní aplikaci bez větší námahy. Funguje to tak, že překladač může vzít při kompilaci programu v úvahu, je-li v prostředí jazyka definováno to či ono. Program využívá toho, že makro `WIN32` nebývá zpravidla při kompilaci na OS Linux definováno.

```
#ifdef WIN32
    #include <winsock2.h>
#else
    #include <arpa/inet.h>
    #include <unistd.h>
#endif
```

Zde vidíme konstrukci, která je v rámci programu použita nesčetněkrát.

5.2 Zdrojový kód programu

Program sestává z několika souborů, jejichž závislosti jsou zobrazené na obrázku 5. Šipky znázorňují použití direktivy `include` v rámci programu.



Obrázek 5.1: Vzájemné závislosti hlavičkových souborů

Na obrázku jsou zobrazeny jsou pouze hlavičkové soubory `*.hpp` (vyjma `main.cpp`). Z obrázku lze vyčíst například to, že soubor `sdilene_fn.hpp` využívá funkcionality poskytované souborem `vyjimka.hpp` a tedy přeneseně i souborem `volby.hpp`.

Každý hlavičkový soubor je direktivou `include` vkládán navíc do *0* až *5* souborů `*.cpp`, které definují prostředky, které přidružený hlavičkový soubor pouze deklaruje. Všechny hlavičkové soubory mají nějaký svůj protějšek v `*.cpp` souboru mimo souboru `volby.hpp`, jež všechen svůj obsah místo deklarování rovnou definuje. Nyní si některé soubory projdeme a popíšeme, jakou úlohu ve výsledném programu zastávají.

O souboru má smysl mluvit pouze v případě `volby.hpp`, `sdilene_fn.hpp`, `sdilene_fn.cpp` a `main.cpp`. Ostatní soubory obsahují vždy jen definice/deklarace určitých tříd. Popis funkcionality nabízené těmito soubory lze nahradit popisem v nich obsažených tříd.

5.2.1 Soubor `volby.hpp`

Tento soubor představuje základní stavební kámen celého programu. Právě zde se vkládají do kódu hlavičkové soubory všech v programu užitých knihoven. Dále se zde definují všechny konstanty v programu používané, což umožňuje jejich snadnou změnu. Program jsem se snažil psát tak, aby se nikde nevyskytovaly magické konstanty, nebyla by-li hodnota, kterou by případně reprezentovaly obecně nezaměnitelná jinou. Změna se samozřejmě projeví až po pře-kompilaci programu. Lze zde například centrálně změnit maximální možnou délku přenosové jednotky `RAMEC_DATA_MAX`.

```
#define RAMEC_DATA_MAX  1500
#define PAKET_MAX       65535
#define IP4_DEFAULT_TTL 255
```

Dále daný soubor obsahuje definice pár jednoduchých datových typů, které jsou napříč celým programem používané.

```
typedef u_int8_t u8;
typedef u_int16_t u16;
typedef u_int32_t u32;
```

5.2.2 Soubor `main.cpp`

V tomto seouboru se na základě uživatelem zadaných parametrů zjistí, zdali uživatel chce spustit program v reživu „generátor“ či v režimu „přijímač“. Podle toho se budou volat potřebné části programu.

5.2.3 Soubory `sdilene_fn.hpp` a `sdilene_fn.cpp`

Zde jsou umístěny všechny funkce, jež jsou sdíleny ve více jiných souborech. Obecně se jedná o funkce, jež vykonávají následující činnosti:

- obarvování standardního výstupu
- převod čísel na řetězce a naopak
- jednoduché nastavování hodnot jednotlivých bitů v bajtu

5.2.4 Třída `vyjimka`

Tato třída se využívá při detekci chyb. Jakmile je při chodu programu nalezen nějaký závažný problém, program daný problém ohlásí skrze tuto třídu. Jakmile je objekt dané třídy vytvořen, už není cesty zpět a program se ukončí.

5.2.5 Třída `konf_soubor`

Stará se o správné načítání uživatelského vstupu z konfiguračního souboru. Zadané hodnoty sice načte, nicméně již nekontroluje, zda-li jsou načtené hodnoty vůbec přípustné. Hodnoty se testují až při samotném vytváření datových struktur (paketů/rámců), které se budou odesílat do sítě.

5.2.6 Třída `hlavicka` a její podtřídy

Tato třída (resp. její podtřídy) je programem užívána k uložení navzájem souvisejících dat. Jedna instance dané třídy může například nést obsah hlavičky IP paketu, obsah hlavičky TCP segmentu či pouze holá data. Na úrovni této třídy se testuje uživatelský vstup.

