Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Diplomová práce

Softwarový firewall pro filtrování na síťové a linkové vrstvě

Plzeň, 2013 Bc. Martin Petrák

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne Bc. Martin Petrák

Abstract

Recently, information technology has penetrated into many branches of human life. We can found computers built-in public transport vehicles, production halls or offices, of course. Every company is using for its business information technology and computers networks today. Years ago, we could find computers networks only in big enterprises. Now computers networks have been expanded into small and medium businesses and households too.

Big amount of information is flowing daily in a company computer network. Some of them are unimportant, but some of them are important or business critical. This critical information should stay in computer network. Nobody wants leaving information out to the internet. There can by a situation, when unknown attacker will try to get into the company network from internet, because he wants steal company data or attack inner computer systems.

In this work, application called *Software firewall for network and data link layer filtering* was developed. This application will be used on network border computer and will filtering network data flow. Some of them will allow going through, some data should be denied. Filter depends on filtering rules, which will be set up by administrator of network. Application can filter on network layer and on data link layer too. It is designed to keep administration of filtering rules clear and easy.

Obsah

1 Úvod

V poslední době se informační technologie šíří do různých odvětví. S počítači se dnes setkáme takřka v každém dopravním prostředku, výrobní hale, nebo kanceláři. Každý podnikatel nebo firma využívají ke svému podnikání informační technologie. Nejedná se ale pouze o konzervované systémy. Velmi často se používají systémy skládající se z různých částí, které mohou být od sebe vzdálené až několik tisíc kilometrů. V každém podniku tak jistě nalezneme počítače spojené do počítačové sítě. Počítačová síť není záležitostí jen velkých firem, ale vlivem rozvoje IT se dostala i do středních a menších podniků či domácností.

Ve firemní počítačové síti proudí denně mnoho dat. Může se jednat o data naprosto nedůležitá, například když nějaký zaměstnanec vyhledává informace na internetu, ale také o data důležitá až kritická. Taková kritická data by měla zůstat pouze ve firemní síti a neměla by se dostat ven mimo ni.

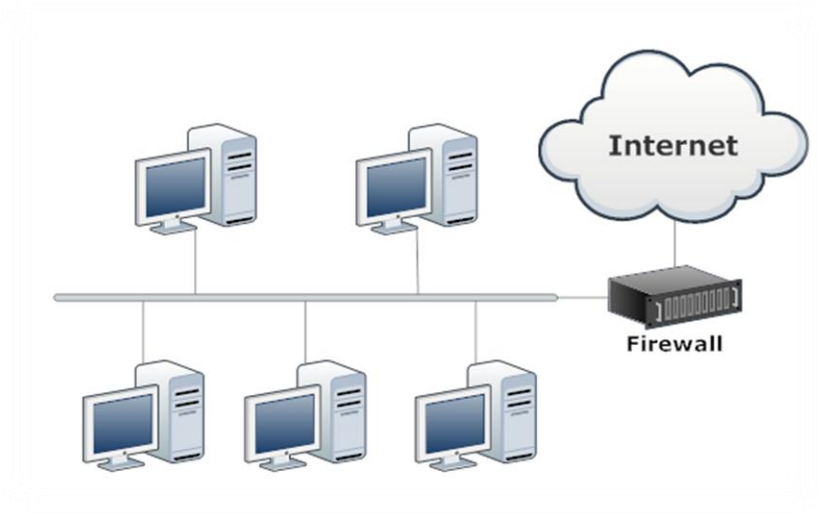
Kromě úniku dat může také dojít k situaci, kdy se neznámý útočník bude pokoušet dostat zvenku firemní sítě dovnitř. Úmyslem takového útoku může být krádež vnitřních firemních dat nebo útok na systémy uvnitř firmy. Ve všech případech se jedná o úmysl nějakým způsobem uškodit podniku, firmě či osobě, která síť provozuje.

Cílem diplomové práce je navrhnout a vytvořit aplikaci, která má být nasazena na nějakém hraničním bodu počítačové sítě a která bude administrátorům sítě pomáhat k zamezení nechtěných útoků na systémy v síti nebo naopak zamezovat nechtěným únikům dat ze sítě do internetu. Aplikace má sloužit jako filtr síťového provozu a propouštět nebo nepropouštět data podle filtrovacích pravidel, která administrátor sítě do aplikace zadá. Oproti běžným filtrům však aplikace umí filtrovat síťový provoz jak na síťové vrstvě modelu ISO/OSI, tak i na vrstvě linkové. Vše je navrženo tak, aby administrace filtračních pravidel byla srozumitelná, přehledná a jednoduchá.

2 Filtrování síťového provozu

2.1 Síťový firewall obecně

Firewall je velmi efektivní typ zabezpečení počítačové sítě. Původně se toto označení používalo pouze ve stavebnictví, kde anglické slovo *firewall* označovalo zeď nebo jakoukoliv překážku, která měla za úkol zabránit rozšíření požáru z jedné části budovy do druhé [1]. Dnes se stejné označení používá pro systémy, které mají shodně jako ve stavebnictví zabraňovat rozšíření nebezpečí. V tomto případě se zabraňuje průchodu z Internetu do privátních počítačových sítí, které jsou k Internetu připojeny. A právě v bodě (případně bodech) připojení sítí do celosvětového Internetu se síťové firewally nacházejí, jak ukazuje obrázek 2.1.



Obrázek 2.1 Umístění síťového firewallu

Veškerá síťová komunikace mezi počítači umístěnými ve vnitřní soukromé síti a síťovým uzlem v Internetu tak probíhá přes toto místo, což dává firewallu příležitost právě zde rozhodnout, jestli je komunikace *přijatelná* [1].

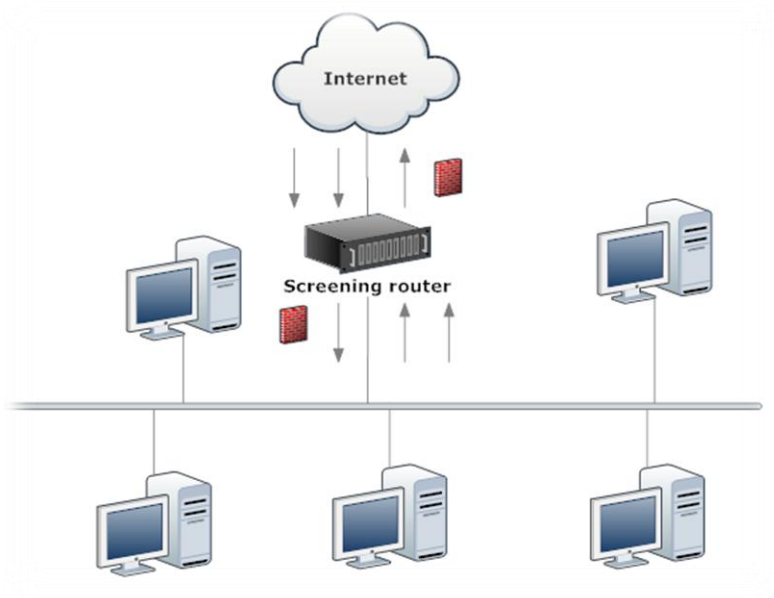
O přijatelnosti komunikace rozhoduje firewall na základě jeho konfigurace, kterou někdo při jeho instalaci provedl či průběžně aktualizoval. Přijatelnost znamená, že komunikace je v souladu s bezpečnostní politikou dané sítě [1], přičemž každá síť má určitě jinou bezpečnostní politiku.

Síťový firewall může tvořit mnoho různých prvků, ať už se jedná o prvky hardwarové nebo softwarové. Můžeme se ale také setkat s řešením některých výrobců zařízení, kteří prodávají jeden box, ve kterém nalezneme všechny důležité komponenty. Jedná se tak o univerzální řešení, které zároveň s ochranou sítě poskytuje například služby směrovače, DNS serveru a mnoho dalších méně či více užitečných vlastností v jednom. Často také do firewallu zahrnujeme samotné připojení k Internetu. Tím se firewall stává součástí sítě a nejedná se o něco, co lze jednoduše vypojit či zahodit [1].

Vše výše uvedené zní, jakoby firewall byl něco úžasného, něco magického, co když zabudujeme do sítě, stane se tak tato síť naprosto bezpečná a nic nám nehrozí. Toto však pravda není. V síti zabudovaný firewall ještě neznamená chráněnou síť. Útočníci jsou chytří a zajisté dokážou jeden takový firewall, byť velmi dobře nakonfigurovaný, obejít. V mnoha případech samotní administrátoři nechávají útočníkům nechtěně otevřená takzvaná *zadní vrátka* (často se jedná a špatnou konfiguraci firewallu). Proto se doporučuje použití firewallu kombinovat s dalšími bezpečnostními prvky uvnitř sítě, mezi něž patří například RMON sondy, které promiskuitně monitorují provoz na síti a dokážou zareagovat na atypický provoz sítě, nebo i správné nastavení uživatelských hesel a oprávnění. Všeobecně tak mluvíme o bezpečnostní politice. Použití firewallu vyžaduje promyšlenou bezpečnostní strategii, nelze pouze bezhlavě umisťovat firewally do různých částí sítě. Stále se však jedná o nejefektivnější věc, jak připojit privátní počítačovou síť do Internetu a zároveň tuto síť chránit [1].

2.2 Paketový filtr

Paketový filtr je jeden z případů firewallu. Standardní paketový filtr směruje pakety mezi uzly ve vnitřní síti a uzly v Internetu. Na základě bezpečnostní politiky sítě pak jednotlivé pakety buďto zpracuje (provede jejich směrování) nebo je zahodí. Takovému systému říkáme *screening router*, jenž je znázorněn na obrázku 2.2.



Obrázek 2.2 Screening router

Oproti běžnému směrovači má tento paketový filtr více práce. Běžný směrovač po přijetí paketu pouze zjistí cílovou adresu, kam má paket namířeno, a odešle ho na rozhraní, které bylo vybráno jako nejlepší možná cesta pro daný paket. Případně si směrovač neví s paketem rady (neví, kam ho poslat). V tomto případě paket zahodí a pouze odešle na zdrojovou adresu paketu oznámení. Jako oznámení slouží většinou zpráva ICMP protokolu „destination host unreachable“.

Paketový filtr typu screening router rozebere celou hlavičku paketu a až když zjistí, zda takovýto paket může dál zpracovávat, začne se rozhodovat o tom, kam daný paket nasměruje.

Paketové filtry ale nemusí operovat na síti jako směrovače. Obecně lze filtrování paketů zařadit v takových síťových zařízeních, kudy pakety probíhají a kde lze implementovat politiku pro zpracování či zahození paketů. Mezi taková zařízení se zařazují i *mosty* či *přepínače*, které dokážou pakety filtrovat. V angličtině tak hovoříme o zařízeních typu *packet filtering bridges* [1]. Jejich princip je velmi podobný principu screening router. Pouze se jedná o zařízení, která pracují na nižší vrstvě síťového modelu (viz kapitola 3). Velmi často se taková zařízení v praxi nepoužívají, jelikož se jedná o zařízení, která je nutno do sítě zabudovat zvlášť a „navíc“, oproti filtrům, které se dají využít zároveň jako směrovače. Přesto ale mají tato zařízení jednu nespornou výhodu pro filtrování paketů. Paketové filtry, které pracují v režimech směrovačů lze totiž velmi snadno na síti odhalit díky samotnému procesu směrování, zatímco mosty a přepínače tuto vlastnost nemají, jelikož samotný proces zpracování paketu jdoucího skrz takové zařízení nijak do struktury paketu nezasahuje a nemodifikuje jej. Přitom poskytují stejné vlastnosti jako filtry v režimech směrovačů [1].

2.2.1 Proces filtrace

Filtrace síťového provozu probíhá na základě informací v hlavičkách jednotlivých paketů, konkrétní popis a možnosti filtrování jsou uvedeny v kapitole 4. Kromě toho může takovýto směrovač zpracovávat mimo hlaviček i samotná data a filtrovat pakety na základě nějaké detailnější informace [1]. Dále lze pakety jdoucí skrze směrovač validovat, jestli mají správnou velikost či nejsou nijak úmyslně či neúmyslně poškozeny.

Kromě samotných informací obsažených v paketu může filtr brát v úvahu také port, na kterém paket do zařízení přišel a port, na kterém by měl paket ze zařízení zase odejít.

Na základě definovaných pravidel může paketový filtr udělat následující akce.

- poslat paket dál směrem k cíli,
- zahodit paket, aniž by odesílatele o tomto činu informoval,
- nepřeposlat paket dále, ale odesílatele o tomto činu informovat,
- zapsat si informace o paketu do vlastního logu,
- informovat někoho (správce sítě), že se při filtraci objevil nějaký konkrétní paket,
- poslat paket jinému cíli, než pro koho byl původně určen,
- pozměnit filtrovací pravidla na základě obdrženého paketu.

Poslední dvě možnosti používají již více sofistikované zařízení. V případě změny pravidel se toto používá například pro dynamické povolení odpovědi nebo zakázaní veškeré komunikace ze sítí, která tento paket poslala [1].

Vhodnou kombinací filtrovaných pravidel můžeme tedy například kompletně zakázat příchozí komunikace z nějaké vzdálené sítě nebo povolit jen jistý typ komunikace (email, www) a ostatní zakázat. Dále se může jednat o blokaci odchozí

komunikace z privátní sítě do nějaké jiné či opět povolení pouze určitých služeb, které z pohledu bezpečnostní politiky chápeme jako služby bezpečné. Zde je již pole působnosti volné a možnosti konfigurace zcela otevřené.

2.2.2 Stavový paketový filtr

Stavovými paketovými filtry nazýváme systémy, které si při filtraci uchovávají informaci o průbězích komunikace. V tomto případě jsou pak taková zařízení schopna povolit například příchozí komunikaci z nějaké sítě v Internetu pouze v případě, že tato komunikace byla navázána z privátní vnitřní sítě. Protože se jedná o filtraci, která závisí na již zachycené komunikaci, a tato filtrace se v čase mění, nazýváme tyto systémy *dynamickými paketovými filtry*.

Pokud navíc systém dokáže filtrovat na základě obsahu paketů a ne pouze jejich hlaviček (viz kapitola 4), často mluvíme o *inteligentních paketových filtrech*.

V praxi většina filtrovacích systémů umí nahlížet do datových částí paketů a také je modifikovat, proto všechny tyto systémy označujeme jako stavové paketové filtry, ačkoliv existují i takové, které i přes toto označení s datovými částmi paketů nepracují [1].

2.3 Výhody a nevýhody filtrování paketů

2.3.1 Výhody

Filtrování paketů je velmi efektivním bezpečnostním prvkem pro ochranu počítačové sítě. Je k dispozici v mnoha zařízeních, především směrovačích. Může se jednat o hardwarová řešení, ale i řešení softwarová. Na trhu lze nalézt komerční produkty, ale také produkty, které lze používat zcela zdarma.

Běžný proces filtrace paketů (na základě hlaviček paketů) není navíc nijak obtížný a nepředstavuje pro zařízení velkou zátěž. Paketové filtry tak neomezují síťový provoz a uživatelé o nich prakticky vlastně neví. Samozřejmě, že pokud po paketovém filtru požadujeme složitější operace, jako je například stavové filtrování, bude zařízení potřebovat více zdrojů, bude docházet k delšímu zpoždění přenosu samotných dat a poté už nejde o výhodu, nýbrž spíše o nevýhodu paketových filtrů.

Asi nejdůležitější a klíčovou výhodou paketových filtrů je jejich efektivita, kdy správně strategicky umístěný paketový filtr může ochránit celou privátní síť připojenu do Internetu. Přičemž pokud je síť připojena do Internetu pouze přes jeden směrovač, právě zde probíhá filtrace a už víceméně nezáleží na samotné velikosti privátní počítačové sítě [1].

2.3.2 Nevýhody

Paketový filtr nemusí přesně odpovídat bezpečnostní firemní politice. Jeho konfigurace tak, aby byla politika splněna stoprocentně, nemusí být možná. Paketové filtry ve většině případů pracují na síťové a transportní vrstvě protokolu ISO/OSI či TCP/IP (popsáno dále v kapitole 3). Firemní bezpečnostní politika ale může stanovit zákaz používání některých aplikací, což je záležitost aplikační vrstvy. Sice jsou dány standardní porty na transportní vrstvě, které se používají

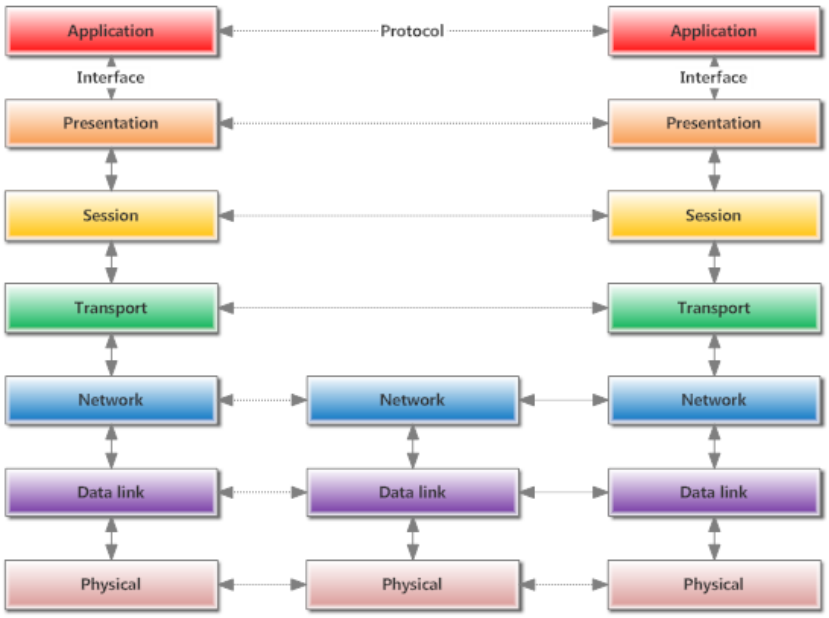
výhradně pro jim určené aplikace. Není to však pravidlo, nýbrž používaný standard. To znamená, že takovou politiku běžný paketový filtr nedokáže splnit a musí se použít jiné, složitější zařízení či úplně jiný typ firewallu.

V případě, že je paketový filtr součástí nějakého směrovače, znamená to pro zařízení další zátěž. Zde vždy záleží na konkrétním zařízení, implementaci a výrobci, ale všeobecně platí, že čím více je filtrace složitější, tím více se zařízení zatěžuje a zpomaluje datový tok. Jako příklad můžeme citovat zdroj [1], který se odvolává na funkcionalitu Cisco zařízení „fastpath“. V normálním případě tato funkce provádí základní směrování a využívá pouze procesoru, který se nachází na kartě příslušného portu. Nezatěžuje hlavní procesor zařízení. Pokud ale je ale použito filtrování paketů o určitých vlastnostech, pak toto filtrování provádí hlavní procesor pro každý paket ze všech portů, což evidentně pakety zdržuje. V tomto případě se jednalo pouze o některé verze zařízení a obsaženého software.

Paketové filtry nejsou úplně dokonalé. U mnoha produktů se můžeme setkat s obtížnou konfigurací filtrovacích pravidel, což samozřejmě může způsobit celkovou špatnou konfiguraci a špatné chování filtru. Také možnosti filtrování mohou být v některém produktu omezené. To je dáno tím, že vytvoření velmi univerzálního paketového filtru, který by pokryl všechny typy sítí a protokolů, je velmi těžké, prakticky nemožné [1].

3 Síťový model ISO/OSI

Síťový model ISO/OSI představuje referenční komunikační model. Jeho název je tvořen zkratkou ze slov „*International Standards Organization*“ a „*Open System Interconnection*“, což v českém překladu znamená název *Mezinárodní organizace pro normalizaci* a *propojení otevřených systémů*. Model je znázorněn na obrázku 3.1.



Obrázek 3.1 Referenční model ISO/OSI

Tento model byl vytvořen již v roce 1983 [2]. Dnes se používá jako názorné řešení počítačové komunikace v počítačových sítích. Jedná se o model, který je složen z vrstev (vrstevnatý), přičemž jednotlivé vrstvy jsou na sobě nezávislé a lze je nahradit.

Model obsahuje celkem sedm vrstev. Každá z vrstev vykonává jinou funkci. Pro svou činnost využívá vrstva *služeb* sousední nižší vrstvy. Sama pak poskytuje službu vrstvě vyšší.

Prostředky pro komunikaci sousedních vrstev mezi sebou nazýváme *rozhraním*. Pokud však mluvíme o komunikaci stejných vrstev mezi sebou, kdy je každá z vrstev na jiném uzlu počítačové sítě, pak je prostředkem komunikace takzvaný *protokol*. Pod protokolem si můžeme představit sadu příkazů a hodnot, jejichž povolenou kombinací definujeme informace pro tutéž vrstvu v jiném uzlu.

3.1 Fyzická vrstva

Anglicky *physical layer*.

Fyzická vrstva definuje fyzickou komunikaci přenosovým médiem. Přenosovým médiem může být například optický kabel, vodič nebo elektromagnetická vlna a další. Fyzická vrstva tak specifikuje mechanické, elektrické a funkční prostředky

k aktivaci, udržení a ukončení fyzického spojení pro přenos bitů mezi dvěma uzly [3]. Fyzickým spojením pak rozumíme přenos bitů, který se vztahuje k jedné relaci komunikace.