5.2.7 Třída `jednotka_prenosu`

Tato třída žádnou velkou funkcionalitu neposkytuje, ač by měla. Moc by se jí hodil například kopírující konstruktor a přetížený operátor přiřazení. Slouží pouze jako kontejner pro přenos ukazatelů na hlavičky jedné jednotky. Jednou „jednotkou přenosu“ zde může být myšlen:

- rámec
- fragment paketu
- paket

- hlavičky + data, jež je nutné přenést ve více paketech

Dle přítomnosti a délky obsažených hlaviček se program rozhoduje, o jaký typ z výše uvededených se jedná.

5.2.8 Třída `radek`

Tato třída parsuje obsah jednoho řádku konfiguračního souboru. Původně byl konfigurační soubor zamýšlen tak, že uživatel bude moci zadávat parametry bez jejich pojmenování. Program následně měl dané parametry načítat a pouze dle jejich pořadí jim přidělovat význam. Sice to takto může doteď fungovat, ale toto rozhodnutí bylo velmi krátkozraké. Funguje to u velmi krátkých konfigurací, kdy člověk chce, aby mu program generoval jednoduché rámce či IP pakety s dále nepříliš definovaným obsahem. Když dojde na ICMP, UDP či TCP, tak si uživatel pořadí parametrů v jejich hlavičkách rozhodně však pamatovat nemůže. Celá tato třída se v tomto světle jeví dost špatně navržená, nicméně s tím už nikdo nic nenadělá.

5.2.9 Třída `konfigurace`

V této třídě dochází k sestavování jednotlivých hlaviček. Informace, jež do nich jsou touto třídou vkládány, jsou přebírány z objektu typu `radek`. Zde se zjišťuje, jaké všechny informace uživatel pro rámec/paket definoval a dle toho se sestaví jednotlivé objekty (instance třídy `jednotka_prenosu`).

5.2.10 Třída `defragmentace`

Nese odpovědnost o správný průběh defragmentace jednotlivých příchozích fragmentů paketů. Jsou jí předkládány všechny přijaté fragmenty. Jejím výstupem jsou pakety, jejich všechny fragmenty byly úspěšně přijaty.

5.2.11 Třída `tcp_spojeni`

Třída, jež se stará o jedno vytvořené TCP spojení od jeho započetí až do jeho ukončení. Upravuje obsah paketů určených k odeslání a zároveň analyzuje obsah paketů přijatých z hlediska TCP spojení.

5.2.12 Třída `adapter`

Hlavním údělem této třídy je fragmentace paketů do rámců a jejich odeslání prostřednictvím funcionality nabízené knihovnou pcap.

5.2.13 Třída `rozhrani`

Třída zajišťující výběr síťového rozhraní uživatelem. Mimoto zodpovídá také za zjištění MAC adresy vybraného síťového rozhraní.

Kapitola 6

Testování

6.1 Použité prostředky

6.1.1 Program Wireshark

Funkčnost programu byla ověřována programem Wireshark, jež je volně k dispozici na `http://www.wireshark.org/`.

Program Wireshark umožňuje uživateli sledovat síťový provoz na libovolném síťovém rozhraní. Nutnou podmínkou je však spuštění programu pod identitou, jež má k dané činnosti oprávnění – typicky pod rootem.

Program přehledně vypisuje odesílané a přijímané rámce, které aktuálně tečou přes vybrané síťové rozhraní. Uživatelské rozhraní programu je vyobrazeno na obrázku 6.1.1.

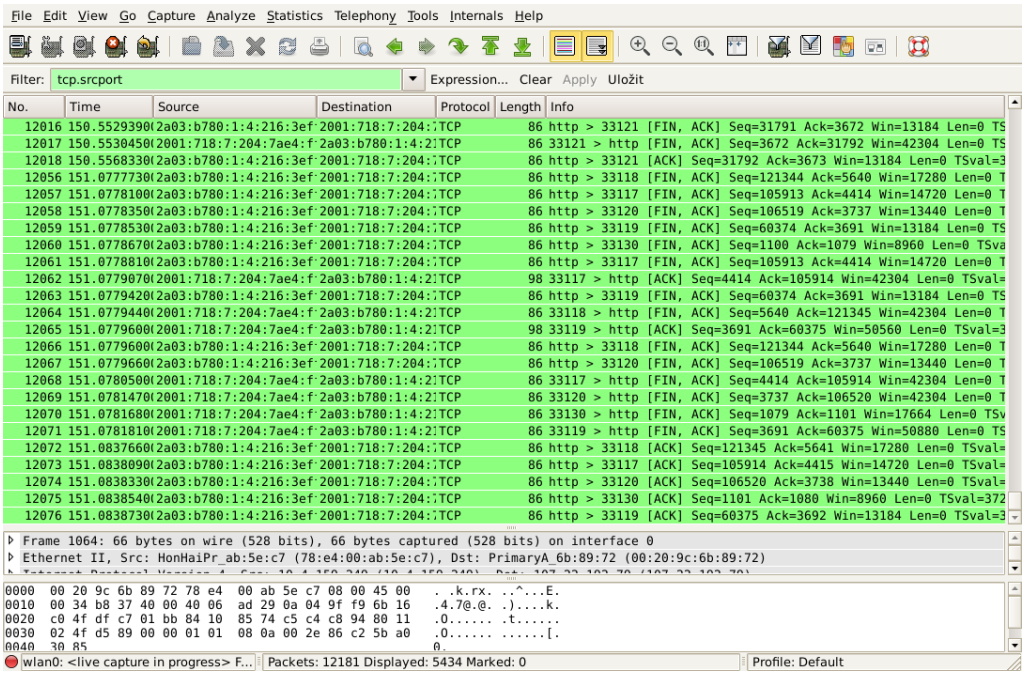
Program umožňuje komunikaci filtrovat dle velmi podrobně nastavitelných pravidel. Informace o jednotlivých rámcích jsou zobrazovány chronologicky dle času jejich příchodu v neustále směrem dolů rozšiřující se tabulce.

Jakýkoliv ze seznamu rámců si lze vybrat a podívat se na jeho obsah detailněji (obrázek 6.1.1).

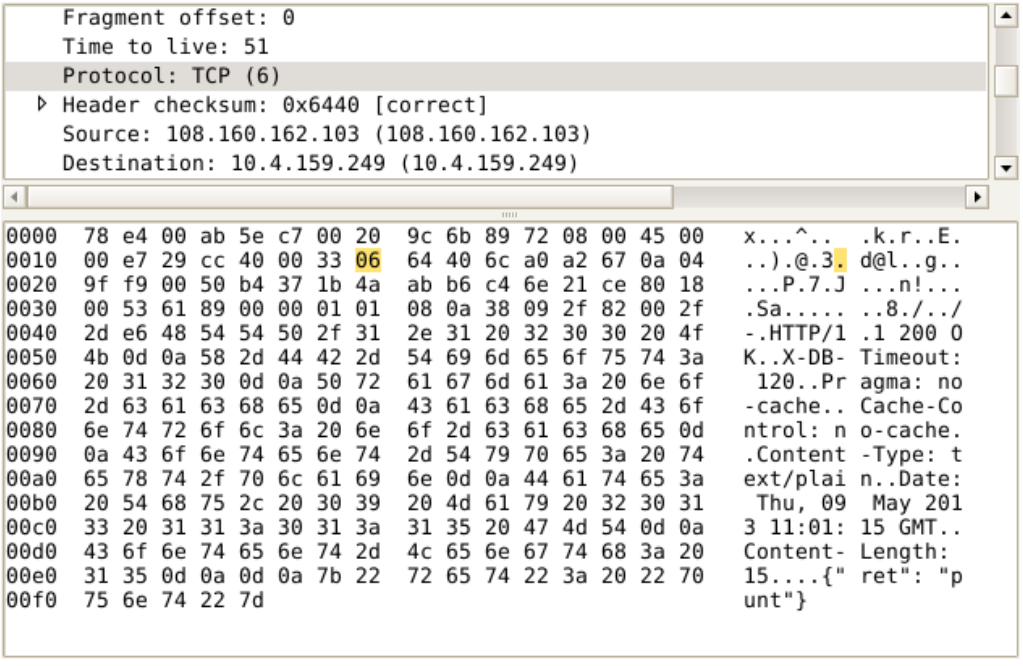
V detailním pohledu je vypsán obsah každého podporovaného protokolu v „lidem čitelné“ podobě. Velmi užitečnou funkcionalitou programu je zvýraznění (viditelné na obrázku 6.1.1) pozice právě označené hlavičky či parametru v ní obsaženém.

6.1.2 Testovací konfigurační soubory

Samotný program Wireshark by na kompletní otestování programu nestačil, respektive stačil, ale jen k otestování přijímací části programu. K tomu,



Obrázek 6.1: Prostředí programu wireshark



Obrázek 6.2: Detail přijatého/odeslaného rámce

abych ověřil, zda-li program také pakety správně generuje, bylo potřeba vytvořit nějaká modelová zadání ve formě konfiguračních souborů.