3.2 Linková (spojová) vrstva

Anglicky *data link layer*.

Linková vrstva poskytuje funkční prostředky pro jednotlivé síťové prvky k zajištění navázání, udržení a ukončení linkového spojení.

Spojení na linkové vrstvě je vytvořeno na základě jednoho nebo více fyzických spojení (spojení na fyzické vrstvě) [3].

Linková vrstva detekuje a případně také opravuje chyby, které vzniknou při přenosu fyzickou vrstvou.

Vrstva shromažďuje data do logických jednotek nazvaných *rámce* [3].

Na této vrstvě pracují síťová zařízení, která nazýváme *mosty* a *přepínače*. Most slouží k propojení dvou částí sítě, které používají zpravidla jiná přenosová média. Přepínače mohou také propojovat různá přenosová média do jedné sítě, primárně ale mají více portů a spojují tak především více částí sítě (stejné přenosové médium). Přepínač tak přijme na jednom z portů data a přepošle je dál na předem známý nebo vybraný port. Existuje mnoho technik a algoritmů, podle kterých přepínač data přeposílá. Vždy záleží na konkrétním zařízení a jeho konfiguraci.

Jak již bylo napsáno, linková vrstva slouží k přenosu mezi sousedícími uzly. Aby bylo možné uzly rozpoznat a například přepínač se mohl rozhodnout, na jaký port jaká data přeposlat, používají se zde *hardwarové adresy*. V případě implementace - *Ethernet* se jedná o *MAC adresy.* Běžně, ačkoliv z hlediska pojmenování vrstev v modelu ISO/OSI nesprávně, se tyto adresy označují jako *síťové*. Adresy jsou neměnné, z velké části dané přímo výrobcem konkrétního zařízení tak, aby každá z adres na síti byla jedinečná.

3.3 Síťová vrstva

Anglicky *network layer*.

Síťová vrstva poskytuje funkční prostředky pro spojový i nespojový přenos mezi prvky transportní vrstvy. Pro transportní prvky se přitom jedná o přenos dat, který nezávisí na směrování dat a udržování spojení [3].

Vrstva zajišťuje navázání, udržení a ukončení síťového spojení mezi systémy, které obsahují komunikaci aplikačních prvků [3]. Jedná se tedy o systémy, na kterých běží nějaké aplikace, které používají síťovou komunikaci. Mimo to vrstva zajišťuje i samotný přenos dat skrze vytvořené síťové spojení.

Přenos dat je nezávislý na jejich směrování. Ve skutečnosti tak vyšší sousední transportní vrstvu nezajímá, kde se dva uzly nachází a kudy se k nim data dostanou. Toto právě implementuje vrstva síťová.

Síťová vrstva shromažďuje data do logických jednotek nazvaných *pakety*.

Na této vrstvě pracují zejména zařízení, kterým říkáme *směrovače*. Tyto zařízení propojují různé podsítě a spojují je do jedné velké počítačové sítě. Uzly na této vrstvě používají takzvanou *IP adresu*. Její název je odvozen z nejpoužívanějšího protokolu této vrstvy. Adresa není oproti adrese na linkové vrstvě pevně daná výrobcem zařízení, ale standardně existují mechanizmy, jak se adresa uzlu přiděluje. Pakety pak obsahují IP adresu zdrojového a cílového uzlu. Směrovač na základě adresy, kam mají být data doručena, rozhodne, na jaký port paket přepošle. Opět dnes existuje více algoritmů a jejich implementací, podle kterých se směrovač rozhodne. Obecně se jedná o směrovací algoritmy. Tyto algoritmy však nejsou předmětem této práce a proto zde rozebírány a popisovány nebudou. Pro účel nám pouze postačí skutečnost, že něco takového na síťové vrstvě probíhá.

3.4 Další vrstvy modelu ISO/OSI

Transportní vrstva (anglicky *transport layer*) zajišťuje přenos dat mezi dvěma uzly. Jak již bylo napsáno výše, tato vrstva neřeší samotné směrování dat. Vrstva vyšším vrstvám poskytuje samotný přenos, který se dělí na služby *spojované* (TCP protokol) a *nespojované* (UDP protokol).

Relační vrstva (anglicky *session layer*) řídí celkovou komunikaci neboli relaci. Synchronizuje dialog dvou uzlů a spravuje výměnu dat.

Prezentační vrstva (anglicky *presentation layer*) poskytuje jednotnou prezentaci informace, která se přenáší mezi uzly aplikační vrstvy [3]. Zajišťuje tedy jednotný a nezávislý pohled na přenášená data.

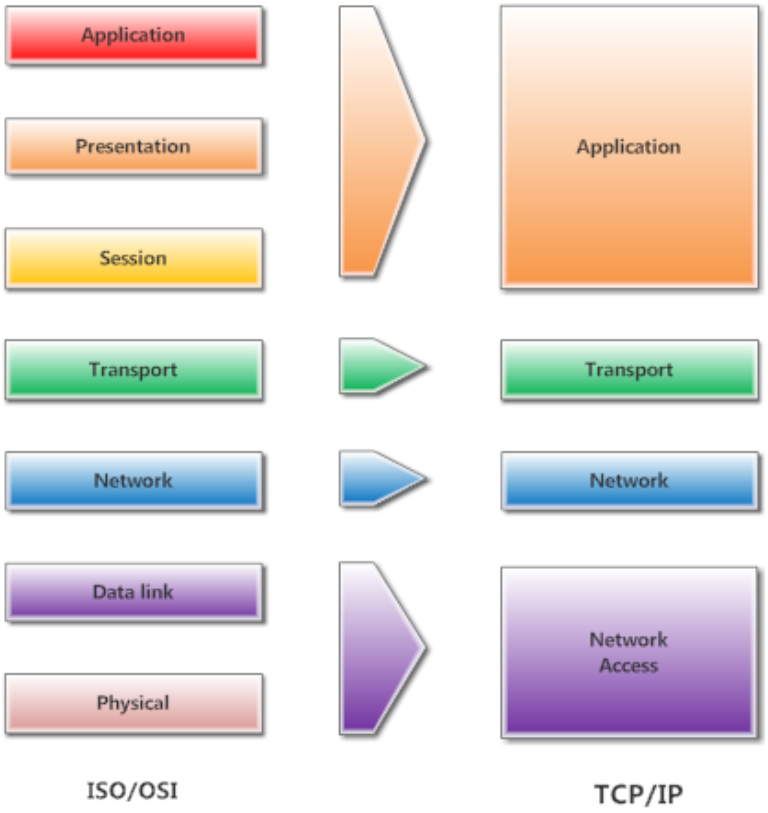
Aplikační vrstva (anglicky *application layer*) je nejvyšší vrstvou modelu ISO/OSI a poskytuje různým procesům a aplikacím schopnost komunikace [3].

3.5 Použití modelu v počítačových sítích

ISO/OSI model nebyl nikdy přesně realizován. Podle modelu je vždy možná komunikace pouze s vrstvou nad nebo pod a všechny vrstvy musí být obsaženy. To v řadě případů přináší komplikace v podobě zbytečné zátěže či složitosti protokolového zásobníku. Přesto je tento model vhodný pro teoretický popis počítačových sítí.

V současné době se v celosvětové počítačové sítí internet používá rodina protokolů z názvem ***TCP/IP***. Název je složen ze dvou nejdůležitějších obsažených protokolů, tedy protokolu *TCP* nacházejícího se v transportní vrstvě modelu, který se používá v případě spojové komunikace a přenosu dat, a protokolu *IP*. Ten se nachází v síťové vrstvě (v TCP/IP modelu se o této vrstvě mluví jako o *internetové*) a zajišťuje služby této vrstvy, viz kapitola 3.3.

Podobně jako ISO/OSI model, se i TCP/IP skládá z několika vrstev. Oproti sedmi zde však nalezneme pouze vrstvy čtyři. Vše je dáno agregací některých vrstev z modelu ISO/OSI do jedné vrstvy modelu TCP/IP, jak je znázorněno na obrázku 3.2. Jednotlivé vlastnosti vrstev se však nemění, došlo pouze vlivem agregace vrstev k jejich spojení.



Obrázek 3.2 Porovnání modelů ISO/OSI a TCP/IP

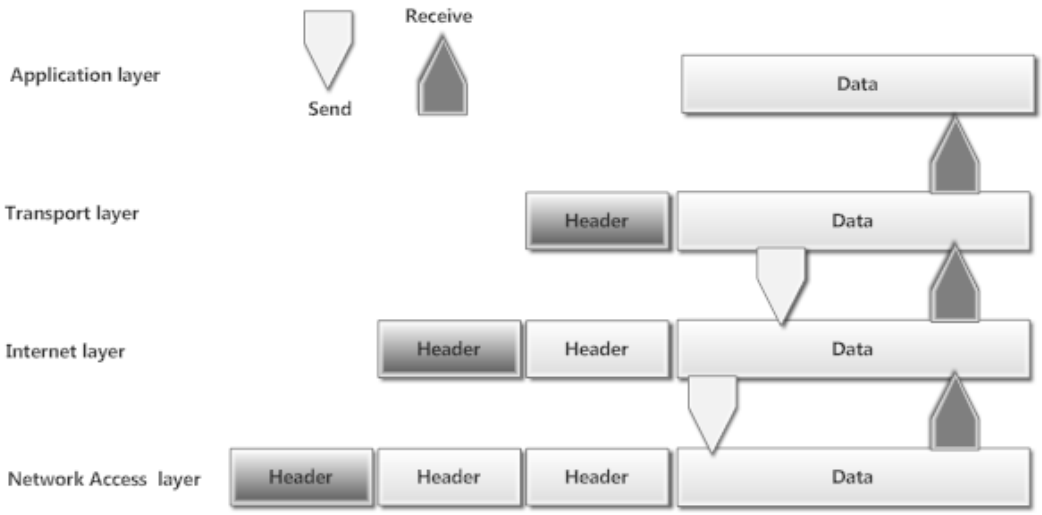
V případě síťové vrstvy zde není žádný problém, jelikož tato vrstva i ve skutečnosti existuje. U linkové vrstvy je situace jiná. Vrstva je ve skutečnosti spojena s fyzickou a tvoří vrstvu síťového rozhraní, viz obrázek 3.2.

4 Typy protokolů a jejich možnosti filtrace

4.1 Přenos informací počítačovou sítí

Pokud přenášíme nějakou informaci skrze počítačovou síť, velmi často se informace rozděluje na menší části, z nichž každá je pak odeslána samostatně. V případě dnes velmi používaných IP sítí se jednotlivé části dat nazývají pakety [1].

Pakety jsou vytvářeny postupně, jak data prochází zásobníkem protokolů TCP/IP. Na každé vrstvě zásobníku má paket dvě části. První část nazýváme *hlavičkou*. Hlavička obsahuje informace ohledně protokolu, který je použit na dané vrstvě při konkrétním přenosu dat. Druhou část paketu, *tělo*, tvoří přenášená data. Data se pak velmi často tvořena celým paketem protokolu vyšší vrstvy v zásobníku [1]. Situaci si můžeme snadno představit tak, že vrstva dostane paket protokolu, jenž byl vybrán na sousední vyšší vrstvě, z tohoto paketu udělá tělo svého paketu a přidá informace svého protokolu jako hlavičku. Výsledný paket pak předá sousední nižší vrstvě. Na aplikační vrstvě tvoří paket pouze data, která se budou přenášet. Tento proces nazýváme *zapouzdřením*, je znázorněn na obrázku 4.1.



Obrázek 4.1 Zapouzdření dat při přenosu

Na druhém konci počítačové sítě se postupuje analogicky v opačném pořadí. Nižší vrstva nejdříve zpracuje data určená svým protokolem a poté posílá vyšší sousední vrstvě již „rozbalené“ tělo paketu tvořené paketem použitého protokolu na vyšší vrstvě.

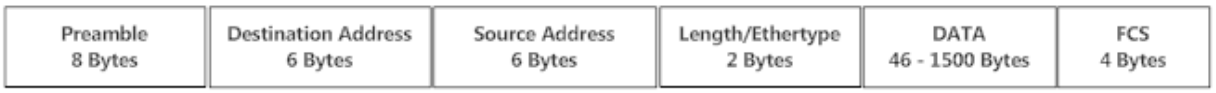
Z hlediska filtrování síťového provozu se nejdůležitější informace nachází v každé z hlaviček různých vrstev [1].

4.2 Možnosti a protokoly linkové vrstvy

Z hlediska zásobníku protokolů TCP/IP tuto vrstvu označujeme také jako vrstvu síťového rozhraní (viz 3.5).

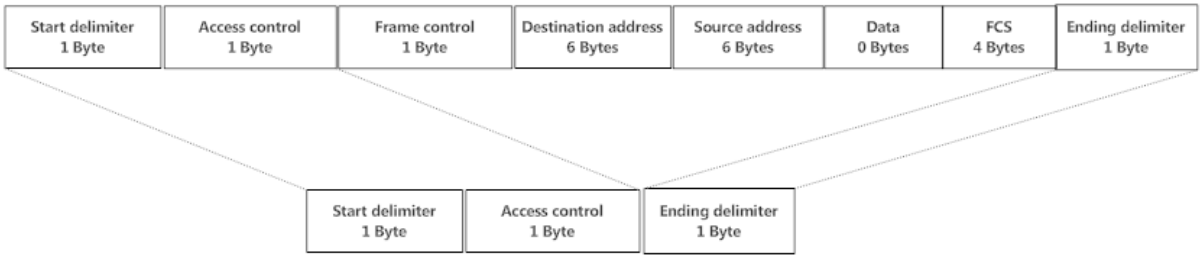
Největším a nejrozšířenějším zástupcem této vrstvy je v současné době *Ethernet*. Patří sem ale další typy sítí a protokoly, jako jsou Token Ring, FDDI, Frame Relay a další.

Z pohledu filtrování paketů (na této vrstvě se označují logické datové jednotky správně rámce, ale pro srozumitelnost a lepší pochopení budeme v souvislosti s filtrováním síťového provozu používat výraz paket, který pro nás tak znamená základní datovou jednotku, kterou lze filtrovat) není až tak výhodné používat k filtraci právě informace z linkové vrstvy, přestože je to teoreticky možné. Problémem v tomto případě je, že všechny příchozí pakety do privátní sítě z Internetu budou obsahovat stejné hardwarové adresy. Cílovou adresu bude představovat právě paketový filtr, nejčastěji směrovač, přes který je síť připojena do Internetu. Dále je tu skutečnost, že mnohdy se sítě připojují do Internetu přes několik rozhraní (záložní spojení apod.), z nichž každé připojení může používat různé typy protokolů. To představuje nutnost často konfigurovat filtrační pravidla pro každé připojení extra. Nelze tak napsat jedno pravidlo, které by šlo aplikovat na všechny rozhraní směrovače, který má dvě připojení typu Ethernet a dvě připojení typu FDDI, protože hlavičky Ethernetu a FDDI nejsou stejné, jak je znázorněno na obrázcích 4.2 a 4.3.



Obrázek 4.2 Ethernet rámec

Obrázek 4.3 FDDI rámec



Obrázek 4.4 Token Ring rámec

Můžeme se všimnout, že všechny zde uvedené typy protokolů používají stejnou velikost hardwarových adres, a také, že Token Ring (obr. 4.4) používá velmi podobnou strukturu, jako FDDI.

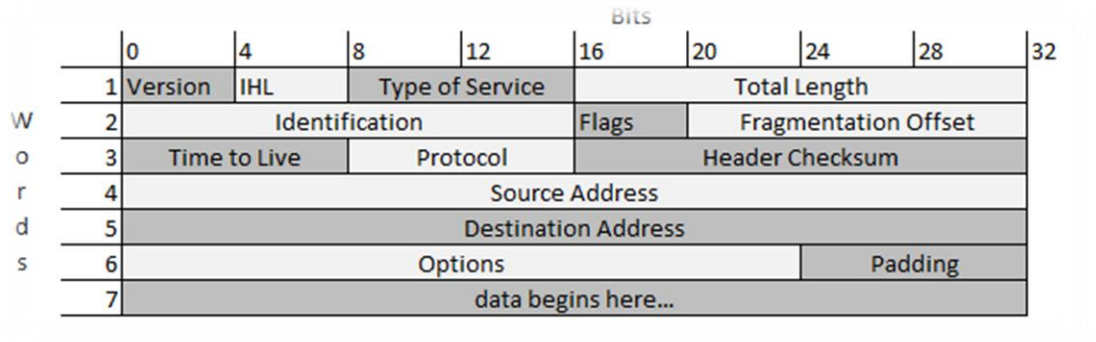
Filtrování na linkové vrstvě ale nemá pouze jen nevýhody a určitě pro něj použití existuje. Pokud například privátní síť představuje menší a omezenou množinu počítačů. Můžeme povolit komunikaci ze sítě právě a pouze těmto počítačům. Nemůže se pak stát, že se v síti objeví nové zařízení, které by mohlo komunikovat s někým v Internetu. Nové zařízení má totiž jinou hardwarovou adresu (viz kapitola 3.2), která je pro filtr neznámá a ten odchozí komunikaci nepustí. Zde je rozdíl oproti síťové vrstvě a IP adresám, které jsou „znovupoužitelné" v zařízeních. Samozřejmě i změnu hardwarové adresy lze dnes udělat a útočníci mají k dispozici mechanismy k tomuto úkonu. Už se ale jedná pro útočníka o další ztížení.

4.2.1 Shrnutí linkové vrstvy

Shrnuto jedinou možností filtrace na linkové vrstvě jsou tedy hardwarové adresy, které však mají význam pouze v lokální síti. V případě Ethernetu pak můžeme přidat pole *TYPE/LENGTH*. Jelikož samotný Ethernet má několik implementací, obsah hlavičky v tomto poli se může lišit. V původní verzi protokolu (IEE 802.3) toto pole udávalo délku celého rámce / paketu. V moderních používaných verzích (Ethernet II) však toto pole představuje číselný kód o velikosti dvou oktetů, který udává typ protokolu vyšší vrstvy, který je do rámce zapouzdřen. To může být další možnost či kritérium, podle čeho lze pakety filtrovat.

4.3 IP protokol síťové vrstvy

Nejrozšířenějším protokolem síťové vrstvy je protokol *IP* (*Internet Protocol*). Samotný Internet je definován jako síť složená z IP podsítí [1]. Jak můžeme vidět na obrázku 4.5, IP paket na síťové vrstvě nabízí podstatně více možností k filtraci oproti protokolům linkové vrstvy. Mezi ty nejpodstatnější určitě patří zdrojová a cílová adresa paketu, typ zapouzdřeného protokolu, verze IP protokolu, fragmentační příznak nebo IP možnosti paketu (popsáno dále v kapitole 4.3.3).



Obrázek 4.5 IPv4 paket

4.3.1 Adresy a jejich typy

IP adresy představují hlavní kritérium, pokud mluvíme o filtrování paketů. Paketový filtr tak může povolit či zakázat komunikaci na, anebo z přesně určených

adres. Ovšem většinou se filtrovací pravidla nekonfigurují pro samostatné adresy, ale používá se celých rozsahů adres. To znamená, že lze specifikovat komunikaci s určitou jinou sítí počítačů a zařízení, které patří do jiné sítě v Internetu. Tyto zařízení mají stejný prefix IP adresy a lze je tak zařadit ve filtru do jedné skupiny. Obdobným způsobem lze také rozdělit počítače a zařízení ve vnitřní síti na určité podmnožiny, jež lze identifikovat stejným prefixem IP adres. Toto je věc správců sítě a nastavené politiky, přičemž důmyslné promyšlení strategie a konceptu sítě evidentně pomáhá i k zajištění lepší bezpečnosti.

IP protokol dnes existuje ve dvou hlavních verzí – verze 4 a 6. Každá z verzí používá jiný formát adres. V této podkapitole, pokud nebude určeno jinak, vždy bereme v úvahu starší verzi protokolu, verzi *IPv4*. Změny, které přineslo zavedení novější verze s označením *IPv6* si popíšeme ve zvláštní kapitole později.

Kromě samotných adres dále rozlišujeme také jejich typy. Pokud je paket určen pouze jednomu příjemci, nazýváme ho *unicast*. Toto je také nejčastější případ. Kromě toho můžeme paket označit za *multicast*, což představuje paket, který je určen pro nějakou skupinu cílů, která byla tvůrcem paketu vybrána jako skupina, pro kterou mohou být informace v paketu důležité. Multicast pakety představují hlavně lepší efektivitu síťové komunikace. Pokud bychom chtěli poslat informaci určitým uzlům sítě, rozeslali bychom tolik unicast paketů, kolik je uzlů (pro každý uzel jeden paket). Když ovšem použijeme multicast paket, do kterého zadáme předem danou a správnou multicast IP adresu, pošleme ve skutečnosti jeden paket, který bude doručen do všech cílových zařízení.

Dalším typem je *broadcast*. Jedná se o podobný princip jako v případě multicast paketů, avšak broadcast je doručen všem. Záleží na konkrétním uzlu, zda informaci vyhodnotí jako důležitou a zpracuje ji nebo zda ji zahodí a bude ignorovat.