6.2 Průběh testování

Testování bylo prováděno mezi dvěma stanicemi v lokální síti. Jedna snanice měla spuštěný program v režimu „přijímač“, druhá pak v režimu „generátor“. Pro přehlednost byl použit sekundární konfigurační soubor `/conf/konf2`, v němž byly uloženy často používané hodnoty:

```
MAC       macto=8c:89:a5:2f:9c:cf
MACPROTO  macto=8c:89:a5:2f:9c:cf  eproto=0x800
IPADDR_A  ipfrom=192.168.0.1  ipto=192.168.0.9
IPADDR_B  ipfrom=10.4.0.1     ipto=8.8.8.8
```

6.2.1 Dodržování standardů protokolu Ethernet

6.2.1.1 Testovací konfigurační soubor

```
1 MAC eproto=0     delkadat=46   data=01
2 MAC eproto=0     delkadat=46   data=060708
1 MAC eproto=1

1 MAC eproto=65534 delkadat=1500 data=09
0 MAC eproto=65534 delkadat=1500 data=0a
1 MAC eproto=65535 delkadat=1500 data=0b
```

6.2.1.2 Detekované chyby

Třetí zadání (řádek) definuje příliš krátkou délku Na prvním řádku se programu nelíbila příliš krátká délka (*0*) datové části. Minimální délka ethernetového rámce je *60* bajtů. Jelikož *14* bajtů zabere hlavička rámce, zbývá povinných *46* bajtů délky dat. Ve druhém řádku není definována délka datové části rámce. Programem tuto délku stanoví automaticky na *46*. Její obsah zde taktéž není definován a tak program daných *46* vyplní výchozí hodnotou *0*. Na stanici s MAC `8c:89:a5:2f:9c:cf` rámce jí určené přišly. Jejich data byla neporušena. Test dopadl dobře. Žádné chyby nebyly nalezeny.

6.2.2 Dodržování standardů protokolu IP

6.2.2.1 Testovací konfigurační soubor

```
2 MACPROTO IPADDR_A proto=none delkadat=50    data=00
2 MACPROTO IPADDR_A proto=none delkadat=65515 data=01
1 MACPROTO IPADDR_A proto=none delkadat=15    data=00
1 MACPROTO IPADDR_A proto=none delkadat=1580  data=00
```

6.2.2.2 Detekované chyby

IP pakety, jenž se nevešly do ethernetových rámců byly po své fragmentaci generátorem přijímačem defragmentovány do původní podoby. Délka jejich datové části odpovídala zadání.

6.2.3 Dodržování standardů protokolu ICMP

6.2.3.1 Testovací konfigurační soubor

```
1 d4:ca:6d:21:52:d5 0x800 ipto=8.8.8.8 ipfrom=10.0.0.2 icmp
1 d4:ca:6d:21:52:d5 0x800 ipto=8.8.8.8 ipfrom=10.0.0.2 icmp ttl=3
10 d4:ca:6d:21:52:d5 0x800 ipto=10.0.0.1 ipfrom=10.0.0.2 icmp
```

6.2.3.2 Detekované chyby

Vzdálená strana odpovídá na žádost o odpověď vždy korektně. Program je schopen generovat pouze ICMP typu „žádost o odpověď“. V případě nízkého `ttl` byla dle očekávání přijata zpráva o vyčerpání životnosti ICMP paketu. ICMP žádosti o odpověď byly správně číslovány. Žádné chyby nebyly nalezeny.

6.2.4 Dodržování standardů protokolu UDP

6.2.4.1 Testovací konfigurační soubor

```
1  MACPROTO IPADDR_A udp 1 1 delkadat=2
1  MACPROTO IPADDR_A udp 2 1 delkadat=65507
1  MACPROTO IPADDR_A udp 3 65535
1  MACPROTO IPADDR_A udp 4 65535 delkadat=12
10 MACPROTO IPADDR_A udp 5 1 delkadat=4
```

6.2.4.2 Detekované chyby

Příjem a odesílání UDP paketů probíhal bezproblémově.

6.2.5 Dodržování standardů protokolu TCP

6.2.5.1 Testovací konfigurační soubor

```
1 MACPROTO 192.168.0.1 192.168.0.9 tcp 233 23 delkadat=2
1 MACPROTO 192.168.0.1 192.168.0.9 tcp 2 80 delkadat=65495
1 MACPROTO 192.168.0.1 192.168.0.9 tcp 3 80
1 MACPROTO 192.168.0.1 192.168.0.9 tcp 4 80 delkadat=12
1 MACPROTO 192.168.0.1 192.168.0.9 tcp 5 1
```

6.2.5.2 Detekované chyby

Program není schopen navázat TCP spojení. Tato funkcionalita nebyla implementována.

6.2.6 Testování odolnosti vůči chybnému uživatelskému vstupu

6.2.6.1 Testovací konfigurační soubor

```
1 MACPROTO 192.168.x.1 192.168.0.9 tcp 233 23 delkadat=2
1 MACPROTO 192.168.0.1 192.168.0.9 tcp 2 80 delkadat=-12
1 MACPROTO 192.168.0.1 192.168.0.9 pct 3 80
1 ip 192.168.0.1 192.168.0.9 tcp 4 80 delkadat=12
x MACPROTO 192.168.0.1 192.168.0.9 tcp 5 1
```

6.2.6.2 Detekované chyby

Na každém řádku tohoto uživatelského vsrtupu program úspěšně chyby objevil. Jedinou vadou na kráse je skutečnost, že program vždy vypíše informace o první nalezené chybě a následně zkončí svou činnost. Kdyby před svým koncem vypsal všechny nalezené chyby, vypadalo by to mnohem lépe.

6.3 Závěr vyvozený z testování

Program zvládne zasílat IP, ICMP a UDP pakety. TCP spojení navazovat neumí. Pakliže by protistrana chtěla navazovanou komunikaci akceptovat, operační systém jí ICMP paketem sdělí, že na portu, na nějž se chce připojit, nenaslouchá žádný proces. Aby se navázání spojení zdařilo, musela by aplikace nějakým způsobem sdělit operačnímu systému, že naslouchá na tom a tom portu. To by snad šlo udělat žádostí o přidělení socketu. Nicméně tato funkcionalita již v programu implementována není.

Závěr

Cílem této práce bylo navrhnout a implementovat aplikaci, jež umožní odesílat skrze síťové rozhraní pakety/rámce předem zadaných parametrů. Aplikace měla být přeložitelná a spustitelná jak na OS Linux, tak i na OS Windows. Těhto cílů se podařilo dosáhnout jen zčásti. Aplikace je schopna generovat a přijímat holé ethernetové rámce, holé IP pakety a IP pakety obsahující ICMP či UDP segment. TCP protokol však implementován není.

Měl-li bych vyzdvihnout nějakou informaci, ke které jsem dospěl během psaní práce a o které jsem dříve neměl tušení, potom bych zmínil skutečnost, že po celou dobu psaní práce mi na síťové rozhraní nepřišel ani jediný (!) fragment paketu, který by nebyl mnou samým vytvořen. Fragmentace IP paketů, o které jsem se zprvu domníval, že bude hlavní náplní práce, se tedy v praxi příliš nepoužívá. Podotýkám, že jsem se vždy pohyboval v sítích s $MTU = 1500B$.

Literatura

[1] Branden, R.; Borman, D.; Partridge, C.: *RFC 1071: Computing the Internet Checksum*. ISI, Cray Research, BBN Laboratories, září 1988. Dostupné z: http://tools.ietf.org/rfc/rfc1071.txt

[2] Edward, T.: TCP/IP and OSI Network Model Comparisons. Dostupné z: http://www.dummies.com/how-to/content/network-basics-tcpip-and-osi-network-model-compari.html

[3] Mallory, T.; Kullberg, A.: *RFC 1141: Incremental Updating of the Internet Checksum*. BBN Communications, leden 1990. Dostupné z: http://www.ietf.org/rfc/rfc1141.txt

[4] Postel, J.: *RFC 791: Internet protocol*. University of Southern California / Information Sciences Institute, září 1981. Dostupné z: http://www.ietf.org/rfc/rfc791.txt

[5] Postel, J.: *RFC 792: Internet control message protocol*. ISI, září 1981. Dostupné z: http://www.ietf.org/rfc/rfc792.txt

Příloha A

Seznam použitých zkratek

IP Internet Protocol

ICMP Internet Control Message Protocol

UDP User Datagram Protocol

TCP Transmission Control Protocol

OSI Open Systems Interconnection Basic Reference Model

MAC Media Access Control

DNS Domain Name System

Příloha **B**

Obsah přiloženého CD

```
/ ..... kořenový adresář CD
├── doc ..... dokumentace generovaná ze zdrojových kódů
│   ├── html ..... formát html
│   └── latex ..... formát LaTeX
├── exe ..... spustitelná verze implementace
│   ├── program ..... OS Linux 32bit
│   └── program.exe ..... OS Windows 32bit
├── src
│   ├── impl ..... zdrojové kódy implementace
│   └── thesis ..... zdrojová forma práce ve formátu LaTeX
├── text
│   ├── thesis.pdf ..... text práce ve formátu PDF
│   └── thesis.ps ..... text práce ve formátu PS
└── readme.txt ..... popis obsahu CD
```