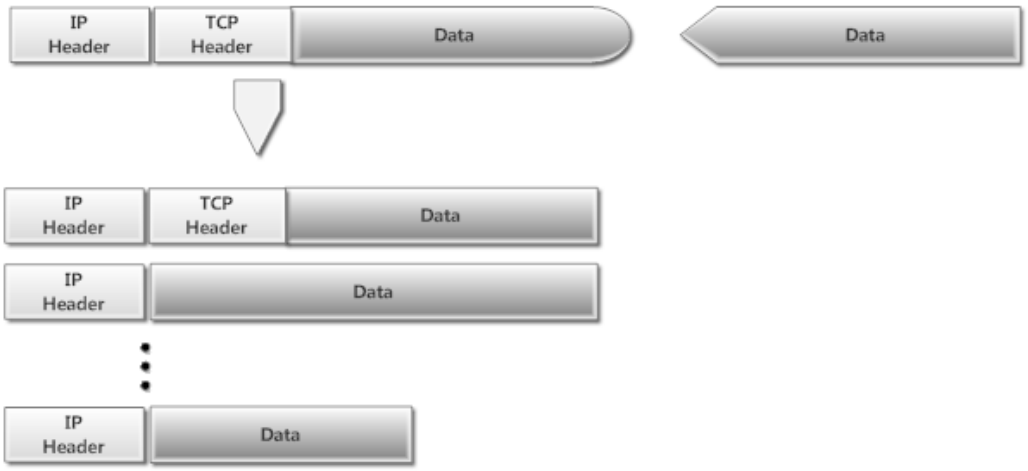
Adresy typu multicast a broadcast se používají jako adresy cílů, nikoliv však jako zdrojové adresy. Zařízení může použít broadcast adresu pouze v několika výjimečných případech, když například nemá k dispozici správnou IP adresu a snaží se o její získání (princip DHCP) [1]. Pokud se ale vyskytne v síti paket se zdrojovou adresou typu multicast nebo broadcast, velmi často to signalizuje pokus o útok. Útočník zadá zdrojovou adresu jako multicast a cílovou adresu zařízení v naší síti. Toto zařízení pak ve skutečnosti využije jako zdroj, který přijme tento „podvrhnutý“ paket a zcela správně se na něj bude snažit odpovědět, čili vytvoří paket, kde cílovou adresou bude právě broadcast a ten tak bude doručen všem okolním strojům. Útočníci velmi často využívají právě tohoto triku k rozšíření se na další okolní zařízení [1].

Na druhou stranu broadcast pakety z vnitřní sítě mohou (ale nemusí) obsahovat důvěrné informace a jejich šíření z vnitřní sítě dál do Internetu také není žádoucí. Dalším problémem může být jejich veliké množství, které síť zahltí. Při zahlcení pak síť není schopna normálního provozu.

I když se dnes mnoho sítí propojuje skrze Internet ve větší sítě, stále je dobré pečlivě nakonfigurovat, do jaké sítě broadcast pakety projít mohou a kam se naopak dostat nesmí.

4.3.2 Fragmentace

IP sítě mají jednu vlastnost, díky níž dokážou přenést velké pakety i přes takové spoje, které nedokážou tak velké pakety přenést (z hlediska různých limitů). Jednoduše velký paket rozdělí na menší části, které se přenášejí sítí samostatně a které pak příjemce spojí zpět v jeden velký paket. Tuto vlastnost nazýváme *fragmentací*. Pokud je paket při přenosu fragmentován, zůstává fragmentovaný až k cílovému zařízení [1].



Obrázek 4.6 Fragmentace IP paketů

Jak znázorňuje obrázek 4.6, fragmentace může pro paketový filtr představovat určitý problém. Přestože všechny fragmenty musí a obsahují IP hlavičku, pokud bude paketový filtr pracovat i s vyšší vrstvou, ta bude obsažena pouze v prvním fragmentu. V ostatních chybí. Běžné paketové filtry toto řeší tím způsobem, že filtrují opravdu pouze první fragment paketu, který obsahuje hlavičku protokolu vyšší vrstvy, a ostatním fragmentům povolují projít skrz. Pokud tedy první fragment paketu neprojde skrze filtr, jeho další fragmenty ano. Cílové zařízení však stejně celkový paket nesestaví, jelikož první fragment chybí, a neúplný paket tak nepřijme. Toto chování bylo označeno za bezpečné [1].

Přesto zde ale problém přetrvává. Pokud totiž přes paketový filtr projdou všechny fragmenty kromě prvního, cílové zařízení si jednotlivé přijaté fragmenty nějakou dobu drží v paměti a čeká, zda chybějící fragment nedorazí (mohlo dojít k nějakému zpoždění). Toto chování nahrává útočníkům k DOS útokům (útoky, jejichž cílem je vyřadit cíl útoku z provozu). Pokud v cílovém zařízení vyprší doba, po kterou se čeká na opožděný fragment, zařízení vygeneruje ICMP zprávu ve které informuje zdroj, že vypršel čas pro sestavení fragmentu. To je opět chování, díky kterému se útočník dozví, že cíl existuje.

Navíc může útočník využít fragmentace k dalším typům útoků. Často jsou podvrženy falešné fragmenty, které obsahují schválně špatně rozdělená data. Například část dat z konce prvního fragmentu se opakuje na začátku fragmentu druhého apod. Cílové zařízení při zpětném skládání dat počítá s tím, že data z druhého fragmentu pokračují přesně tam, kde data v prvním fragmentu skončila.

Podvržením neprávně fragmentovaných paketů tak může dojít k tomu, že cíl si nebude při skládání dat vědět rady, proces nějakým způsobem zkolabuje a tím může zkolabovat celý operační systém v zařízení. Tím útočník dokáže ochromit část počítačové infrastruktury.

V opačném komunikačním směru zase může útočník použít fragmenty k tomu, aby v nich dokázal dostat vnitřní data za firewall do Internetu.

Existují i takové paketové filtry, které dokážou fragmentované pakety spojit, zkontrolovat a poté poslat dál, případně opět fragmentovat. Taková řešení spolehlivě ochrání síť nejlépe před nástrahami a útoky spojenými s fragmentací IP paketů. Na druhou stranu ale celý proces více zatěžuje zdroje zařízení a zpožďuje síťovou komunikaci. Jako dalším řešení tak může být zakázání průstupu všem fragmentům kromě toho prvního. To naopak může zase vést k tomu, že některá spojení se nebudou navazovat. Zde se jedná o rozhodnutí vlastníka nebo správce dané sítě, co je pro něj výhodnější a co lépe naplňuje bezpečnostní politiku. Nutno však dodat, že fragmentovaných paketů v síti ubývá úměrně s používáním MTU algoritmu pro zjištění maximální velikosti paketu, aniž by musel být při přenosu fragmentován.

4.3.3 IP možnosti

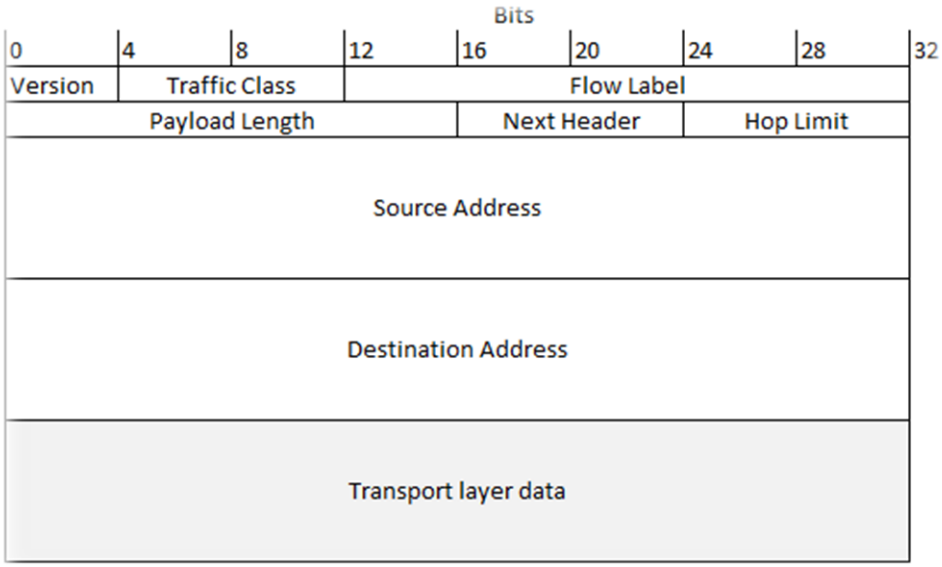
Jak uvádí obrázek 4.5, v hlavičce IP paketu nalezneme pole možnosti. Většinou se toto pole v paketech nepoužívá a zůstává prázdné. Jeho smyslem bylo vytvořit při vývoji protokolu pole, kam lze v budoucnu zapsat důležité informace, které však nemohou být uvedeny v hlavičce jinde.

Jedním z takových případů je i možnost nastavit paketu přesnou cestu k cíli. Tím lze obejít směrování paketů na každém směrovači. Směrovač tak paket přepošle přesně podle hodnot v poli, nebude používat svých směrovacích informací. Toto je vhodné zejména vyskytne-li se někde v síti chyba ve směrování. Zdroj pak může paketům určit přesnou cestu, kudy mají pakety „obejít“ chybný bod. V praxi se ale tato možnost zneužívá útočníky, kteří se tak snaží obejít různé směrovače a firewally a dostat se k cíli útoku.

Některé paketové filtry nabízí možnosti filtrovat na základě různých nastavených hodnot v poli IP možností. Některé zase shrnují IP možnosti jako nastavené nebo nenastavené a podle toho se rozhodují, což ve výsledku nepředstavuje žádné problémy.

4.3.4 IPv6

S rozvíjejícím se internetem postupně během času narůstala spotřeba IP adres. Ačkoliv byly vymyšleny techniky typu NAT apod., adresní prostor IP protokolu verze 4 je téměř celý využit. To byl asi největší impuls ke vzniku IP protokolu verze 6 (IPv6). Protokol IPv6 tedy přišel se zásadní změnou v podobě adres, jejichž pole se zvýšilo z 32 bitů na 128 a tím pádem se dosáhlo většího adresového prostoru.



Obrázek 4.7 IPv6 paket

Jak je vidět z obrázku 4.7, změny v nové verzi IP protokolu se nedotkly pouze adres. Celkově byla velikost hlavičky stanovena na pevnou délku 40B. Díky tomu v hlavičce již nenalezneme pole udávající její délku, dále pak chybí IP možnosti, příznaky, fragmentace a kontrolní součet. Naopak zde zůstal typ IP protokolu, tzv. „hop limit“, typ další hlavičky (protokolu) a síťová a zdrojová adresa. Přibyl pak identifikátor přenosu.

IPv6 používá techniku zřetězených hlaviček, což v praxi znamená, že za IPv6 hlavičku může být další hlavička například s informacemi ohledně šifrování nebo ověřování (autentifikace). Tohoto mechanismu se také používá v případě vnoření IPv4 v IPv6. Samozřejmě pod další hlavičkou se bude asi nejčastěji skrývat hlavička použitého protokolu na transportní vrstvě.

Důležitou informací je, že IPv6 nepodporuje fragmentaci paketů ve směrovačích (větší pakety zahazuje a posílá zpět ICMPv6 zprávu o zahození) a počítá s tím, že odesílatel nejprve použije MTU algoritmus (více o algoritmu je možné se dočíst například ve článku na serveru Root.cz: http://www.root.cz/clanky/velke-trable-s-malym-mtu/) ke zjištění největší možné velikosti paketu. Odesílatel může fragmentaci použít. Použije pak zřetězené hlavičky, kam zadá informace o fragmentu.

Z pohledu paketového filtru však v hlavičce stále nalezneme ty nejdůležitější možnosti filtrace, jež jsou bezesporu IP adresy (zde v jiném formátu) a typ přenášených služeb (protokol) vyšší vrstvy, stejně jako tomu je v případě verze IPv4.

4.4 Ostatní protokoly síťové vrstvy

Ačkoliv je IP protokol jasným reprezentantem síťové vrstvy protokolového zásobníku a modelu TCP/IP, není jediným protokolem této vrstvy. Pro příklad si

můžeme uvést protokoly *AppleTalk* nebo *IPX*. Pro srovnání s protokolem IP i ostatní protokoly poskytují stejný typ služeb. Pouze hlavičky protokolů se liší.

Filtrování jiných protokolů síťové vrstvy, než je IP, není příliš rozšířeno. Důvodem je především skutečnost architektury Internetu, který byl definován jako síť složená z IP podsítí [1]. Často se tedy při přenosu ostatních protokolů skrze Internet používá zapouzdření těchto protokolů do IP (IP pakety v sobě nesou další protokol síťové vrstvy). Přesto, pokud je použit nějaký protokol přímo, bez zapouzdření, ve většině případů se jedná o použití v konkrétním malém místě – soukromé podsíti. Účelem paketových filtrů jakožto typu firewallu je ale chránit podsítě. I z tohoto důvodu jsou paketové filtry orientovány především na IP protokoly a možnosti filtrace ostatních protokolů síťové vrstvy jsou omezené.

Pokud je paketový filtr označován, jako filtr podporující i jiné síťové protokoly, často se jedná pouze o rozpoznání, že jde právě o jiný protokol než IP a konfigurace nabízí tyto pakety zahodit nebo povolit. Nalezneme také filtry, které se zaměřují na konkrétní protokol a dokážou hlavičku tohoto protokolu „rozebrat“, nicméně takové filtry se používají v málo případech a jedná se o speciální systémy.

5 Dostupná API pro filtrování na platformě Linux

Projektů, které se zabývají filtrováním síťového provozu, nalezneme poměrně mnoho. Téměř vždy se jedná o softwarové balíky či frameworky, které se dostávají do jádra operačních systémů a poskytují tak v systému služby spojené s filtrováním paketů (nejenom filtrování paketů, ale také další služby spojené se síťovým provozem). Některé z nich jsou součástí operačních systémů standardně.

Ne každý projekt je však určen pro platformu Linux a ne každý poskytuje takové služby, které od něj potřebujeme. Jedná se především o schopnost filtrovat již na linkové vrstvě modelu ISO/OSI. Většina projektů se soustředí především na operační systémy platformy UNIX, jako FreeBSD apod. Některé jsou pro tyto platformy určené primárně, ale obsahují i „klony“ pro platformu Linux a další.

Dále si probereme některá z dostupných řešení, která lze na platformě Linux použít.

5.1 IP Filter

IP Filter je softwarový balíček, který lze nainstalovat do operačního systému založenému na typu *UNIX* (*UNIX-like*) jako modul jádra spouštěný za běhu. Také lze modul přidat do jádra přímo jako jeho součást. Modul pak poskytuje v operačním systému služby pro filtrování síťové komunikace, případně i překlad síťových adres (NAT).

IP Filter je standardně součástí operačních systémů FreeBSD, NetBSD, Solaris, OpenSolaris, lze ho však také použít s jádrem operačního systému Linux. Nicméně právě na Linuxu se příliš nepoužívá. Na stránkách produktu [4] je pak uvedeno, že IP Filter byl na operačním systému Linux testován pouze v distribucích RedHat 9.0 a SuSE 9.1 a obecně jej lze použít s jádry ve verzích 2.4 a 2.6.

Lze tak konstatovat, že IP Filter nemá na platformě Linux příliš silné zastoupení.

5.2 Linux IP Firewalling Chains

Linux IP Firewalling Chains, neboli *ipchains*, vznikl jako přepis podobného projektu s názvem *ipfwadm*, který byl určen pro jádra operačních systémů BSD. Jeho vznik způsobila potřeba nějakým způsobem kontrolovat a nastavovat paketový filtr v jádře operačního systému Linux. Do jádra byl tento projekt zařazen od verze 2.1.102 [5].

Ipchains dokáže filtrovat síťový provoz na základě informací z hlaviček síťové a transportní vrstvy. Neposkytuje však možnosti pro filtraci na vrstvě linkové.

5.3 IP firewall administration

IP firewall administration, zkráceně *ipfwadm*, je vlastně předchůdce ipchains [5], jenž je popsaný v předešlé kapitole. Jedná se o utilitu, která dokáže spravovat služby firewallu jádra operačního systému Linux [6].

Ipfwadm lze použít s jádry Linuxu ve verzích 1.2 a 2.0. S novějšími verzemi se doporučuje přejít právě na ipchains [6].

Ani ipfwadm nedokáže filtrovat síťový provoz na základě hodnot získaných z hlaviček linkové vrstvy.

5.4 Netfilter

Netfilter je projekt, který v současné době nahrazuje předchozí dvě řešení ipfwadm a ipchains. Jedná se o framework, jenž je z jeho větší části implementován v samotném jádře operačního systému Linux. Standardní součástí jádra je od verze 2.4 výše.

Netfilter bývá velmi často spojován s jeho hlavní částí – iptables. Jedná se o program a stejnojmennou komponentu celého frameworku, které jsou velmi často používány za účelem filtrace paketů na operačních systémech typu Linux. Na frameworku jeho vývojáři stále pracují a zdokonalují jej ve všech oblastech / komponentách, které nabízí.

Vlastní filtrace paketů probíhá na základě přípojných míst v kódu jádra, ke kterým lze tak, jak paket kódem prochází, připojit filtrovací pravidla. V těchto místech jsou pak pakety s pravidly porovnávány a na jejich základě je paket poslán v kódu jádra dále nebo je zahozen.

Díky projektu ebtables lze Netfilter použít pro filtrování již na linkové vrstvě, i když zde jsou nějaká omezení. Ta si společně s detailnějším představením celého frameworku popíšeme v samostatné kapitole dále (kapitola 6).

5.5 Berkeley Packet Filter

Berkeley Packet Filter (BPF) vznikl původně pro jádra UNIX, lze jej také ale použít na jádrech Linux. To umožňuje projekt *netsniff-ng* [7], který obsahuje komponentu *bpfc*. Právě ta dokáže zkompilovat a použít kód BPF v jádře Linuxu.

Filtrování BPF je realizováno pseudokódem. Jedná se o kód, který připomíná assembler. Kód pak znamená postupné procházení hlaviček odchyceného paketu a porovnávání hodnot v nich zachycených s hodnotami specifikovanými v kódu. Na konci průchodu je verdikt ano / ne, neboli propustit paket dále či ho zahodit. [8]

Jelikož BPF ve svém filtračním pseudokódu pracuje s paketem ve formátu, v jakém byl přijat síťovým rozhraním, lze tímto způsobem analyzovat i hlavičku protokolu na linkové vrstvě.

BPF se používá především pro filtraci příchozích paketů do systému za účelem jejich monitorování či analytického zpracování, kdy se do monitoringu či analýzy zařazují pouze některé pakety. Jako příklad můžeme uvést program *Wireshark*, jenž slouží právě k zachycování a monitorování provozu na síti. Použití BPF pro pakety jdoucí skrze zařízení není příliš časté, avšak [8] uvádí, že i toto je možné.

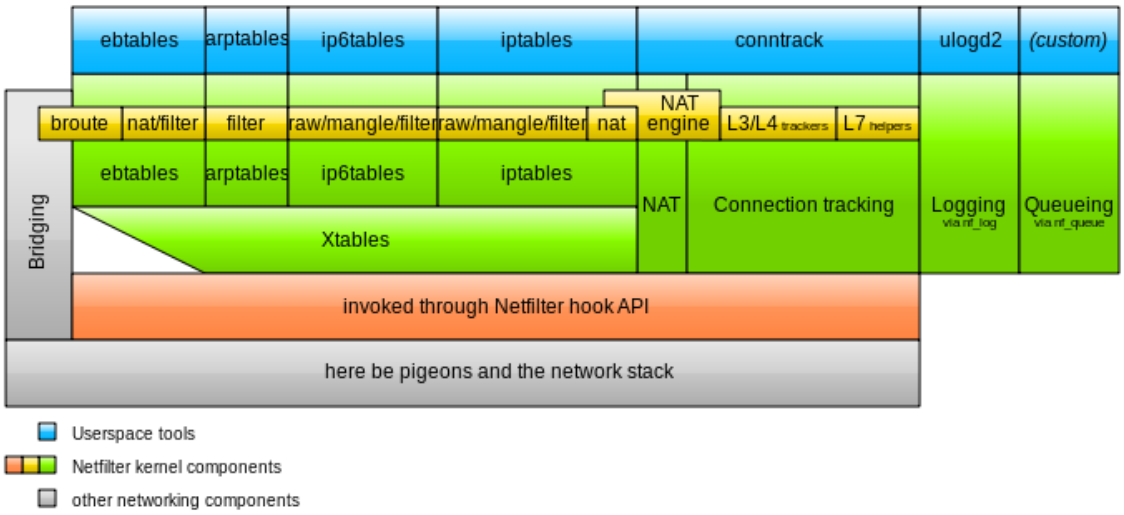
5.6 Srovnání

Z výše uvedených dostupných možností se jako použitelné pro univerzální softwarový firewall jeví framework Netfilter a Berkeley Packet Filter. Zvolen byl právě Netfilter. Důvodem jeho výběru oproti BPF byl hlavně fakt, že Netfilter je součástí standardního jádra operačního systému Linux v současných verzích. Dále

se Netfilter v současnosti hodně používá, a to nejen pro účel filtrování síťového provozu, což je bezesporu výhodou. Framework je stále udržovaný a stále se na něm pracuje. Detailněji je Netfilter popsán v následující kapitole.