Příloha C

Uživatelská příručka

C.1 Kompilace

C.1.1 Linux

1. Program ke svému chodu vyžaduje nainstalovanou knihovnu `libpcap`, kterou je možno stáhnout z `http://www.tcpdump.org/` nebo nainstalovat prostřednictvím balíčkovacího systému vaší distribuce. Například příkazem `aptitude install libpcap0.8-dev` v linuxové distribuci Debian.

2. Dále je potřeba zkopírovat obsah adresáře `/src/code/project1` z přiloženého CD na pevný disk.

3. V místě, kam jste obsah adresáře `/src` nahráli spusťte kompilaci programu příkazem `make`. V adresáři vznikne spustitelný soubor `program`.

C.1.2 Windows

1. Program ke svému chodu vyžaduje nainstalovanou knihovnu `winpcap`, kterou je možno stáhnout z `http://www.winpcap.org`.

2. Dále je potřeba zkopírovat obsah adresáře `/src/code` z přiloženého CD na pevný disk.

3. Jeden ze souborů, co takto zkopírujete (`projekt.sln`) lze otevřít v Visual Studiu.

4. Po otevření ve Visual Studiu můžete program kompilovat.

C.2 Spuštění

Buďto si zkompilujete program ze zdrojových souborů sami dle návodu C.1, nebo využijete již zkompilovaných verzí programu připravených k použití. Ty se nacházejí v adresáři `/exe`. Více informací o nich se dovíte v příloze B nebo ze souboru `readme.txt`, jež se nachází v kořenovém adresáři přiloženého CD.
Program je možné spustit z příkazového řádku. Možností má uživatel hned celou řadu:

```
./program
./program nazev_souboru
./program nazev_souboru nazev_souboru
./program identifikator_rozhrani
./program identifikator_rozhrani nazev_souboru
./program identifikator_rozhrani nazev_souboru nazev_souboru
```

C.2.1 Volba režimu

Program může běžet buďto jako generátor ethenetových rámců nebo jako jejich přijímač. To, v jakém režimu uživatel chce, aby program běžel se určí podle parametrů, které uživatel uvede při spuštění programu. Pakliže uživatel neuvede jediný parametr, který by reprezentoval cestu k souboru ve filesystému, bude program spuštěn v režimu „přijímač“. Naopak, pokud uživatel alespoň jeden parametr obsahující cestu k souboru uvede, bude program spuštěn v režimu „generátor“.

C.2.2 Volba rozhraní

Programu je nutné nějakým způsobem předat identifikátor rozhraní, nad kterým uživatel chce, aby program běžel. Jako identifikátor rozhraní lze zadat název rozhraní, což na OS Linux bývá typicky `eth0` či `wlan0`. Na OS Windows je to však podstatně složitější řetězec typu

```
\Device\Tcpip_{EF8BD219-1D00-4EE2-A755-DD343B646785}
```

, což není zrovna komfortní pro použití. Proto lze rozhraní specifikovat taktéž jeho MAC adresou s dvojtečkovou notací – např. `4A:22:CF:02:DA:37`. Poslední možností je specifikovat chtěné rozhraní aliasem, jež je tvořen obyčejným číslem v dekadickém zápisu. Dané číslo je však k rozhraní přiřazeno až za běhu programu a není zcela zaručeno, že bude při opětovném spuštění programu stejné. Hodí se však pro interaktivní výběr rozhraní, k čemuž se tímto dostávám.

C.2.2.1 Parametrem předaným skrze příkazovou řádku

Uživatel si vybere jeden ze tří výše uvedených způsobů identifikace rozhraní a předá ji programu formou parametru. Tento parametr se uvádí vždy jako první.

C.2.2.2 Interaktivně až za samotného běhu programu

Pakliže uživatel nezadá identifikaci rozhraní formou parametru, program hned po svém startu vypíše seznam dostupných síťových rozhraní a vyzve uživatele k zadání identifikace požadovaného rozhraní. Výzva se bude cyklicky opakovat, dokud uživatel zadá identifikátor, jenž bude shledán platným.

C.3 Ukončení

Program se mimo výzvy k zadání identifikace síťového rozhraní uživatele nikdy na nic neptá a tedy nikdy nečeká na uživatelský vstup. Vždy končí buďto přirozenou smrtí nebo je jeho činnost ukončena uživatelem.