6 Netfilter

Netfilter (v textu budeme dále uvádět zkráceně jako NF) představuje sadu přípojných míst (v angličtině se tato místa označují slovem *hooks*) v implementaci protokolového zásobníku v Linuxu, díky kterým je možné registrovat do těchto míst různé moduly jádra, které nějakým způsobem mohou s daty v různých fázích zásobníku pracovat.



Obrázek 6.1 Netfilter komponenty

NF je spojen především se známým programem *iptables*, jenž se dnes používá jako firewall nástroj pro Linux. Ve skutečnosti je iptables pouze programem běžícím nad NF a využívající jej. Používá však stejná jména přípojných míst a řetězců, proto se tyto dva pojmy často nesprávně považují za jedno a to samé. Rozdělení a strukturu celého frameworku můžeme vidět na obrázku 6.1 [9].

6.1 Architektura

Jak již bylo napsáno, Netfilter tvoří přípojná místa umístěna v protokolovém zásobníku, která dokážou vyvolat obslužné metody. Rozmístění jednotlivých míst / bodů je samozřejmě zvoleno pečlivě a každé umístění má své odůvodnění. Obslužné metody tvoří moduly jádra. V místě přípojných bodů se pak ze síťového kódu jádra zavolají metody registrovaných modulů. Každá funkce modulu má svou prioritu, podle které jsou jednotlivé funkce jednotlivých modulů volány.

Každá zavolaná funkce registrovaná k jednomu z přípojných bodů má k dispozici paket, se kterým může dle libosti manipulovat. Modul může udělat s paketem následující [10]:

- pokračovat v procházení paketu beze změny (NF_CONTINUTE),
- zahodit paket a ukončit jeho procházení zásobníkem (NF_DROP),
- poslat paket z kódu jádra do uživatelského prostoru a zařadit jej do předem definované fronty (NF_QUEUE),
- zavolat metodu přípojného místa znovu (NF_REPEAT),

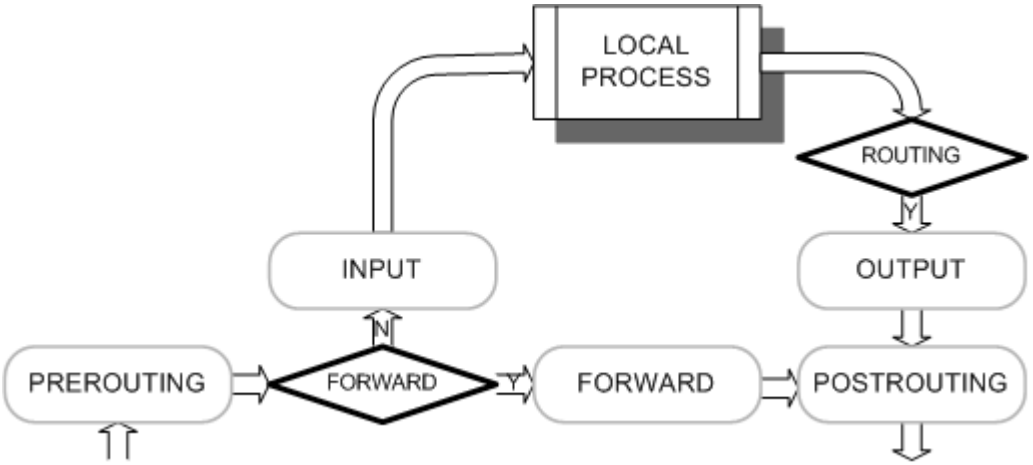
- nepokračovat v procházení zásobníkem, jelikož se modul dostal „přes“ samotný paket (NF_STOLEN).

Použití front je popsáno v následujícím textu v kapitole 6.1.2.

Jako názorný příklad si uvedeme IP paket a jeho průchod zásobníkem. Po přijetí do operačního systému se nejprve provedou prvotní kontroly paketu (např. kontrolní součet). Poté je vyvolán první bod NF_IP_PRE_ROUTING (PREROUTING). Dále se rozhoduje, zda je paket určen danému zařízení nebo se bude směrovat na jiný port zařízení. Podle toho jsou vyvolány další přípojné body NF_IP_LOCAL_IN (INPUT) nebo NF_IP_FORWARD (FORWARD). V případě směrovaného paketu pak tento paket projde kódem k závěrečnému bodu NF_IP_POST_ROUTING (POSTROUTING) a poté je odeslán.

Odchozí IP pakety, které jsou vygenerované samotným zařízením, po předání do protokolového zásobníku projdou nejprve přípojným bodem NF_IP_LOCAL_OUT (OUTPUT), poté se pro ně spustí směrovací kód a dále jsou předány závěrečnému přípojnému bodu NF_IP_POST_ROUTING. Ve skutečnosti je směrovací kód spuštěn před odchozím bodem NF_IP_LOCAL_OUT, aby byly vypočteny zdrojová adresa a některé IP možnosti (popsáno v kapitole 4.3.3) [10].

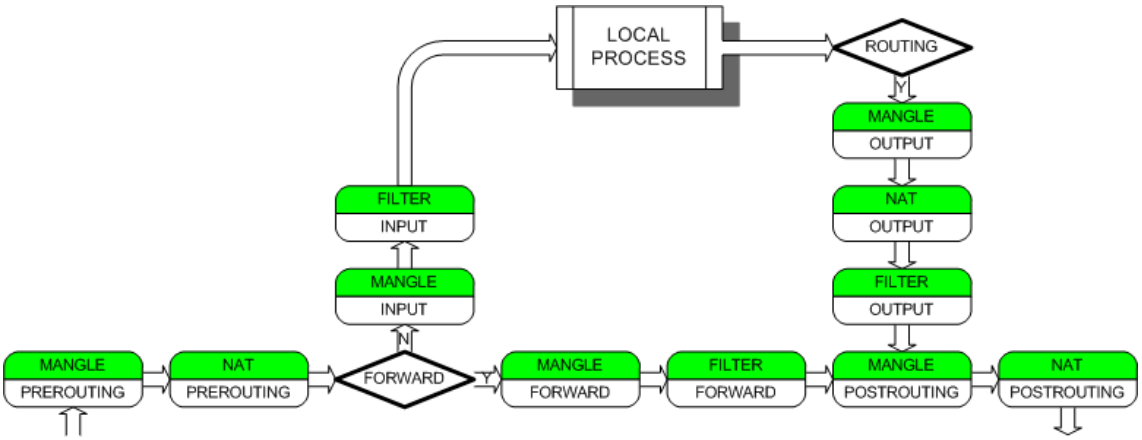
Celý průchod a jednotlivé přípojné body jsou znázorněny na následujícím obrázku 6.2 [11].



Obrázek 6.2 Přípojné body NF

6.1.1 Tabulka filtrování paketů

Jednotlivé moduly jádra vytvářejí a registrují v NF tabulky. Pakety pak těmito tabulkami procházejí. Samotný NF obsahuje tabulky FILTER, NAT a MANGLE. Jejich místa připojení jsou znázorněna na obrázku 6.3 [11].



Obrázek 6.3 Tabulky a přípojné body NF na síťové vrstvě

MANGLE tabulka se používá k modifikacím některých informací v paketu a tato tabulka je připojena na všechny přípojné body NF.

NAT tabulka zajišťuje překlad adres. Je připojena k bodům NF_IP_PRE_ROUTING a NF_IP_POST_ROUTING.

Pro nás z hlediska filtrování paketů je nejdůležitější tabulkou tabulka FILTER. V této tabulce nelze obsah jednotlivých paketů nijak měnit, pouze pakety filtrovat. Tabulka je připojena k bodům NF_IP_LOCAL_IN, NF_IP_LOCAL_OUT a NF_IP_FORWARD. Jak je vidět z obrázků 6.2 a 6.3, každý paket prochází filtrační tabulkou právě jednou, což představuje jisté zjednodušení pro uživatele oproti jeho předchůdci *ipchains* [10].

6.1.2 Uživatelské fronty paketů

Jak již bylo napsáno dříve v této kapitole, jednou z akcí NF, kterou lze aplikovat na procházející pakety, je odeslání paketu z kódu jádra do uživatelského prostoru a zařazení do jedné z uživatelem definovaných front.

V případě zařazení paketu do uživatelské fronty je nutné jej do určité doby předat z fronty zpět do jádra či dále, aby nedocházelo ke ztrátě paketů nebo zpožďování komunikace.

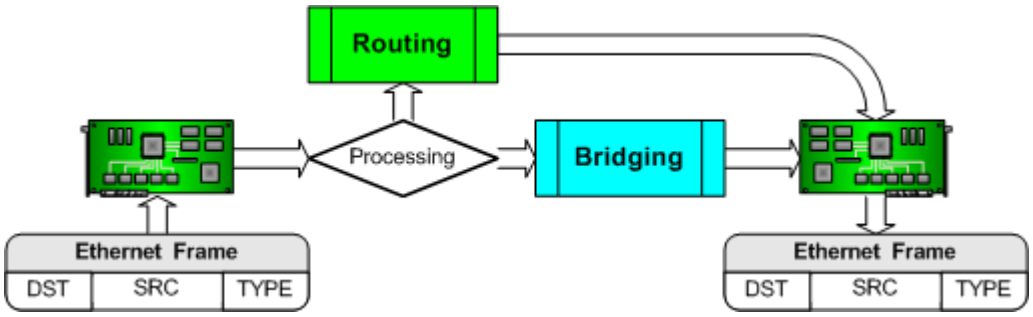
Fronty si musí uživatel nadefinovat předem, programem iptables. Pokud je pak v jádře vyvolán kód obslužné „připojené“ metody a ta rozhodne na základě pravidel nebo konfigurace o zařazení paketu do uživatelské fronty, je paket přístupný v dané frontě z uživatelského prostoru. V uživatelském prostoru se pak nejčastěji spouští program (kód), který pakety z fronty vybírá a nějakým způsobem je zpracovává, přičemž tento program neběží v jádru operačního systému a není jeho součástí. Může jít o různé programy. Je už na samotném kódu programu, jak bude s paketem vybraným z fronty naloženo. Pro vybírání paketů a manipulaci s pakety existuje knihovna *libnetfilter_queue*, která je dodávána organizací stojící za celým NF a která představuje API.

6.2 Netfilter na linkové vrstvě

Celý Netfilter framework cílí především na síťovou vrstvu a IP protokol. To je z hlediska jeho využití a koncepce celého Internetu zcela logické. Z jednotlivých komponent znázorněných na obrázku 6.1 lze ale vyčíst, že NF není jenom o IP protokolu (iptables, ip6tables), ale obsahuje další možnosti. Tou pro nás důležitou a použitelnou vzhledem k linkové vrstvě je komponenta *ebtables*.

Název ebtables vznikl složeninou anglického sousloví Ethernet Bridge Tables. Komponenta je dostupná v jádře systému Linux od verze 2.6 výše (existuje i patch pro verzi 2.4) a rozšiřuje možnost NF o filtrování či překlad MAC adres na linkové vrstvě pro rámce sítě Ethernet, jakožto nejrozšířenějšího typu počítačové sítě.

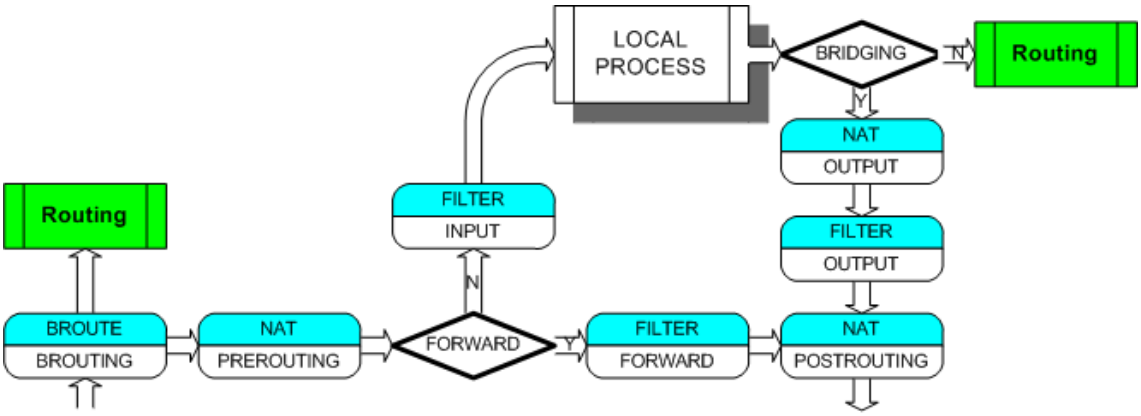
Ebtables lze použít pouze v případě, že na daném zařízení v operačním systému Linux je nakonfigurován logický most (bridge). Pouze v tomto případě pakety (ve formě rámců) prochází zásobníkem v kódu linkové vrstvy, kde má NF nadefinované a připravené přípojné body. V opačném případě se rámce ihned rozbalují a putují rovnou do vrstvy síťové jako pakety. Průchod je naznačen na obrázku 6.4 [11].



Obrázek 6.4 Průchod rámce skrze NF

6.2.1 Přípojné body a tabulky na linkové vrstvě

Přípojné body NF pro komponentu ebtables jsou téměř shodné s body na síťové vrstvě (iptables komponenta). Jak je vidět z obrázku 6.5 [11], přibyl pouze bod BROUTING. Na tento speciální bod je připojena tabulka *broute*, která umožňuje na základě informace z rámce / paketu buďto poslat rámec / paket do kódu směrovače nebo mostu [11]. Tato vlastnost nás však z hlediska filtrování příliš nezajímá. Důležité je, že filtrovací tabulka používá stejné body jako na síťové vrstvě. NF i na této vrstvě obsahuje NAT tabulku, MANGLE však chybí.



Obrázek 6.5 Přípojné body a tabulky NF na linkové vrstvě

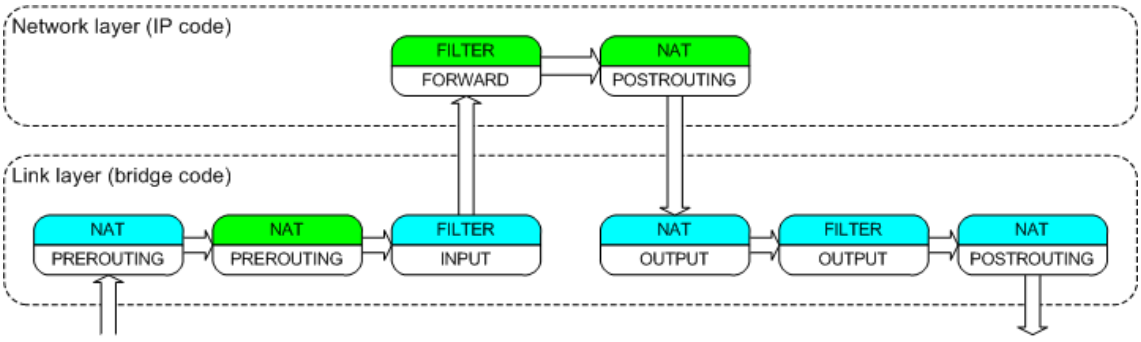
Pro každý přijatý rámec platí následující rozhodnutí:

- Pokud je cílová hardwarová adresa známá a je na jiné straně mostu, je rámec předán.
- Pokud je cílová hardwarová adresa známá a je na stejné straně mostu, odkud rámec přišel, je ignorován.
- Pokud je cílová hardwarová adresa neznámá, je rámec rozeslán na všechny ostatní porty mostu.
- Pokud je cílovou hardwarovou adresou adresa zařízení, je rámec předán vyšší vrstvě.

6.2.2 Spojení se síťovou vrstvou

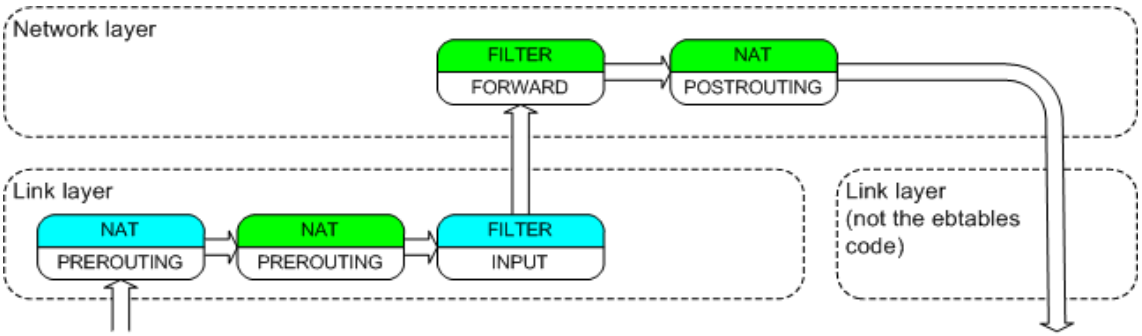
Výše jsou popsány obě základní vrstvy NF a naznačen průchod dat skrze jejich jednotlivé přípojné body a tabulky. Obě dvě vrstvy lze samozřejmě zkombinovat. Jak si ukážeme dále, průchod paketů jednotlivými tabulkami na jednotlivých vrstvách není vždy stejný. Záleží na vlastnostech samotného paketu a zejména na příchozích a odchozích portech.

Obrázek 6.6 [11] naznačuje průchod paketů skrze NF v zařízení, které je použito jako most a zároveň i jako směrovač. Obrázek ukazuje případ, kdy je rámec na linkové vrstvě určen pro samotné zařízení, avšak na síťové vrstvě bude paket přeposlán. Paket pak opustí zařízení opět na jednom z portů mostu.



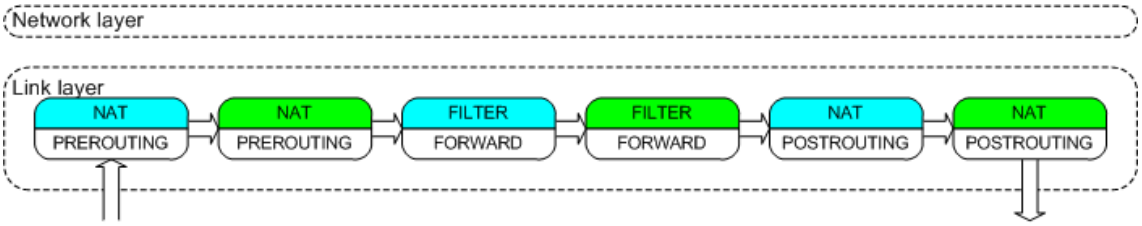
Obrázek 6.6 Směrování paketu na port mostu

Pokud je však na síťové vrstvě rozhodnuto, že se paket přepošle na port, který není součástí mostu, neprochází už tabulkami na linkovou vrstvu, jak znázorňuje obrázek 6.7 [11].



Obrázek 6.7 Směrování paketu na port směrovače

V dalším případě se jedná o pakety, které jsou pouze předávány z jednoho portu mostu na druhý. Tyto pakety se vůbec nedostanou na síťovou vrstvu [11]. NF však i v tomto případě pakety posílá skrze tabulky iptables, jak je naznačeno na obrázku 6.8 [11].



Obrázek 6.8 Bridge paketu

6.3 Shrnutí

V příloze A této práce nalezneme ucelený detailní pohled průchodu paketu NF na linkové i síťové vrstvě. Jsou zde názorné všechny možné cesty, kterými jednotlivé pakety mohou procházet, společně se všemi přípojnými body a tabulkami, které jsou na těchto cestách zařazeny. Z pohledu filtrování paketů jsou pro nás důležité především tabulky FILTER jednotlivých modulů NF a jejich řetezce.

Jak je v příloze také naznačeno, hlavní volbou, kudy jednotlivý paket bude procházet, je rozhodování o zpracování paketu v módu mostu nebo směrovače. Toto záleží na konfiguraci zařízení a portu, na jakém je paket přijat. V případě, že je paket rovnou při přijetí poslán do směrovací části kódu, automaticky tak přicházíme o možnost filtrování na základě informací z hlavičky linkové vrsty. Stejné chování pak můžeme pozorovat na straně odeslílání paketů, kde možnost filtrace na linkové vrstvě opět závisí na konfiguraci portu, kterým paket zařízení opouští.

Z pohledu transparentního filtrování v zařízení pracujícím jako síťový most nabízí NF framework zajímavé a velmi často dostačující možnosti filtrace.

7 Návrh univerzálního softwarového firewallu

7.1 Specifikace filtrovacího pravidla

Filtrovací pravidlo představuje jakousi množinu nastavení jednotlivých voleb v hlavičce paketu, vůči které se samotný paket porovnává. Výsledkem porovnání je shoda nebo neshoda. V případě shody pak pravidlo definuje, co se s daným paketem udělá.

Kromě přesných hodnot polí může pravidlo definovat množinu. Jako příklad lze uvést IP adresu. Můžeme nadefinovat pravidlo, které bude aplikováno v případě přesné shody s nějakou například zdrojovou adresou. Toto však ve skutečnosti nebude příliš praktické. Pokud budeme chtít povolit komunikaci určitému segmentu sítě, který by se při správné konfiguraci sítě měl vyznačovat stejným síťovým prefixem adresy (adresa sítě, k jejímu oddělení se vedle IP adresy uvádí i takzvaná maska sítě), museli bychom definovat pro každé zařízení pravidlo zvlášť. Místo toho můžeme ale definovat pravidlo jedno, kde jako IP adresu zdroje uvedeme právě adresu sítě společně s maskou sítě. Toto pravidlo pak zafunguje pro všechny zařízení daného segmentu.

Další možností u každé hodnoty je její negace. Pro vysvětlení si uvedeme opět názorný příklad. Pokud budeme chtít povolit veškerou komunikaci kromě protokolu UDP, zajisté můžeme postupně povolit každý známý protokol zvlášť. Jednodušší však je jedním pravidlem zakázat pakety nesoucí UDP protokol a jedním pravidlem povolit pakety, které nesou „něco jiného“, než UDP, tj. takové pakety, kde se informace v hlavičce indikující nesoucí protokol nerovná protokolu UDP. Znalý čtenář zde jistě namítne, že tuto situaci lze vyřešit jednodušeji nastavením výchozí akce (viz kapitola 7.1.3), a má pravdu. Nicméně zde tuto úvahu bereme pouze jako ukázkový příklad pro názorné použití negací ve filtrovacích pravidlech.

Samozřejmě ve filtrovacím pravidle nemusíme hodnotu vůbec specifikovat. V tom případě bude tato při porovnání vynechána a nebude na ni brán zřetel. Jinak řečeno, pravidlo zafunguje pro jakoukoliv hodnotu daného parametru.

7.1.1 Parametry filtrovacího pravidla

Jelikož navrhovaný systém má být systémem univerzálním a zároveň má být schopen filtrovat síťový provoz i na linkové vrstvě, byly parametry, podle kterých lze pakety filtrovat, vybrány následovně:

- vstupní a výstupní porty do / ze zařízení,
- na linkové vrstvě zdrojová a cílová hardwarová adresa,
- na linkové vrstvě typ neseného protokolu síťové vrstvy,
- na síťové vrstvě zdrojová a cílová hardwarová adresa (včetně masky),
- na síťové vrstvě typ neseného protokolu vyšší transportní vrstvy.

Vstupní a výstupní porty není třeba nějak složitě vysvětlovat. Jedná se o port, na kterém byl paket do zařízení přijat případně ze zařízení odeslán. Tento parametr se nevztahuje k linkové ani síťové vrstvě, avšak jedná se o důležitý parametr, který by neměl zůstat opomenut. Ve skutečnosti totiž filtrace probíhá v hraničních

bodech počítačové sítě, kde se může spojovat několik segmentů, z nichž každý je připojen na jeden port (lze i více). My tak dokážeme do velké míry říci, jaké adresy se mohou objevit na určitých segmentech a tím i portech. Případ, že se objeví na některém portu námi neočekávaná adresa, je velmi často indikací pokusu o útok.

Zdrojové a cílové hardwarové adresy linkové i síťové vrstvy jsou také velmi jasným parametrem. V obou případech lze zadat rozsah adres maskou sítě.

Typ protokolu neseného linkovou vrstvou neboli typ protokolu síťové vrstvy představuje parametr s omezeným, avšak velkým, výběrem možností (lze použít pouze u sítí typu *Ethernet*, viz kapitola 4.2). Uživateli tak není dovoleno specifikovat parametr napsáním libovolné hodnoty. Pro příklad si můžeme uvést některé nejčastější možnosti tohoto parametru:

- IP protokol verze 4 (IPv6),
- IP protokol verze 6 (IPv4),
- ARP protokol (ARP),
- protokol Apple Talk (ATALK),
- nespecifikovaný protokol staršího rámce typu Ethernet 802.3 (LENGTH), viz kapitola 4.2.

Parametr typ neseného protokolu síťovou vrstvou je způsobem zadání shodný s protokolem neseným na linkové vrstvě, který je popsán v předchozím odstavci. Opět se jedná o výběr možností z nadefinované omezené množiny. Pro příklad si uvedeme možnosti *tcp*, *udp*, *icmp* a *all*. Poslední možnost pak slouží jako neutrální volba parametru, tj. parametr nespecifikován.

Kromě parametrů týkajících se porovnávání s pakety musí každé pravidlo obsahovat další informace. Jednou z nich je samozřejmě typ akce, která se má provést s paketem, který bude odpovídat pravidlu v porovnávacích parametrech. Akce na výběr budou celkem pouze dvě. První z nich povolí paketu průchod dále (ACCEPT), druhý pak paket zahodí bez odeslání zprávy o zahození příjemci zpět (DROP). Druhá možnost zahození s odesláním zprávy o zahození paketu, která se označuje jako REJECT, bývá také zneužívána pro připravování útoků, proto tuto možnost náš firewall nenabízí. Navíc v případě transparentního filtrování na linkové vrstvě (mód síťového mostu) je tato možnost vlastně i nežádoucí, jelikož transparentní filtr by o sobě neměl dávat vůbec vědět. Využití by se tak našlo pouze ve filtrování v módu síťového směrovače.

Mezi další parametry pravidla pak patří jeho název a popis. Název by měl být výstižný a krátký. Slouží pro jednoznačnou identifikaci v nějakém seznamu pravidel při správě a konfiguraci firewallu. Naopak popis může být již delší a konkrétnější než název.

Všechny parametry pravidla platí v jeho definici najednou. Mluvíme zde tedy o logickém součinu (AND) mezi parametry. Pouze v případě, že paket vyhovuje všem parametrům najednou, je jeho další zpracování závislé na definované akci onoho pravidla. V opačném případě probíhá porovnání s dalším pravidlem v řadě. Pokud tedy definujeme pravidlo pro filtraci na základě hlaviček protokolů použitých na linkové a síťové vrstvě zároveň, musíme brát v potaz tu skutečnost, že daný paket

/ rámec musí splňovat požadavky definované v pravidle zároveň. Nelze tímto pravidlem ovlivnit pakety, které ačkoliv mají například stejnou hlavičku protokolu síťové vrstvy (standardně IP), nesplňují už požadavky na linkové vrstvě. Ovlivnění takových paketů je chybnou interpretací.

7.1.2 Dva typy pravidel

Z důvodů popsaných v kapitole 6.2 nelze dosáhnout při definování pravidel takové univerzálnosti, že jedním stejným pravidlem nadefinujeme filtraci jak v režimu směrovače, tak síťového mostu. Kvůli tomu existují v našem firewallu dva typy pravidel. Ačkoliv samotný firewall úlohu filtrovaní v režimu směrovače i mostu zvládne najednou. Problém spočívá v různých průchodech paketů kódem použitého frameworku Netfilter. Především se jedná o zařazení filtračních tabulek schopných analyzovat linkovou vrstvu paketu / rámce pouze na portech zařazených konfigurací do logického síťového mostu zařízení. Jinak řečeno na portech síťového mostu, viz kapitola 6.3. Toto je však vcelku logické chování. Přijetí na jiný port, než port mostu, znamená, že je paket na linkové vrstvě určen pro zařízení samotné. Obsah hlavičky protokolu linkové vrstvy tak dokážeme předpovědět a hlavně bude vždy takřka shodný (může se lišit v typu neseného protokolu).

První typ pravidel bude aplikován na všechny pakety jdoucí skrze zařízení v módu síťového mostu. Tyto pravidla nabízí možnost filtrovacích parametrů na obou vrstvách – linkové i síťové. Uživatel nejprve specifikuje parametry linkové vrstvy (viz kapitola 7.1.1) a pokud zvolí protokol nesený linkovou vrstvou, který lze dále analyzovat (podpora IPv4 a IPv6), může dále zvolit další parametry týkající se vybraného protokolu síťové vrstvy.

Druhým typem pravidel jsou pravidla aplikovatelná na všechny typy paketů bez rozdílu, zda se jedná o průchozí pakety v módu síťového mostu nebo pakety přijaté na standardních portech zařízení jako síťová stanice či směrovač. Jelikož se pro tyto pakety nespouští kód a přípojné body frameworku Netfilter v linkové vrstvě, tyto pravidla nepodporují filtrovací parametry linkové vrstvy. Podporována je tak pouze vrstva síťová a pouze ty síťové protokoly, které lze analyzovat (podpora IPv4 a IPv6). Na druhou stranu těmito pravidly lze filtrovat i provoz síťového mostu. Oproti prvnímu typu pravidel však tímto typem můžeme na mostě filtrovat pouze na základě informací protokolu síťové vrstvy. Tento typ pravidel tak nabízí široký zásah na všechny typy paketů bohužel na úkor zmenšení množiny parametrů.

7.1.3 Princip porovnávání a výchozí akce

Jednotlivá pravidla jsou v systému uložena v uživatelem definovaném pořadí. Při průchodu paketu systémem se vezme první pravidlo ze seznamu a paket se vůči němu porovná. V případě, že porovnáním je pravidlo na paket aplikováno, je s paketem naloženo dle akce pravidla. V případě, že pravidlo na paket aplikováno být nemůže, je vybráno další pravidlo z pořadí. Takto paket prochází celým seznamem pravidel.

V systému je nutno pravidla tedy řadit. Pravidla obou typů (viz kapitola 7.1.2) se nacházejí ve stejném seznamu, tj. v jednom pořadí. Ve skutečnosti je tedy potřeba

brát v potaz, že některá pravidla ze seznamu mohou být při filtraci přeskočena. Záleží na typu pravidla a zpracovávaného paketu.

Pokud při filtraci na paket nelze aplikovat ani jedno z definovaných pravidel, použije se takzvaná *výchozí akce*. Výchozí akce je definovaná v nastavení celého firewallu a stejně jako u pravidla jsou zde možnosti ACCEPT a DROP. Od nastavení výchozí akce se pak odvíjí celá konfigurace firewallu. Záleží i na bezpečnostní politice dané počítačové sítě. V některých případech se zakazuje veškerá komunikace právě výchozí akcí a povoluje se jen komunikace určitých parametrů (pravidla s akcemi ACCEPT). Někde je naopak politika taková, že je veškerý provoz výchozí akcí povolen a určitá komunikace se zakazuje (pravidla s akcemi DROP).

7.2 Netfilter jako základ

Navrhovaný univerzální softwarový firewall musí zvládat filtrování paketu na linkové i síťové vrstvě. Dále musí umět filtraci jak v režimu síťového mostu, tak v režimu směrovače. V tomto ohledu lze nalézt úspěšné i neúspěšné projekty, jejímž cílem je filtrování paketů, případně další možnosti práce s pakety v síťovém provozu.

Jako základní stavební kámen našeho softwarového firewallu byl zvolen framework ***Netfilter***. Framework je popsán v kapitole 6. Zde pouze zopakujeme jeho hlavní výhody, jež představují argumenty pro zvolení právě této cesty.

Netfilter představuje framework pro práci s prostředky síťové komunikace na platformě Linux. V dnešní době se jedná o nejrozšířenější projekt v této oblasti, což dokazuje i fakt, že je standardní součástí jádra operačních systémů typu Linux. Je hojně používaný (což znamená také vyzkoušený) a stále se na něm pracuje. Tím je do jisté míry zajištěn i rozvoj v případě nových verzí současně používaných protokolů. Z hlediska filtrování provozu nám jeho funkce dostačují. Z frameworku použijeme hlavně jeho tři komponenty. Komponenty ***iptables*** a ***ip6tables***, kterými dokážeme filtrovat pakety na síťové úrovni především v módu síťového směrovače (ip6tables pracuje s IP protokolem verze 6, iptables pak s verzí číslo 4), a komponentu ***ebtables***. Tímto modulem naopak můžeme filtrovat pakety na linkové vrstvě v módu síťového mostu.

Použitím frameworku Netfilter však práce nekončí, naopak se dá říci, že tím vše začíná. Samotný framework je velmi rozsáhlý. Ačkoliv jsme si už řekli, jaké komponenty budeme k filtrování používat, je nutné tyto komponenty správně skloubit dohromady a využívat jejich jednotlivé části zvlášť či najednou tak, aby byl výsledný systém firewallu a především pak konfigurace pro jeho obsluhu jednoduchá a přehledná.

7.2.1 Aplikační rozhraní

Jak už zde bylo několikrát napsáno, Netfilter je z větší částí součástí jádra operačního systému. Jelikož náš firewall bude běžet v operačním systému nějakého počítače nebo řekněme zařízení v normálním uživatelském režimu, musíme s kódem v jádře nějak „komunikovat“. Zde připadají v úvahu dvě možnosti.

První možností je získat z kódu jádra samotná komunikační data, čili paket nebo rámec, a v aplikaci použít vlastní filtrovací algoritmus. To znamená analýzu

hlavičky (případně hlaviček) paketu a rozhodnutí o dalším pokračovaní na základě porovnání v ní obsažených informací s filtrovacími pravidly zadanými do systému uživatelem. V případě povolení paketu bude pak paket předán zpět do kódu jádra pro jeho další zpracování.

Na tomto principu fungují uživatelské fronty (viz kapitola 6.1.2), kterými bychom si mohli „poslat“ všechny pakety z kódu jádra do uživatelského prostoru a poté zpět říci verdikt. Zahodit nebo propusti paket dále. Zde ovšem musíme říci kódu v jádře, jaké pakety má předávat do uživatelských front. To uděláme filtrovacím pravidlem. I když použijeme uživatelskou frontu pro všechny pakety, stejně bude potřeba do tabulek frameworku v jádře zapsat alespoň jedno pravidlo. Navíc získání celého paketu včetně jeho hlavičky z uživatelské fronty nepředstavuje triviální činnost. Zde by se tak mohlo jednat i o kritický bod z hlediska propustnosti systému. Pokud by byl algoritmus analýzy paketů příliš náročný, mohl by výsledný firewall při větší síťové komunikaci výrazně omezovat celkový síťový provoz.

Proto se přikloníme k druhému způsobu, jímž je nechat porovnávání paketů na frameworku, kde je již toto vyřešeno a vyzkoušeno. Pakety tak zůstávají v kódu jádra. Zde je nutné ovšem vyřešit princip zápisu filtrovacích pravidel z našeho firewallu do tabulek frameworku. Toto by oproti prvnímu způsobu (uživatelské fronty) nemělo představovat žádné extremní navýšení požadavků na zdroje zařízení při vyšším síťovém provozu.

Součástí projektu Netfilter jsou i jednoduché konzolové aplikace (uživatelský prostor OS), které jsou pojmenované stejně, jako jednotlivé komponenty (znázorněno na obrázku 6.1), a které slouží k manipulaci s filtrovacími pravidly uloženými v tabulkách odpovídajících komponent v jádře. Aplikace využívají pro komunikaci s komponentami jádra knihovny (*libiptc*, *libebtc*). Bohužel tyto knihovny nejsou označeny za aplikační rozhraní (API) a autoři a vývojáři projektu Netfilter jejich použití jako rozhraní k jednotlivým komponentám nedoporučují, jelikož se tyto knihovny mohou v dalších verzích frameworku změnit či odstranit [12].

Vývojáři tak doporučují jako náhradu za prozatím chybějící rozhraní a nejschůdnější řešení použití příkazu *[x]tables-restore* [12]. Tento program se používá pro nahrání filtrovacích pravidel do tabulek obecně ze vstupu do programu (například ze souboru). Program je součástí konzolových aplikací pro obsluhu pravidel. [x] v názvu příkazu v praxi nahradíme prefixem odpovídající komponenty a jejího aplikačního nástroje. Pro opačný proces (výpis pravidel) se používá obdobně příkaz *[x]tables-save*.

7.2.2 Použití tabulek a řetězců

Pro účely filtrování síťového provozu použijeme z frameworku Netfilter pouze jednu tabulku s příhodným názvem ***FILTER***. Tabulka slouží právě pro účely filtrování paketů. Tuto tabulku obsahuje každá z námi využitých komponent (iptables, ip6tables, ebtables).

Tabulka dále obsahuje takzvané řetězce. Každý z řetězců již v sobě obsahuje seřazená filtrovací pravidla. V každé z komponent, které použijeme, má tabulka

FILTER řetězce shodné – *input*, *output* a *forward*. Jak jejich název napovídá, každý z řetězců je spouštěn přípojným bodem v jiných místech kódu jádra. Detailněji je toto popsáno v kapitole 6 a graficky znázorněno na kompletním grafu všech řetězců a tabulek frameworku Netfilter v příloze A této práce. Tabulka tedy specifikuje možnost a formu definice pravidla, zatímco řetězcem specifikujeme, v jaké části zpracování a průchodu paketu se bude pravidlo aplikovat.

Dále je nutné uvést, že zatímco pravidla v tabulkách síťové vrstvy (iptables, ip6tables) umí operovat s hodnotami v hlavičkách protokolů síťové vrstvy, komponenta linkové vrstvy (ebtables) a její filtrovací tabulka dokáže analyzovat hlavičku protokolu vrstvy síťové i linkové. To v praxi znamená, že pravidlo v této tabulce dokáže rozhodnout o paketu na základě informací linkového i síťového protokolu. Tato funkčnost je velmi vítanou, jelikož v jedné tabulce / jejím řetězci dokážeme aplikovat logický AND všech parametrů pravidla najednou.

Pokud mluvíme o filtrování paketů někde na hraničním zařízení v síti, hlavním řetězcem pro nás bude řetězec *forward*. Právě tímto řetězcem prochází všechny pakety jdoucí skrze naše zařízení. A to platí v případě módu směrovače i síťového mostu. Rozdíl je pouze v komponentě. Zatímco směrované pakety prochází forward řetězcem tabulky FILTER z komponenty iptables, respektive ip6tables, přepínané pakety jdoucí skrze síťový most prochází řetězcem stejného názvu ve stejnojmenné tabulce ovšem komponenty ebtables, viz kapitola 6.

Všechny filtrovací pravidla se tedy standardně zapisují do řetězců forward. Pravidla typu pro pakety pouze v módu síťového mostu se zapisují pouze do řetězce tabulky FILTER v komponentě ebtables (na obrázku z přílohy modrá barva). Pravidla druhého typu, tedy pro všechny typy paketů, se zapisují do obou tabulek FILTER, tedy síťové i linkové vrstvy (modrá i zelená barva). Jak jsme si již vysvětlili v kapitole 7.1.2, tento typ pravidel umožňuje zadat parametry pouze pro síťovou vrstvu. To je právě z toho důvodu, že tabulka na síťové vrstvě umožňuje pouze tyto parametry oproti tabulce na linkové vrstvě. Při zápisu do tabulky na linkové vrstvě se tedy u tohoto typu pravidel zapisují pouze parametry síťové vrstvy.

Pokud se navíc vrátíme k obrázku 6.8, zjistíme, že při průchodu paketu skrze zařízení v módu síťového mostu se i pro tento paket spouští filtrování podle pravidel v tabulce síťové vrstvy. Toto chování je výhodou při použití filtrace pouze na síťové vrstvě, čímž tato filtrace může ovlivnit i přepínané pakety v mostu. Jelikož ale my používáme pro filtraci tabulky obou vrstev, je pro nás toto chování nežádoucí. Při zachování tohoto modelu bychom totiž mohli nesprávně ovlivnit filtrací některé pakety. Například pokud budeme mít pravidlo pouze pro přepínané pakety, které bude povolovat pakety se zdrojovou IP adresou 192.168.2.5 a zdrojovou MAC adresou 3C:7B:1A:56:78:E0, a zároveň pravidlo pro všechny pakety, které zakáže komunikaci z IP segmentu 192.168.2.0/24. Pravidla budou definována v tomto pořadí. Takto tedy může ze segmentu 192.168.2.0/24 komunikovat pouze zařízení s IP 192.168.2.5 a s MAC 3C:7B:1A:56:78:E0. Navíc tyto pakety projdou pouze skrze most, v módu směrovače se i toto zařízení „zablokuje“. Problémem však je, že na mostu pakety projdou první filtrovací tabulkou z komponenty ebtables a poté putují do druhé tabulky iptables, kde dle druhého pravidla skončí. My dle definice pravidel ale tyto pakety chceme propustit

dále. Pokud však tabulku síťové vrstvy (iptables) odstraníme z průchodu přepínaných paketů v mostu, bude tato tabulka využita pouze pro směrované pakety a naopak tabulka linkové vrstvy bude ovlivňovat pouze přepínané pakety. Toto chování je pro náš případ firewallu lepší. Naštěstí lze tabulku z průchodu odstranit jednoduchým zápisem *0* do proměnné operačního systému */proc/sys/net/bridge/bridge-nf-call-iptables*, respektive *bridge-nf-call-ip6tables* [13].