C.3.1 Přirozená smrt

Nastane v případě, když je program spuštěn jako „generátor“. Program odešlě všechny zadané rámce/pakety a skončí.

C.3.2 Vynucená smrt

Je potřeba v případě, když program běží v režimu „přijímač“, kdy přijímá a vypisuje síťový provoz. Pakliže uživatel chce program ukončit a neběží-li mu na pozadí, zvládne tak pomocí klávesové kombinace `[ctrl]+[c]`. Pakliže program běží na pozadí, bude nutné využít programů `ps` a `kill`.

C.4 Konfigurace

Veškerá konfigurace pro vygenerování požadovaných rámců/paketů se píše do konfiguračního souboru, jehož adresa se programu předává formou parametru při spuštění. Jak jste si ale možná všimli u výčtu možných volání programu výše, je možné definovat hned dva konfigurační soubory – hlavní a vedlejší. Chce-li uživatel použít vedlejšího konfiguračního souboru, předá jeho název vždy jako ten úplně poslední parametr.

C.4.1 Hlavní konfigurační soubor

Každý řádek souboru představuje konfiguraci jednoho rámce/paketu, jenž se může libovolněktát opakovat. Každý řádek může obsahovat množinu parametrů, jimiž se nastaví vlastnosti rámce/paketu. Parametry se oddělují mezerou. Řádky, které obsahují pouze mezery či odřádkování, jsou ignorovány. Stejně tak jsou ignorovány i řádky začínající znakem #, jež jsou považovány za komentář. Konfigurační soubor může vypadat například takto:

```
parametr parametr parametr parametr
#parametr parametr parametr

parametr parametr
parametr #parametr
#
```

Druhý, třetí a poslední řádek budou v daném případě při zpracování konfiguračního souboru ignorovány. Zbylé řádky se budou považovat každý za konfiguraci rámce/paketu, jehož případné opakování se nastavuje vždy prvním parametrem na řádce.

C.4.2 Parametry

Parametry program rozlišuje na povinné a nepovinné. Povinné parametry jsou nezbytně nutné pro správné sestavení rámce/paketu. Při jejich absenci program ukončí svou práci vypsáním chybové hlášky s informací, jaký parametr chybí. Povinné parametry bude program brát z řádky popořadě, tak jak jsou obsaženy v jednotlivých hlavičkách.

```
13 aa:aa:aa:aa:aa:aa 0 77
```

Program tento zápis pochopí následovně: Uživatel chce `13` rámců/paketů adresovaných na MAC adresu `aa:aa:aa:aa:aa:aa` s protokolem vyšší vrstvy nastaveným na hodnotu `0`, čili má jít o pouhý ethernetový rámec. Data vložená do tohoto rámce mají mít délku `77`. Všechny uvedené parametry jsou povinné a program je na řádce rozliší podle jejich pořadí. Pořadí většího množství parametrů je obtížné si zapamatovat, proto je možné parametrům přidávat prefix s jejich názvem. Názvy jednotlivých parametrů jsou stanoveny v souboru `volby.hpp` pomocí direktiv `#define`. Parametry s prefixem nemusí dodržovat pořadí.

```
aa:aa:aa:aa:aa:aa pocet=13 77 proto=0
```

Tento zápis zafunguje zcela stejně. Zde již nezáleží na pořadí parametrů s výjimkou parametru `aa:aa:aa:aa:aa:aa` a `77`, jež stále musí dodržovat pořadí.

C.4.2.1 Nepovinné parametry

Některé parametry program výslovně nevyžaduje a dokáže se obejít i bez nich. Příkladem může být MAC adresa odesílatele. Není-li uvedena, program použije MAC adresu použitého rozhraní. Chce-li uživatel nepovinný parametr uvést, musí tak učinit vždy včetně jeho prefixu. Z toho vyplývá, že u nepovinných parametrů nikdy nezáleží na jejich pořadí, protože jejich součástí je vždy prefox.

```
13 aa:aa:aa:aa:aa:aa 0 macfrom=54:D8:7A:52:FE:FA
```

C.4.3 Vedlejší konfigurační soubor

Není povinný a pro běh programu není v některých případech vůbec potřeba. Definují se v něm „proměnné“, které pak lze použít v hlavním konfiguračním souboru. Uživateli se v hlavním konfiguračním souboru mohou opakovat parametry se stejnou hodnotou na více řádcích zároveň. Pokud by uživatel chtěl tyto parametry hromadně měnit nebo prostě jen méně psát, může si ve vedlejším konfiguračním souboru nadefinovat proměnné, které pak může v hlavním konfiguračním souboru používat místo plného zápisu parametrů. Například:

```
mac1 45:14:12:74:A1:A8
mac2 macfrom=AA:BB:CC:DD:EE:FF
```

Napíše-li uživatel do hlavního konfiguračního souboru řetězec `mac2`, bude tento řetězec nahrazen řetězcem `macfrom=AA:BB:CC:DD:EE:FF`. Analogicky bude nahrazen i jakýkoliv výskyt řetězce `mac1`.