Firewall však nefiltruje pouze průchozí pakety, ale také pakety v jiných cestách. Konkrétně příchozí a odchozí pakety do zařízení v řetězcích output a input. Tyto řetězce však mohou obsahovat trochu upravená pravidla. Názorně si tento problém vysvětlíme na řetězci input, kde by se v pravidle neměl vyskytnout parametr výstupního portu. Vysvětlení je jednoduché. Pokud je paket určen pro zařízení, žádný výstupní port neexistuje. Obdobně je tomu i u řetězce output, kde by se zase v pravidle neměl objevit parametr vstupního portu. Námi navrhovaný firewall pamatuje i na tyto možnosti. Nechceme však po uživateli, aby nejdříve specifikoval řetězce, kam lze pravidlo zapsat, a tak se v tomto směru systém chová intuitivně. V případě, že uživatel zadá pravidlo, které neobsahuje vstupní port, systém při zápisu pravidel zapíše toto pravidlo standardně do řetězce forward a také do řetězce output. Chování je analogické při situaci, kdy uživatel nezadá výstupní port nebo ani jeden port (zapíše se do řetězců input, output i forward).

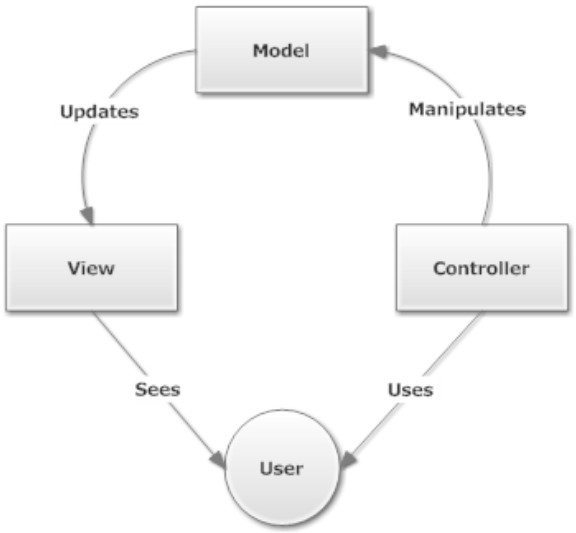
Stejné intuitivní chování se používá i v řetězcích na linkové vrstvě. Zde tímto chováním můžeme filtrovat příchozí pakety na portech síťového mostu s možnostmi analyzovat hlavičku protokolu linkové vrstvy před předáním paketu / rámce do vyšší vrstvy. Analogicky pak na druhé straně v situaci odesílání paketu.

7.3 Architektura

Aplikace softwarového firewallu je napsána v programovacím jazyce ***C++***. Při tvorbě grafického uživatelského rozhraní byla použita knihovna pro tvorbu grafických rozhraní - ***Qt***.

7.3.1 Model/view architektura

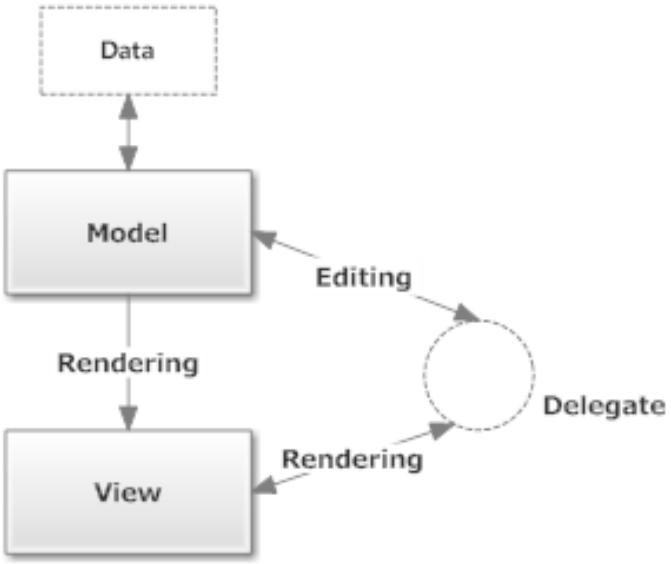
Model/view architektura je založena na architektonickém vzoru *Model-View-Controller* (MVC). Tento vzor se velmi často používá pro návrh uživatelských rozhraní. MVC se skládá ze tří hlavních skupin objektů. Skupina View obsahuje grafickou prezentaci dat, která jsou uložena v aplikaci v objektech skupiny Model. Skupina Controller pak zajišťuje reakci uživatelského rozhraní na podměty přicházející ze strany uživatele. MVC je naznačen na obrázku 7.1.



Obrázek 7.1 Návrhový vzor MVC

Architektura model/view představuje upravený architektonický vzor MVC, kdy jsou objekty skupiny Controller sloučené se skupinou View. Stále je však zachována hlavní výhoda vzoru MVC. Stále jsou samotná data aplikace oddělena od objektů, které data prezentují. Díky tomu můžeme jednotně uložená data prezentovat uživateli několika různými způsoby. V případě nových typů pohledu na data se tak nemusí měnit jejich struktura. Po sloučení skupin View a Controller do jedné skupiny View je však architektura aplikace jednodušší [14].

Právě tuto architekturu podporuje knihovna Qt, která byla použita pro implementaci grafického uživatelského rozhraní. Knihovna obsahuje různá rozhraní pro různé pohledy na data. Dále implementací nějakého z rozhraní v aplikačních objektech můžeme data v něm obsažená zobrazit v již předdefinovaných standardních prvcích uživatelského rozhraní, jako jsou například tabulky, stromy nebo seznamy. Tyto objekty je samozřejmě možné vylepšit k našemu prospěchu vytvořením vlastní třídy jako potomka jednoho z nich.



Obrázek 7.2 Model/view architektura

Při pohledu na obrázek 7.2 zjistíme, že i model/view architektura obsahuje třetí skupinu aplikací s názvem *delegate*. Tato skupina obsahuje objekty, které se využívají v případě, že uživatel v uživatelském rozhraní data mění, a také jako poskytovatelé dat v objektech skupiny view. Například při použití tabulky delegate zajišťuje zobrazení každé buňky, případně formu její editace. Právě tyto objekty pak komunikují s modelem dat „zpětně“. Výhodou skupiny delegate je, že můžeme měnit princip, jakým budou data zobrazována a editována.

7.3.2 Knihovní třídy

Kromě tříd ve skupinách architektury model/view nalezneme ve zdrojových kódech knihovní třídy. Jedná se o skupinu tříd, které zajišťují nějaké jim specifické činnosti. Například načítání dat z XML souboru nebo ukládání konfigurace. Téměř všechny třídy mají ale jedno společné. Z pohledu architektury aplikace zajišťují interakci s okolím.

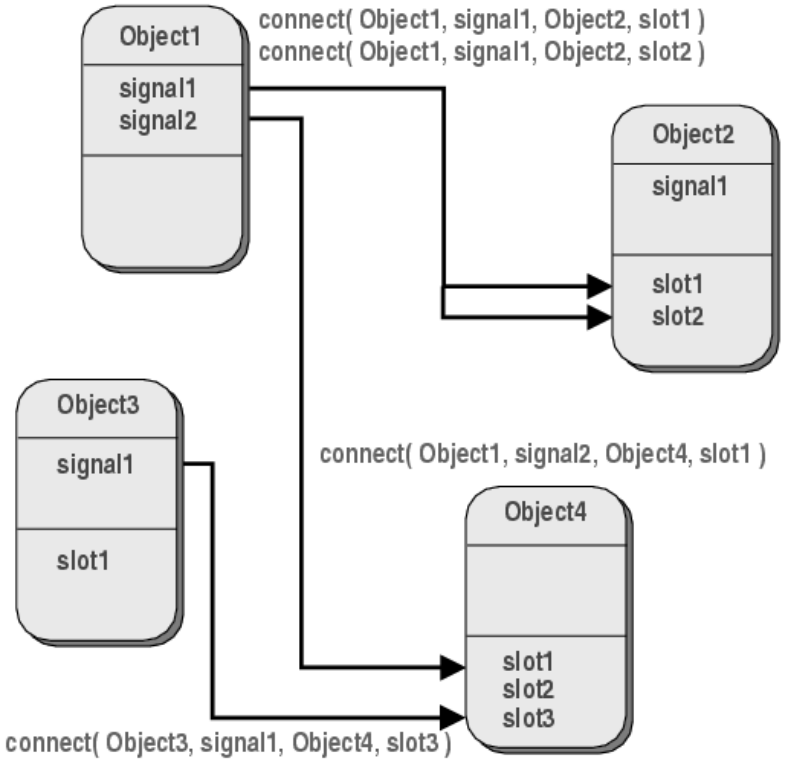
Pro nás jsou v tomto případě okolím různé soubory, kde se ukládají data, viz kapitola 7.4, a také Netfilter. Ten totiž není přímou součástí naší aplikace, jelikož jeho převážná část běží v kódu jádra. Z našeho pohledu ho tedy můžeme chápat jako běžící službu operačního systému, kterou v aplikaci využíváme. V knihovnách ale nalezneme i třídy, které vykonávají nějakou činnost uvnitř naší aplikace. Příkladem tohoto typu může být logování, které se používá napříč aplikací.

Obecně lze tedy říci, že knihovní třídy poskytují naší aplikaci služby. Už je jedno, jestli se jedná o služby v rámci aplikace nebo jestli se jedná o služby spojené s jinými aplikacemi či službami operačního systému. Zavedení knihovních tříd má výhodu v tom, že pokud je nutné službu upravit nebo pokud byla vnější služba změněna (nová verze, jiné volání apod.), stačí to udělat v knihovně a zásah se promítne skrze celou aplikaci.

7.3.3 Signály a sloty

Signály a sloty jsou jedním ze základních mechanismů *Qt*, který je použit v našem firewallu pro budování grafického uživatelského rozhraní. Mechanismus slouží pro komunikaci dvou a více objektů. Obecně tak lze pokrýt příklady, kdy jedna část uživatelského rozhraní vykoná nějakou činnost, na základě níž potřebujeme jinou část rozhraní informovat o této situaci a patřičně na ní zareagovat. Názorně můžeme uvést například situaci, kdy uživatel vybere v seznamu filtrovacích pravidel jedno pravidlo. V tomto případě pak požadujeme, aby část uživatelského rozhraní, která slouží k editaci filtrovacího pravidla, zareagovala tak, že vybrané pravidlo „načte“ do patřičných editačních komponent (editační pole, výběry možností, zaškrtávací políčka) a umožní uživateli toto pravidlo změnit.

Signály a sloty mají tu výhodu, že dané komunikující objekty o sobě nemusí přímo vědět. Připojení neboli zaregistrování slotu k signálu se většinou provádí v nějakém nadřazeném objektu (například hlavní okno uživatelského rozhraní apod.) při inicializaci objektů. Navíc na jeden vyslaný signál může zareagovat ve svých slotech libovolné množství zaregistrovaných objektů, viz obrázek 7.3 [15]. Každý přitom může reagovat svým způsobem.



Obrázek 7.3 Signály a sloty

Pokud tedy nějaký objekt změní svůj vnitřní stav a usoudí, že je vhodné dát ostatním o této skutečnosti vědět, může emitovat signál. Signál je definován jako normální funkce bez návratové hodnoty s libovolným počtem argumentů definovaných typů. Argumenty zde slouží pro předání informací / hodnot slotům. Po emitaci signálu dojde k zavolání všech slotů, které jsou k němu připojeny. Volání probíhá standardním způsobem jako volání normálních metod objektů.

Slot představuje standardně definovanou metodu objektu, jejíž jedinou nadstavbou nad metodou je fakt, že může být zaregistrována k nějakému (i více) signálům. Stejně jako signál, i v tomto případě metoda nemá žádnou návratovou hodnotu, pouze argumenty, které by měly odpovídat argumentům připojeného signálu včetně jejich pořadí. Může jich být i méně, než argumentů signálu. V tom případě budou použity pouze definované argumenty v hlavičce metody / slotu.

Pro ukázku si můžeme ukázat definici jednoduché třídy nějakého čítače. Třída má vnitřní atribut udržující stav hodnoty. Veřejný konstruktor, metodu pro získání hodnoty a metodu pro nastavení hodnoty, jež slouží zároveň jako slot. Dále třída obsahuje signál, který se emituje při každé změně hodnoty.

```
#include <QObject>
 class Citac : public QObject {
     Q_OBJECT
 public:
     Citac() { hodnota = 0; }
     int hodnota() const { return hodnota; }
 public slots:
     void setHodnota(int hodnota);
 signals:
     void hodnotaZmenena(int novaHodnota);
 private:
     int hodnota;
 };
```

Předchozím příkladem jsme si naznačili princip definice a deklarace signálů a slotů. V následujícím příkladu si uvedeme situaci přímo z kódu aplikace, kdy je uživateli zobrazen dialog s nastavením aplikace. Uživatel provede nějaké změny v nastavení a my potřebujeme, aby se po potvrzení nastavení jednak změny uložily, ale také, aby se změny ihned promítly v celé aplikaci. Dialog s možnostmi nastavení je zobrazován třídou `SettingsDialog`, jenž je potomkem standardní třídy `QDialog`. Tato třída již obsahuje námi potřebný signál emitovaný při akceptaci dialogu s názvem *accepted* bez parametrů.

Jelikož je dialog nastavení vlastně samostatným oknem, právě ke komunikaci dialogu s hlavním oknem aplikace využijeme již popsaného signálu *accepted* a v hlavním okně definujeme slot *newSettings* bez parametrů. Zde se nabízí možnost, zda si z dialogu nepředat jako parametr signálu do slotu nové nastavení. Návrh je však takový, že nastavení aplikace se načítá z XML souborů při startu aplikace a je drženo v hlavním okně uživatelského rozhraní (hlavní třída celé aplikace). Pokud je pak otvírán dialog pro změnu nastavení, je při tvorbě tohoto dialogu do jeho konstruktoru předán ukazatel na strukturu se stávajícím nastavením. Změny se tak z dialogu do této struktury zapíší ihned, jedná se o parametr předávaný odkazem. Signálem pak pouze dáváme najevo změnu v tomto nastavení, kdy je nutné v případě vypnutí nebo zapnutí logování zobrazit nebo naopak skrýt a vypnout zobrazení logu v hlavním okně. V tomto případě se jedná o jednoduchý problém, který by šel jistě vyřešit i bez použití signálů a slotů, nicméně pokud budeme brát v úvahu možnost rozšiřování aplikace v dalším vývoji, je pak toto připraveno na další položky nastavení a potřebnou reakci na změnu v nich, kdy se budou muset překreslovat a měnit určité prvky uživatelského rozhraní.

Při uložení nastavení v dialogu se tedy nejprve uloží změněná konfigurace do XML souboru (v následujícím kódu nás tato akce tolik nezajímá, proto jsou příkazy vynechány) a emituje se signál. Po emitaci je dialog uzavřen.

```
/* save configuration */
…
if (!configuration->saveToXML()) {
    /* error message */
    ...
} else {
    emit accept();
    close();
}
```

Situace v kódu hlavního okna, jenž má na starosti při stisku odpovídající položky hlavního menu otevření okna, vypadá následovně.

```
SettingsDialog *settings = new SettingsDialog(this, this-
>configuration);

    QObject::connect(settings, SIGNAL(accepted()), this,
SLOT(newSettings()));

    settings->show();
```

Nejprve se vytvoří dialog nastavení s předáním odkazu na strukturu současného nastavení a poté je vidět propojení signálu dialogového okna se slotem neboli metodou, ve které se kontroluje nové nastavení a tomu se hlavní okno přizpůsobuje (příkazy přizpůsobení jsou vynechány).

```
if (this->configuration->isDebugMode()) {
         /* some code */
    } else {
        /* some code */
    }
    Logger::getInstance()->debug("New settings accepted");
```

7.4 Ukládání dat

V aplikaci je tak potřeba ukládat především filtrovací pravidla a nastavení. Soubory s daty jsou ukládány do složky *data* v hlavním adresáři aplikace.

Data jsou ukládána ve formě ***XML*** souborů. XML (*Extensible Markup Language*) řadíme mezi takzvané značkovací jazyky. Jazyk se nezabývá vizuální formou dat, ale vyznačuje se tím, že společně s daty uchovává i jejich konkrétní význam a celkovou strukturu. XML je v současné době velmi využívaným, jeho použití pro ukládání dat přináší výhodu toho, že již samotný XML soubor s filtrovacími pravidly nebo nastavením je i bez kontextu aplikace čitelný a vypovídá sám o sobě. Je tak možné tyto soubory použít i v jiných aplikacích nebo také tyto soubory ručně upravovat bez znalosti přesné struktury (rozmístění parametrů apod.).

Použití XML souborů pro ukládání filtrovacích pravidel bylo zvoleno také z toho důvodu, že lze při použití více XML souborů operovat s více seznamy pravidel. To asi nebude používat běžný uživatel a správce firewallu, ale v případě dalšího

rozšiřování lze touto cestou implementovat do aplikace další funkčnosti v podobě zálohování pravidel či různých šablon.

7.4.1 Čtení a zápis

Čtení a zápis informací do XML souborů zajišťují knihovní třídy (kapitola 7.3.2). Pro operaci s konfigurací se používá třída `Configuration`, pro filtrovací pravidla pak třída s názvem `RulesXML`.

Obě dvě knihovní třídy ve své implementaci používají pro zápis i čtení dat knihovnu ***libxml2***. Tato knihovna poskytuje parser pro načtení XML dokumentu jednorázově do paměti. V paměti je pak vytvořen z jednotlivých značek v dokumentu strom, přičemž výstupem z parseru je základní element neboli kořen tohoto stromu. Každý element v paměti má několik atributů. Kromě názvu elementu a jeho typu (textový element, element představující značku apod.) obsahuje i odkazy na seznam potomků a atributů. Díky těmto atributům lze pak jednoduše načtený strom v paměti procházet od kořene dále a postupně tak data z XML souboru načítat do aplikace.

V případě zápisu je proces obdobný, ale opačný. Nejdříve je nutné vytvořit v paměti strom elementů. Opět je nutné každému uzlu vytvořit jeho jméno a přiřadit mu potomky v podobě dalších elementů a atributů daného elementu. Po vytvoření stromu je jeho kořen předán do patřičné metody (použita metoda *xmlSaveFormatFileEnc*, ve které se dále specifikuje mimo umístění souboru i formát kódování) a data z paměti jsou zapsána do XML souboru.

7.4.2 Filtrovací pravidla

Parametry filtrovacího pravidla jsme specifikovali v kapitole 7.1.1.

Kořenovým elementem XML souboru s pravidly je element s názvem `rules`. V tomto elementu se nachází seznam pravidel, který je uložen v uživatelem specifikovaném pořadí. První uložené pravidlo je tak považováno za první pravidlo i po načtení do aplikace. Pravidlo je specifikováno elementem `rule`.

V elementu jednoho pravidla se vyskytují atributy, které tak v programu představují základní vlastnosti pravidla. Patří sem jméno pravidla (atribut `name`), akce (`action`) a také nepovinný element naznačující, zda se jedná o typ pravidla ovlivňující pouze pakety procházející v módu síťového mostu (atribut `bridge_only`). Pokud tento atribut není u pravidla uveden, znamená to, že je pravidlo druhého typu, tj. ovlivňující všechny pakety (viz kapitola 7.1.2).

Specifikace samotného pravidla se skládá z následujících elementů:

- `description` element pro popis pravidla,
- `in_iface` element pro vstupní port,
- `out_iface` element pro výstupní port,
- `link` element pro parametry linkové vrstvy,
- `net` element pro parametry síťové vrstvy.

Parametry obou vrstev se dále shodně specifikují v elementech:

- `protocol` pro nesený protokol vyšší vrstvy,
- `source` pro specifikaci zdroje,
- `dest` pro specifikaci cíle.

Adresy se dále specifikují elementy `address` a `mask`. V případě masky na síťové vrstvě je v programu hodnota brána jako celé číslo. Naopak na linkové vrstvě je obsah elementu masky interpretován stejně jako HW adresa, tj. jako textový řetězec. Interpretace adresy je v obou případech / vrstvách stejná. Obsah elementu představuje textový řetězec specifického formátu pro každou vrstvu.

Tam, kde je to možné a použito, je potřeba uvést negaci parametru. Negace se v XML souboru promítne atributem u elementu, který je negovaný. Atribut nese označení `negation`. U parametrů filtrovacích pravidel, které nejsou uživatelem negovány, se standardně tato skutečnost neuvádí, tj. neuvedením atributu `negation`.

Na následujícím příkladu je názorně ukázáno uložení dvou pravidel, z nichž je specifikace pravidla ukázána pouze na pravidle prvním. První pravidlo je určeno pouze pro přepínané pakety (mód síťového mostu) a povoluje pakety protokolu IPv4 nesoucí data protokolu TCP z určených síťových a hardwarových adres. IP adresa zdroje musí pocházet ze sítě 192.168.1.0, přičemž cíl je v tomto pravidle specifikován přímo jako zařízení s adresou 192.168.1.150 (maska není a nemusí být v tomto případě uvedena). Vstupním portem musí být v tomto případě port označený jako eth0, výstupním portem naopak nesmí být port eth1. Druhé pravidlo v příkladu je pak použito pro všechny typy paketů.

Je potřeba si zde uvědomit, že tento příklad není použit v praxi. Jedná se pouze o demonstraci pravidla a jeho zápis v XML souboru. Parametry jsou voleny tak, aby příklad ukazoval všechny možnosti volby parametrů a jejich zápis.

```
<?xml version="1.0" encoding="UTF-8"?>
<rules>
  <rule name="jedna" bridge_only="bridge_only" action="ACCEPT">
    <description>První pravidlo pro ukazku</description>
    <in_iface>eth0</in_iface>
    <out_iface negation="negation">eth1</out_iface>
    <link>
      <protocol>IPv4</protocol>
      <source>
        <address>AA:AA:AA:CC:CC:CC</address>
        <mask>AA:00:00:00:00:00</mask>
      </source>
      <dest negation="negation">
        <address> AA:AA:AA:FF:FF:FF </address>
        <mask>AA:00:00:00:00:00</mask>
      </dest>
    </link>
    <net>
      <protocol>tcp</protocol>
      <source>
        <address>192.168.1.0</address>
        <mask>24</mask>
      </source>
      <dest>
        <address>192.168.1.150</address>
        <mask>-1</mask>
      </dest>
    </net>
  </rule>
  <rule name="dva" action="ACCEPT">
     ...
  </rule>
</rules>
```

7.4.3 Nastavení aplikace

Aplikace má velmi jednoduché nastavení. Uživatel může konfigurovat pouze defaultní akci pro filtrování pravidel (viz kapitola 7.1.3), zapnout či vypnout ladící mód (viz kapitola 7.7.3) a zapnout či vypnout zápis pravidel při spuštění aplikace (kapitola 7.5.3).

Kořenem XML souboru s konfigurací je element s názvem `configuration`. Tento element neobsahuje žádné atributy, pouze tři další elementy. Jedním z nich je element `default_action`. Ten obsahuje výchozí akci pro filtr pravidel. Dalšími elementy jsou pak elementy s názvem `debug` a `write_on_start`. Oba dva mohou mít stejný obsah, a to buďto hodnotu `enable` nebo `disable`. Jak je patrné, hodnoty znamenají zapnutý nebo vypnutý ladící mód, respektive zápis pravidel při spuštění aplikace.

Příklad konfiguračního souboru s výchozím pravidlem ACCEPT, zapnutým ladícím módem a zapnutým zápisem pravidel při startu tak vypadá následovně:

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <debug>enable</debug>
  <write_on_start>enable</write_on_start>
  <default_action>ACCEPT</default_action>
</configuration>
```

7.5 Zápis pravidel do systému

Námi navrhovaný firewall pro účely filtrace používá framework Netfilter, jenž je podrobněji rozepsán v kapitolách 6 a 7.2. Dalo by se vlastně říci, že samotná aplikace zajišťuje správu pravidel, která se pak zapisují do kódu jádra operačního systému Linux.

V kapitole 7.2.1 jsme si také uvedli, že jediným doporučovaným způsobem, jak komunikovat z Netfilterem v jádře, je použití příkazů *Xtables-restore*, kterými se standardním vstupem předávají strukturované příkazy obsahující definici pravidel.

Tak, jako v případě ukládání a zpětného načítání dat aplikace (konfigurace, filtrovacích pravidel), i v tomto případě se o proces zápisu pravidel starají knihovní třídy (viz kapitola 7.3.2), konkrétně třída s názvem `RulesPusher`. Důvod je opět jednoduchý. V případě vydaní nové verze frameworku Netfilter včetně očekávaného aplikačního rozhraní bude žádoucí firewall upravit tak, aby využíval nové aplikační rozhraní. Změny se dotknou pouze knihovní třídy. Vše ostatní zůstane v aplikaci zachováno.

Ve všech námi použitých komponentách (ebtables, iptables a ip6tables) Netfilteru jde o stejný princip zápisu. Je nejprve nutné vyexportovat všechna pravidla včetně výchozí akce do souborů. Soubory se používají tři, pro každou použitou komponentu frameworku Netfilter jeden. Poté je spuštěn příkaz operačního systému *iptables-restore*, respektive *ebtables-restore* nebo *ip6tables-restore*, do kterého je jako standardní vstup nasměrován právě daný vytvořený soubor (např. *iptables-restore < ipfile*). Po úspěšném provedení příkazu, kdy si aplikace samozřejmě hlídá návratovou hodnotu, jsou vygenerované dočasné soubory pro komponenty Netfilteru opět smazány.

7.5.1 Mapování pravidla na příkaz

Pro zjednodušení budou dále v textu probírány pouze komponenty iptables a ebtables. V případě ip6tables jsou principy shodné s iptables.

Při exportu je nutné filtrovací pravidlo převést do textové řádkové formy, které rozumí a kterou zpracuje příkaz *Xtables* (Xtables-restore vlastně čte standardní vstup a pro každý řádek volá příkaz Xtables).

Možné argumenty příkazu iptables, respektive ebtables či ip6tables je možné nalézt na manuálových stránkách [16], [17] a [18], kde jsou také jednotlivé argumenty dobře popsány. V aplikaci se pak parametry filtrovacích pravidel „namapovali“ na argumenty příkazů dle tabulky 7.1. Připomeňme si, že příkaz komponenty iptables není použit (z důvodu nedostatečné podpory) pro parametry linkové vrstvy, a proto argumenty v příslušných buňkách tabulky uvedeny nejsou.

Parametr filtrovacího pravidla	Argument příkazu ebtables	Argument příkazu iptables
vstupní port	-i	-i
výstupní port	-o	-o
nesený protokol linkovou vrstvou	-p	
adresa zdroje na linkové vrstvě (zdrojová MAC / HW adresa)	-s	
adresa cíle na linkové vrstvě (cílová MAC / HW adresa)	-d	
nesený protokol síťovou vrstvou	--ip-protocol	-p
adresa zdroje síťové vrstvy (zdrojová IP adresa)	--ip-source	-s
adresa cíle síťové vrstvy (cílová IP adresa)	--ip-dest	-d
akce	-j	-j

Tabulka 7.1 Mapování filtrovacího pravidla na argumenty příkazů ebtables a iptables

Všechny argumenty příkazů z tabulky jsou v programu nadefinovány jako konstanty knihovní třídy, která se o zápis pravidel do systému stará (`RulesPusher`). Opět tento přístup napomáhá při realizaci změn, kdy je třeba opravit pouze definici konstanty, která se používá nadále v kódu, a změny se tak dostanou do celého kódu. Kromě konstant představujících argumenty příkazů uvedené v tabulce 7.1 jsou nadefinované další argumenty.

Celý příkaz má svou strukturu. Kromě parametrů pravidla je nutné specifikovat také fakt, že se pravidlo má do Netfilteru přidat. Dále je nutné určit cílovou tabulku a její řetězec pravidel. Jaké tabulky a jaké řetězce se používají, jsme si již vysvětlili v kapitole 7.2.2, zde pouze uvádíme fakt, že i tyto skutečnosti jsou nutné v generovaném příkazu uvést (viz manuálové stránky [16] a [17]), stejně jako případnou negaci jednotlivých parametrů filtrovacího pravidla.

7.5.2 Proces generování

V této části si názorně naznačíme, jak jsou z filtrovacích pravidel v aplikaci vygenerovány soubory představující vstupy do příkazů námi použitých komponent Netfilteru ebtables a iptables.

Je nutné zde uvést, že ačkoliv jsou příkazy komponent stejné a člověk by asi očekával stejnou syntaxi vstupních souborů, nejspíš tím, že se o komponentu

ebtables stará jiná skupina vývojářů, než v případě komponenty iptables, jsou požadavky na vstupní soubory odlišné. Nejedná se však o žádné velké odlišnosti, jde spíše o detaily (umístění značky pro negaci parametru filtrovacího pravidla, obsah hlavičky a patičky generovaného souboru). Nicméně i detaily způsobují to, že je nutné v kódu použít pro generování dvě odlišné skupiny metod. Jedna pro generování vstupu do iptables a ip6tables, druhou pak pro generování vstupu do ebtables.

Situace je tedy taková, že se vždy generuje více výstupních souborů (pro každou z komponent). Nejprve je do každého výstupu nutné vložit jeho hlavičku. V hlavičce se nastavují výchozí akce pro každý řetězec pravidel námi použité tabulky FILTER (viz kapitola 7.2.2). Kromě hlavičky obsahuje výstup také takzvanou patičku, avšak pouze výstup do komponenty iptables. Patička obsahuje příkaz COMMIT jímž jsou změny do frameworku zapsány a aktivovány „najednou“ (stejný princip jako například u databází).

Mezi hlavičkou a patičkou výstupů dochází k samotnému generování pravidel. Filtrovací pravidla jsou předána do algoritmu v takovém pořadí, jak je uživatel v aplikaci seřadil, a sekvenčně zpracovávána. Každé pravidlo se standardně zapisuje do řetězce FORWARD komponenty ebtables. Dále se kontroluje, zda je pravidlo možné zařadit do řetězce INPUT nebo OUTPUT (viz kapitola 7.2.2). Při zápisu je pak pravidlo „převedeno“ do textové podoby příkazu a přidáno do výstupu. Vždy je dodrženo pořadí pravidel z aplikace.

Algoritmus je naznačen v následujícím odstavci psaném v pseudokódu.

```
vystup = hlavička;

pro každé pravidlo proveď {

        pokud je pravidlo možné zapsat do řetězce INPUT {

            vystup += pravidlo(pro řetězec INPUT);

        }

        vystup += pravidlo(pro řetězec FORWARD);

        pokud je pravidlo možné zapsat do řetězce OUTPUT {

            vystup += pravidlo(pro řetězec OUTPUT);

        }

    }

vystup += patička;
```

Jak je vidět z příkladu, pravidlo je do textové podoby příkazu převáděno vždy stejnou metodou, jejímž argumentem je cílový řetězec, který je nutné v příkazu specifikovat a který je jediným argumentem, který se může v metodě změnit na základě vnějších vlivů / okolností.

Je nutno dodat, že do výstupu pro komponentu iptables se dostávají pouze pravidla, která jsou typu ovlivňujícího všechny pakety (jak je popsáno v kapitole 7.1.2). Naopak ve výstupu pro komponentu ebtables najdeme všechna pravidla z aplikace. Výše uvedený příklad je tak příklad výstupu pro ebtables. V případě komponenty iptables je nutno přidat ještě jednu podmínku.

```
vystup = hlavička;

pro každé pravidlo proveď {

        pokud není pravidlo určeno pouze pro bridge pakety {

            pokud je pravidlo možné zapsat do řetězce INPUT {

                vystup += pravidlo(pro řetězec INPUT);

            }

            vystup += pravidlo(pro řetězec FORWARD);

            pokud je pravidlo možné zapsat do řetězce OUTPUT {

                vystup += pravidlo(pro řetězec OUTPUT);

            }

        }

    }

vystup += patička;
```

7.5.3 Zápis pravidel při startu aplikace

Restart zařízení, na kterém firewall běží, znamená jeden problém. Filtrovací pravidla totiž v jádře operačního systému restart „nepřežijí“ a po startu je operační systém bez pravidel. Standardně si tento problém řeší administrátoři sami a například v konfiguracích síťových rozhraní používají právě příkazy Xtables-restore, do kterých směřují vlastní textové soubory z pravidly podobné těm, které generuje tento firewall.

Navrhovaný firewall tento problém řeší následujícím způsobem. Vždy při startu aplikace načte uložená pravidla z XML souboru, a pokud je v nastavení povolen zápis pravidel při startu, všechna pravidla hned také zapíše do jádra operačního systému. Předpokládá se, že uživatel bude firewall po startu zařízení spouštět (manuálně nebo automaticky) a nemusí tak dále v systému řešit tento problém zvlášť.

Zápis pravidel při startu lze také vypnout v nastavení aplikace. Toto se nedoporučuje, nicméně jako možnost pro určité ladění firewallu či nastavení je tato možnost v nastavení přístupná.

7.6 Získávání statistik

Již v zadání je uvedeno, že výsledný softwarový firewall má umět kromě vlastního filtrování zobrazovat i svůj stav. Stavem u filtrovacího firewallu rozumíme především soubor statistik jednotlivých pravidel, který u každého pravidla uvádí počet paketů nebo bytů, které byly pravidlem „ovlivněny".

Statistiky nám tak zprostředkovávají jakousi zpětnou vazbu. Uživatel vidí, jaké pravidlo se jak moc používá a ovlivňuje síťový provoz. Statistiku lze také použít při ladění konfigurace, kdy můžeme zjistit, proč se určité pakety skrze filtr propouští či naopak nedostávají. V neposlední řadě pak statistiky vypovídají i o struktuře síťového provozu a lze je použít k jeho monitorování. Pokud například chceme monitorovat přístup do určitého segmentu sítě, stačí si nadefinovat jedno pravidlo, ve kterém můžeme již standardně povolenou komunikaci explicitně povolit. Díky statistikám, kolikrát bylo pravidlo použito, pak můžeme snadno zajistit lehkou formu monitoringu, i když pro účel monitorování se nabízí řešení určená přímo k tomuto účelu a toto není primárním účelem našeho filtrovacího firewallu. Pro nás statistiky představují právě zpětnou vazbu celé aplikace.

Počet paketů či bytů, které byly filtrovacím pravidlem „ovlivněny" počítá již v jádru operačního systému Linux framework Netfilter. Před námi tak stojí pouze problém, jak tyto počty dostat z jádra do aplikace a uživateli je zobrazit.

Získávání statistik je implementováno opět v knihovní třídě, tentokrát s názvem `RulesStatsLoader`. Jako v předešlých případech (zápis pravidel nebo ukládání dat) se opět jedná o interakci s vnějším prostředím aplikace, které se může změnit, a proto je použita knihovní třída, ve které by se změna vnějšího prostředí projevila (nikoliv napříč celou aplikací).

7.6.1 Použití komentářů

Jako jedno z možných řešení se nabízí u každého pravidla použít komentář. Ten představuje jednoznačný identifikátor pravidla. Identifikátor je vypočten hashovaní funkcí ze jména pravidla (zadává uživatel) algoritmem *Jenkins One-at-a-Time Hash* [19]. Algoritmus je následující (uvedeno v jazyce C++):

```
uint32_t FilterRule::hashString(const char *s) {

    uint32_t hash = 0;

    for (; *s; ++s) {

        hash += *s;

        hash += (hash << 10);

        hash ^= (hash >> 6);

    }

    hash += (hash << 3);

    hash ^= (hash >> 11);
```

```
    hash += (hash << 15);

    return hash;

}
```

Takto vypočtený identifikátor je pak použit jako komentář pravidla při jeho zápisu do systému (kapitola 7.5).

Získání statistik pravidel pak probíhá následujícím způsobem. Aplikace zavolá příkaz iptables, který slouží ke správě stejnojmenné komponenty frameworku Netfilter. Pokud se použijí správné parametry (konkrétní podoba příkaze je *iptables –L –v –x -n*) [17], příkaz na standardní výstup vypíše seznam pravidel ve všech řetězcích tabulky FILTER včetně komentáře a počtu „ovlivněných“ paketů či bytů. Výstup příkazu je zobrazen na obrázku 7.4.

```
Chain INPUT (policy ACCEPT 4405 packets, 5456K bytes)
 pkts bytes target     prot opt in     out     source               destination
  167 25238 ACCEPT     udp  --  any    any     anywhere             anywhere            /* 1092125516 */

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
    0     0 ACCEPT     udp  --  any    any     anywhere             anywhere            /* 1092125516 */

Chain OUTPUT (policy ACCEPT 3448 packets, 334K bytes)
 pkts bytes target     prot opt in     out     source               destination
  166 10521 ACCEPT     udp  --  any    any     anywhere             anywhere            /* 1092125516 */
```

Obrázek 7.4 Příklad výstupu se statistikami komponenty iptables

Jak je vidět z výše uvedeného příkladu výstupu, každé pravidlo je obsaženo na jednom řádku. V aplikaci (knihovní třídě zajišťující sběr statistik) tak postupně pro každé pravidlo aplikace projedeme odchycený výstup spuštěného příkazu řádek po řádku, přičemž kontrolujeme každý řádek, zda se v něm nenachází identifikátor pravidla jako komentář. Pokud řádek identifikátor pravidla obsahuje, zjistíme z tohoto řádku statistické údaje uvedené v prvním a druhém sloupci a připočteme je ke stávajícím údajům v pravidle.

Záměrně a zcela správně je uvedeno slovo *připočteme*. Před každým sběrem se totiž statistiky pravidel v aplikaci vynulují a poté se počítá součet údajů uvedených na výstupu. Musíme si uvědomit, že se pravidlo definované v aplikaci může dostat do více řetězců, jak je popsáno v kapitole 7.2.2, a tudíž při zpětném sběru statistik musíme jednotlivé dílčí statistiky ze všech řetězců sečíst dohromady a tento součet u pravidla v aplikaci uvést.

Po celou dobu při vysvětlování komentářů jsme používali pouze komponentu iptables. Důvod je takový, že druhá námi použitá komponenta ebtables komentáře pravidel nepodporuje. Zde musí statistiky získat jiným způsobem. V případě komponenty ip6tables je princip získávání statistik totožný jako u iptables.

7.6.2 Použití pořadí pravidel

Z důvodu absence podpory komentářů filtrovacích pravidel v komponentě ebtables musíme statistická data z této komponenty frameworku Netfilter získat jiným způsobem.

Jako řešení se nabízí způsob obdobný komentářům, kdy je opět při použití příkazu tentokrát ebtables se správnými argumenty (*ebtables –L --Lc --Ln*) [16] zajištěn výstup pravidel použitých v této komponentě, znázorněný na obrázku 7.5. Při výstupu se u každého pravidla zobrazuje také jeho pořadové číslo v řetězci a to je pro nás znak pravidla, podle kterého můžeme pravidla z komponenty ebtables identifikovat.

V aplikaci máme totiž pravidla seřazená a přesně víme, v jakém pořadí se do jakých řetězců zapisují. Volba řetězců je vysvětlena v kapitole 7.2.2. Při sběru statistik tak procházíme stejně sekvenčním způsobem jako v případě zápisu jednotlivá pravidla, přičemž si interně v algoritmu uchováváme počítadlo zapsaných pravidel pro každý řetězec zvlášť. Na základě definice pravidla v aplikaci tedy víme, do jakých řetězců se pravidlo zapisuje / zapsalo. Dále tedy známe čítač pro každý řetězec a díky tomu můžeme ve výstupu každého řetězce najít řádek s odpovídajícím číslem pravidla a tento řádek pak dalším algoritmem zpracovat.

V další fázi, pokud již máme daný řádek výstupu, tento řádek analyzujeme. Je pravidlem, že statistické informace jsou odděleny od pravidla znakem mezery a dále jsou tyto informace popsány řetězci *pcntl* a *bcntl*. Díky těmto „ukazatelům“ již dokážeme informace z řádku extrahovat a načíst do aplikace.

```
Bridge table: filter

Bridge chain: OUTPUT, entries: 3, policy: ACCEPT
1. -p IPv4 -s 0:0:0:0:0:0/0:0:0:0:0:0 -d ! 0:0:0:0:0:0/0:0:0:0:0:0 -o ! eth0 --ip-src 192.168.1.0/24 --ip-
2. -p IPv4 --ip-proto udp -j ACCEPT , pcnt = 0 -- bcnt = 0
3. -p IPv6 --ip6-proto udp -j ACCEPT , pcnt = 0 -- bcnt = 0
```

Obrázek 7.5 Příklad výstupu se statistikami komponenty ebtables pro jeden řetězec

Nesmíme opomenout zmínit, že je nutné spouštět příkaz pro získání statistik z každého řetězce tabulky FILTER zvlášť, jelikož každý řetězec má svoje číslování pravidel. Proto si musíme „odchytit“ výstupy z příkazu volaného pro každý řetězec (ve skutečnosti tedy voláme třikrát příkaz *ebtables –L <řetězec> --Lc --Ln*) a poté tyto celkem tři výstupy (řetězce INPUT, FORWARD a OUTPUT) procházet zvlášť, přičemž i v aplikaci je nutné si držet čítače pravidel pro každý výstup.

7.7 Logování

Nejen pro účely ladění je vhodné, aby aplikace uměla v průběhu její činnosti sbírat informativní zprávy a zapisovat je někam do souboru, případně i rovnou zobrazovat uživateli v grafickém rozhraní. Tuto funkčnost označujeme za log nebo logování.

V navrhovaném firewallu bylo logování implementováno přesně, jak je popsáno v předešlém odstavci. Různé části aplikace při svých činnostech používají ladící log (tzv. *debug log*). Ten je určen, jak jeho název již napovídá, především pro účely ladění, i když jeho použití není nijak omezeno. Ladící log obsahuje informativní výpisy o různých akcích, které byly v danou chvíli aplikací provedeny. Jako příklad můžeme například uvést zápis pravidel do systému Netfilter, načítání statistik z Netfilteru nebo ukládání dat do XML souborů. To jsou pro nás nejdůležitější akce,

které bychom si měli „hlídat“. Zprávy vkládané do logu se uživateli zobrazují na hlavním okně aplikace a také se zapisují do souboru *data/debug.log*.

Při startu aplikace je vždy log vymazán. Můžeme tedy říci, že v souboru a samozřejmě v okně logu se nachází vždy zprávy z posledního běhu firewallu.

7.7.1 Log jako singleton

Funkci logu plní jedna z knihovních tříd. V tomto případě se jedná o třídu s názvem `Logger`. Jelikož požadujeme použití této třídy napříč celým kódem aplikace, není ideálním nápadem si předávat v různých objektech instanci této třídy. Také vytvářením instance při potřebě se nejeví jako ideální nápad. Proto bylo zvoleno řešení v podobě návrhového vzoru *singleton*.

Singleton (česky jedináček) řeší problém, kdy je potřeba v celé aplikaci jedné instance. Třída tak má ve skutečnosti sama v sobě jednu instanci, ke které se lze dostat statickou metodou této třídy (pojmenovanou jako *getInstance*, bez parametrů). Pokud vnitřní instance nebyla vytvořena (je NULL), je vytvořena právě zde v této metodě. Pokud však již instance vytvořena byla některým předešlým (prvním) voláním statické metody, je pouze předán odkaz na vytvořenou instanci. Implementaci znázorňuje následný kód (C++).