C.4.4 Jak program vyhodnocuje konfigurační soubor

Program vyhodnocuje sekvenčně – řádek po řádku – obsah celého konfiguračního souboru. Vždy ho zajímá jen jeden řádek. Jeho kontext – řádky v jeho okolí – vůbec nevnímá. Projdeme si teď postupně, jak program postupuje. Jednotkou pro nás bude jeden řádek hlavního konfiguračního souboru. V rámci konfiguračního souboru se tento postup počtřádků-krát opakuje.

C.4.4.1 Je daný řádek něčí konfigurací

Ze všeho nejdříve je potřeba zkontrolovat, má-li cenu se řádkem vůbec zabývat. Řádky obsahující pouze bílé znaky nebo začínající znakem # dále již nejsou zpracovávány a program je ignoruje.

C.4.4.2 Začíná se rámcem

Program začne sestavovat požadovanou jednotku od nejnižších vrstev. Nejmenší jednotkou, jakou je program schopen vygenerovat je `rámec`. Program začne na řádku hledat parametry povinné pro tvorbu rámce. Pokud nějaký z povinných parametrů neuvedete, program zahlásí chybu a skončí. Následně program analyzuje obsah parametru obsahující identifikátor protokolu vyšší vrstvy. Nalezne-li hodnotu, jenž bude vyjadřovat protokol IP, přeskočí rovnou na kapitolu C.4.4.5. Jinak bude pokračovat kapitolou C.4.4.3

C.4.4.3 Pokračuje paketem

Nenajde-li program povinné parametry pro sestavení IP paketu, skončí společně s výpisem chyby. Jinak zanalyzuje obsah parametru, jenž by měl udávat protokol vyšší vrstvy. Bude-li obsahovat „programem podporovanou hodnotu“, přejde se ke stavbě segmentu transportní vrstvy respektive ke konstrukci ICMP paketu C.4.4.4. V opačném případě se program ubere na C.4.4.5.

C.4.4.4 Následuje segment

Nenajde-li program povinné parametry pro sestavení UDP/TCP segmentu či hlavičky ICMP, skončí společně s výpisem chyby. Jinak se přejde k poslednímu možnému bodu C.4.4.5.

C.4.4.5 Doplnění daty

V této části program opět skončí s výpisem chybové hlášky, nenajde-li povinné parametry. Pakliže je najde, zkontroluje, zda-li je taková délka dat vyšší vrstvy v daném útvaru (rámci/paketu/segmentu) vůbec přípustná. Pakliže ano, sestavování je hotovo.

C.4.5 Slovník konfiguračních voleb

Bylo by nanejvýš vhodné uvést případnému uživateli aplikace seznam všech parametrů, jež lze při nastavování jedné skupiny rámců/paketů uplatnit a

tím ovlivnit jejich výslednou podobu. Jedna skupina rámců/paketů je nastavována prostřednictvím jednoho řádku v konfiguračním souboru. Seznam všech možných parametrů je uveden v tabulce C.1.

Kategorie	Nastavovaná položka	Identifikátor	Povinný	Validní hodnoty
obecné	počet objektů	pocet	ano	číselný zápis
rámec	MAC příjemce	macto	ano	MAC adresa zapsaná dvojtečkovou notací
	MAC odesílatele	macfrom	ne	MAC adresa zapsaná dvojtečkovou notací
	protokol vyšší vrstvy	eproto	ano	{číselný zápis, "ip", "IP"}
paket	time to live	ttl	ne	číselný zápis
	protokol „vyšší“ vrstvy	proto	ano	{"icmp", "tcp", "udp", "none"}
	IP odesílatele	ipfrom	ano	IP adresa zapsaná tečkovou notací
	IP příjemce	ipto	ano	IP adresa zapsaná tečkovou notací
segment	zdrojový port	sport	ano	číselný zápis
	cílový port	dport	ano	číselný zápis
data	délka dat	delkadat	ne	číselný zápis
	obsah dat	data	ne	číselný zápis

Tabulka C.1: Parametry, jimiž je možné ovlivňovat generované objekty