```
class Logger {

private:

    static Logger *logger;

public:

    static Logger *getInstance() {

        if (!logger) {

            logger = new Logger();

        }

        return logger;

    }


}
```

Třída samozřejmě neobsahuje jen zde uvedené atributy a metody. Zde jsou uvedeny jen ty, které jsou důležité z hlediska návrhového vzoru singleton. Další veřejné metody pak zajišťují zápis zpráv do logu, či jeho smazání. Dále jsou v definici použité prvky pro realizaci komunikace mechanismem signálů a slotů (viz kapitola 7.7.2). Také je potřeba uvést, že pro názornost privátních a veřejných metod návrhového vzoru singleton je v příkladu kód spojen oproti skutečnosti, kde

je definice třídy oddělena v hlavičkovém souboru `Logger.h` a souboru `Logger.cpp`.

Pokud si řekneme, že třída pro zalogování zprávy obsahuje metodu *message* s parametrem představujícím právě zprávu k zalogování, pak tuto zprávu vložíme do logu příkazem:

```
Logger::getInstance()->message("Zprava k zalogovani");.
```

7.7.2 Zobrazení logu

Jak jsme již psali v úvodu této kapitoly věnující se logu (kapitola 7.7 a její podkapitoly), zprávy se kromě ukládání do souboru také zobrazují uživateli na hlavní obrazovce. Pro tento účel byla vytvořena třída ze skupiny *view* (viz 7.3.1) s názvem `LogView`. Třída představuje potomka třídy z frameworku *Qt* – `QTextEdit`, jenž se používá pro text o více řádcích. V potomkovi jsme však tuto třídu rozšířili o jednu veřejnou metodu, vlastně slot.

Pro tento případ komunikace, tedy komunikace mezi knihovní třídou logu a jejím zobrazením, byl použit mechanismus signálů a slotů. Mechanismus je podrobněji popsán v kapitole 7.3.3.

Při vytváření hlavního okna aplikace, kdy již v kódu byla třída logu použita k uložení nějakých zpráv (například při načtení uložené konfigurace a filtrovacích pravidel), je vytvořeno i okno logu. Právě při tomto vytváření se zaregistruje k signálu knihovní třídy logu slot z třídy představující výstupní okno logu v grafickém uživatelském rozhraní. Při vyváření okna se do něj ještě doplní již uložené zprávy do logu (ty ohledně načtené konfigurace apod.). Dále pak přidávání zpráv probíhá tím způsobem, že kdekoliv v aplikaci je volána metoda knihovní třídy pro vložení zprávy do logu, je touto metodou emitován signál s parametrem obsahujícím zprávu. Signál je zachycen oknem v uživatelském rozhraní a zpráva je tak v okně zobrazena.

7.7.3 Zapnutí / vypnutí

Logování je věc, která nemusí být uživatelem vždy žádoucí, a proto ji lze v nastavení aplikace vypnout. V tom případě je z hlavního okna aplikace odstraněno okno s výpisy logu a v knihovní třídě logu zablokovány všechny signály. Tím na log přestanou reagovat všechny připojené komponenty grafického uživatelského rozhraní. V současnosti se jedná pouze o jednu komponentu, která je navíc z hlavního okna odstraněna, nicméně pro případ rozvoje aplikace či změně v podobě neodstraňování okna logu, ale jeho „zneaktivnění“, se signály blokují.

Pokud pak uživatel log opět zapne, jsou signály odblokovány a okno logu opět zobrazeno.

7.8 Model seznamu pravidel

Z hlediska zvolené architektury (viz kapitola 7.3) je nutné pro seznam filtrovacích pravidel v aplikaci definovat modelovou třídu. V této třídě je pak zapotřebí definovat metody rozhraní, přes které tato modelová třída bude komunikovat s třídami v grafickém uživatelském rozhraní.

Modelová třída seznamu pravidel se jmenuje `FilterRulesModel` a je potomkem třídy z knihovny Qt – `QAbstractListModel`. Toto dědění nám napomáhá v lepší konkretizaci modelové třídy, kdy je řečeno, že půjde o model představující nějaký seznam. Použitím abstraktní třídy pro model seznamu jsme si navíc ušetřili práci s definicemi všech atributů a metod a v implementaci tak budeme definovat opravdu jen ty nejdůležitější věci.

Vnitřně v této třídě si uchováváme seznam pravidel v instanci objektu `QList`. Tento seznam je možné parametrizovat na libovolný typ objektů, které uchovává. Pro nás je tímto typem modelová třída filtrovacího pravidla, jenž je popsána v kapitole 7.9. Nad tímto seznamem pak definujeme metody rozhraní pro architekturu model / view a také další potřebné metody.

7.8.1 Rozhraní model / view

Aby třídy z grafického uživatelského rozhraní věděli, kolik prvků je v seznamu obsazeno, bylo potřebné implementovat metodu

```
int rowCount(const QModelIndex& parent) const
```

vracející počet prvků v seznamu. Parametrem metody je index, jenž představuje úroveň zanoření (pro představu se tento parametr využívá v modelech pro různé stromy), nicméně v naší implementaci jsou si všechny prvky v seznamu (filtrovací pravidla) rovny a mají tak stejný nulový index. Z tohoto důvodu na tento parametr není brán zřetel a vždy se vrací počet prvků.

```
QVariant FilterRulesModel::data(const QModelIndex& index, int
role) const
```

je další z metod rozhraní. V tomto případě se jedná o poskytnutí dat, která se mají na daném indexu v dané roli zobrazit. Parametr index v sobě nese jak řádkový index, tak také index sloupcový. V našem případě opět bereme v potaz pouze index řádku. Role odlišuje různé přístupy k modelu. Pro nás je důležitá role specifikovaná jako *Qt::DisplayRole*. Pouze tehdy vrátíme jméno pravidla na daném řádku v seznamu. Tímto tak docílíme toho, že v uživatelském rozhraní v komponentě pro zobrazení seznamu pravidel, uživatel uvidí seznam názvů jednotlivých filtrovacích pravidel.

Další nutnou metodou rozhraní je specifikace hlaviček neboli názvů jednotlivých indexů.

```
QVariant FilterRulesModel::headerData(int section,
Qt::Orientation orientation, int role) const
```

Toto si lze nejlépe představit v modelech implementující tabulky, kde je dobré uživateli uvést názvy jednotlivých sloupců a řádků. Navíc model může reagovat na orientaci zobrazovací třídy (použitelné například v různých mobilních zařízeních). Ta tak do modelu předává této metodě informaci, zda se jedná o vertikální nebo horizontální zobrazení. V našem případě se ve vrstvě grafického rozhraní používá třída bez popisků položek, proto i tato metoda není nijak důležitá. Je však potřeba ji nadefinovat, a proto byl zvolen standardní způsob, kdy v případě horizontální

orientace vrátíme popisek *Column* + číslo sekce (indexu) a v případě vertikální orientace vrátíme popisek *Row* + číslo sekce.

Posledním nutným úkonem je pro každou položku modelu na daném indexu definovat, co může, respektive co nesmí. K tomuto účelu slouží příznaky – anglicky *flags*. Je nutné tedy implementovat metodu:

```
Qt::ItemFlags FilterRulesModel::flags(const QModelIndex& index)
const.
```

V implementaci si pro daný index zjistíme standardní příznaky z třídy předchůdce, jenž nám zajistí správné zobrazení prvků bez možnosti editace. K těmto standardním příznakům přidáme z důvodů funkce *drag and drop* (kapitola 7.8.2) následující:

- v případě validního indexu přidáme příznak pro povolení akce drag,
- v případě nevalidního indexu přidáme příznak pro povolení akce drop.

7.8.2 Drag and drop podpora

U filtrovacích pravidel záleží na pořadí. V grafickém uživatelském rozhraní je umožněno pravidla řadit v jejich zobrazovací třídě za pomoci technologie *drag and drop*. Samozřejmě tuto technologii musí také podporovat i model.

Je tedy potřeba implementovat další metody rozhraní. Těmi jsou také metody pro vložení a odebrání řádku. Možná někdo namítne, že se musí tyto metody implementovat standardně, ale to není pravda. Tyto metody slouží pro odebrání, respektive přidání řádku vyvolané právě zobrazovací třídou z grafického rozhraní, a proto je jejich potřeba implementace uvedena až zde. Pokud bychom nepoužili technologii drag and drop, kde právě dochází při přesunu řádku k těmto akcím, přidávali bychom řádky (nebo odebírali) formou stisknutí nějakého tlačítka v grafickém rozhraní, které by vyvolalo jinou metodu modelu, data modelu by se upravila a model by informoval třídu o aktualizaci dat.

```
bool FilterRulesModel::removeRows(int position, int rows, const
QModelIndex &parent)

bool FilterRulesModel::insertRows(int position, int rows, const
QModelIndex &parent)
```

V metodách se parametry specifikuje pozice řádku, počet vybraných řádků (standardně 1) a rodičovský index, který v naší implementaci modelu nepoužijeme.

V modelu musíme také zavést podporu pro drag and drop. To uděláme jednak příznaky, jak je uvedeno v kapitole 7.8.1, a také implementací metody

```
Qt::DropActions FilterRulesModel::supportedDropActions() const,
```

ve které specifikujeme, že podporujeme akce typu přesunu (*Qt::CopyAction*) nebo kopírování (*Qt::MoveAction*).

K realizaci drag and drop potřebujeme pracovat s MIME daty. Těmi určíme, o jaký druh dat se jedná. Jelikož v našem případě realizujeme drag and drop pouze v seznamu pravidel, kdy nepovolujeme „dropnout“ data z jiných aplikací nebo jejich částí a podobně ani nepočítáme, že „dragnutá“ data bude chtít uživatel použít jinde. Definujeme si tedy vlastní MIME typ filtrovacích pravidel - *application/vnd.filterrule.list*. Pokud tedy uživatel myší vezme filtrovací pravidlo ze seznamu, je toto pravidlo „převedeno“ do pole bytů, přičemž je pravidlo ze seznamu vyjmuto. Pole bytů je pak uloženo ve struktuře specifikující MIME typ a samotný obsah, tedy naše pole. Tuto činnost zajišťuje metoda

```
QMimeData *FilterRulesModel::mimeData(const QModelIndexList
&indexes) const.
```

Na druhé straně je pak pravidlo vkládáno do modelu. Toto se děje tak, že uživatel přesouvané pravidlo někam umístí a uvolní tlačítko myši. Zde je spuštěna metoda

```
bool FilterRulesModel::dropMimeData(const QMimeData* data,
Qt::DropAction action, int row, int column, const QModelIndex&
parent).
```

V ní se kontroluje typ MIME dat a model přijme pouze námi definovaný typ pro filtrovací pravidla. Tím zabráníme vložení jiných dat. Data s naším MIME typem se tedy převedou zpět z pole bytů na objekt filtrovacího pravidla a vloží na příslušném řádku do modelu.

Poslední nutnou metodou pro realizaci mechanismu drag and drop bylo nutné implementovat metodu

```
QStringList FilterRulesModel::mimeTypes() const,
```

která slouží pouze pro zjištění MIME typů, které model podporuje. V našem případě pouze náš vlastní typ *application/vnd.filterrule.list*.

7.8.3 Ostatní

Mezi ostatní metody modelové třídy seznamu filtrovacích pravidel metody operující nad vlastním seznamem:

- získání jednoho pravidla na konkrétní pozici,
- získání celého seznamu pravidel,
- načtení pravidel ze souboru (používá se knihovní třída),
- vložení nového pravidla na konkrétní pozici,
- smazaní pravidla na konkrétní pozici,
- duplikace pravidla na konkrétní pozici,
- indikace, zda je seznam prázdný či nikoliv.

Metody se od těch, které jsou uvedeny v kapitole 7.8.1 a 7.8.2, liší tím, že tyto metody se používají primárně v aplikaci pro operace nad seznamem pravidel, zatímco metody rozhraní jsou volány pouze zobrazovacími třídami grafického rozhraní.

7.9 Model filtrovacího pravidla

Modelová třída filtrovacího pravidla obsahuje především atributy pravidla uvedené v kapitole 7.1.1 společně s metodami pro jejich získání a nastavení (tzv. *getry* a *setry*).

Kromě těchto atributů a metod pak dále ve třídě nalezneme:

- metodu pro zápis pravidla do datového streamu,
- metodu pro načtení pravidla z datového streamu,
- metodu indikující možný zápis do INPUT řetězce,
- metodu indikující možný zápis do OUTPUT řetězce,
- metodu pro výpočet jedinečného identifikátoru pravidla.

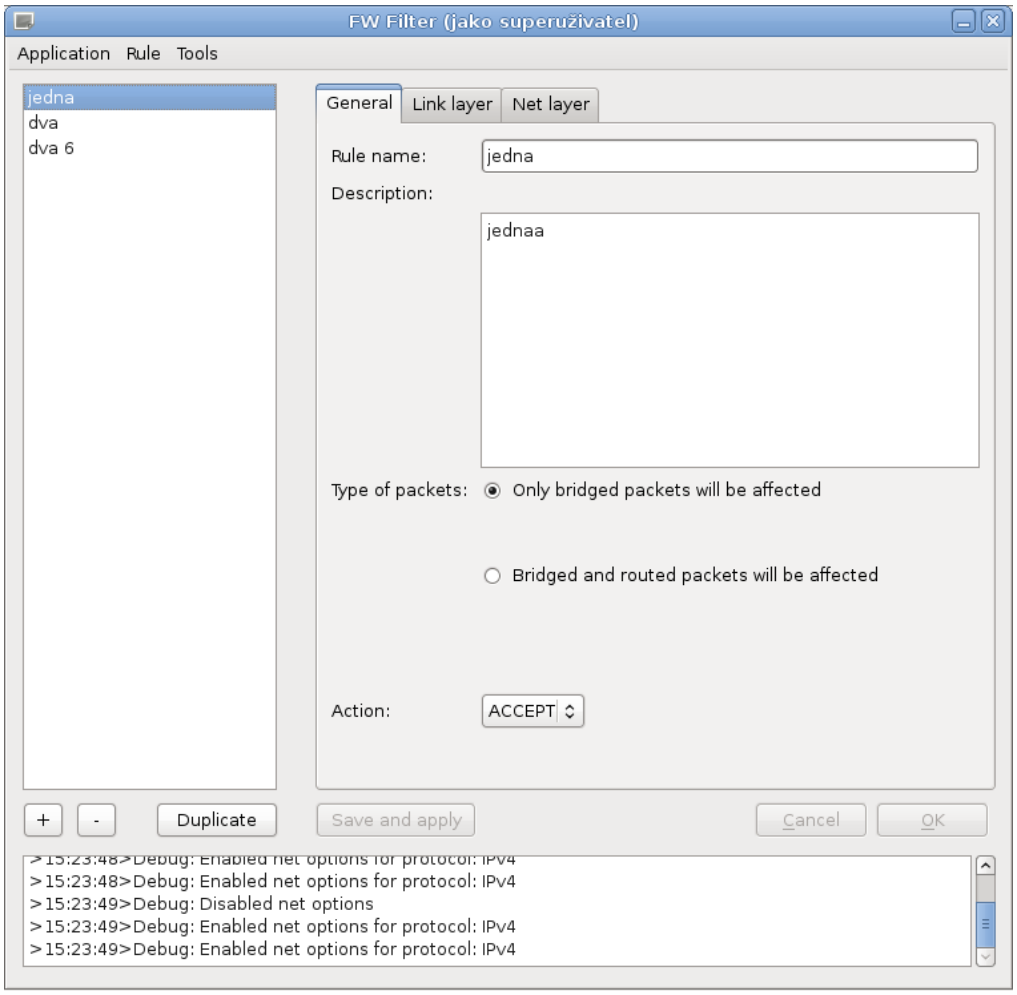
Obě dvě metody operující nad datovým streamem se používají z důvodu realizace mechanismu drag and drop v modelu seznamu pravidel.

Na základě hodnot atributů pak filtrovací pravidlo obsahuje metody, kterými v případě dotazu dokáže říci, zda je určeno pro nějaký rozšiřující řetězec pravidel v frameworku Netfilter (popsáno v kapitole 7.2.2).

Identifikátor pravidla se používá z důvodu získávání statistik. Detailnější popis včetně naznačeného algoritmu je uveden v kapitole 7.5.3.

7.10 Grafické uživatelské rozhraní

Grafické uživatelské prostředí aplikace je koncipováno do jednoho okna, ve kterém probíhá téměř veškerá potřebná činnost.



Obrázek 7.6 Hlavní okno grafického rozhraní

V následujícím textu si postupně popíšeme jednotlivé komponenty grafického rozhraní a na závěr hlavní okno znázorněné na obrázku 7.6.

7.10.1 Seznam pravidel

Samotný seznam pravidel je realizován již v modelové třídě popsané v kapitole 7.8. Zde, v grafickém rozhraní, pouze použijeme standardní třídu z Qt – `QListView`. Tuto třídu přitom není potřeba nijak upravit. Pouze nastavíme základní atributy, jako:

- rozměry,
- povolení přijímat události typu drop,
- povolení událostí typu drag,
- omezení událostí typu drag and drop pouze na vnitřní mód, tj. žádná interakce s ostatními komponentami v rozhraní,
- nastavení defaultní akce typu drop jako přesun,
- povolení přesunu položek.

Jak je patrné, téměř všechna nastavení se týkají správného nastavení mechanismu *drag and drop*, kterým je realizováno řazení pravidel. Defaultní akce pro akci drop

představuje interakci komponenty, kdy uživatel v našem případě přesouvá pravidlo z jedné pozice do druhé, a my požadujeme, aby kurzor ukazoval přesun, ne kopírování položky, tj. u kurzoru nebylo zobrazeno znaménko +, které ve většině případů identifikuje režim kopírování položky.

7.10.2 Zobrazení logu

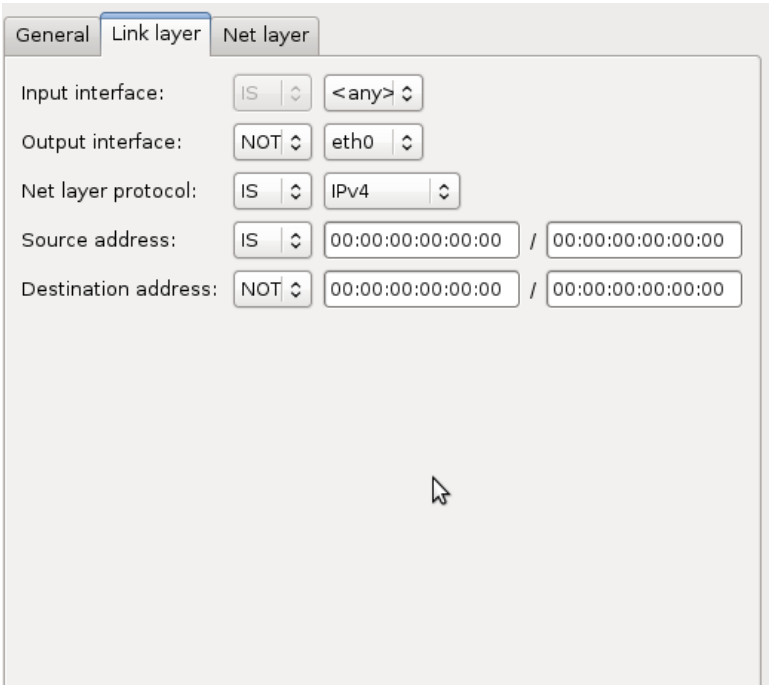
Celá kapitola 7.7 se věnuje logování, zde si jen popíšeme komponentu umístěnou na hlavním okně v případě, že je logování zapnuté. Pouze tehdy je log na hlavním okně umístěn a viditelný.

Pro zobrazení logu byla použita vlastní třída `LogView`, která rozšiřuje svého předka `QTextView`. Rozšíření se týká veřejného slotu pro zobrazení nové zprávy mechanismem signálů a slotů (kapitola 7.3.3).

Každá zpráva logu je zobrazena na vlastní řádce společně s časem, kdy zpráva vznikla. Nejnovější zprávy se zobrazují nejníže v okně, přičemž při přidání nové zprávy se okno automaticky posune. K posunu a prohlížení zpráv slouží vertikální posuvník.

7.10.3 Editační panel pravidla

Jak je znázorněno na obrázku 7.6, hlavní okno obsahuje dvě hlavní části. V pravé se nachází seznam pravidel, v levé pak editační panel. V tomto panelu se vždy zobrazuje jedno vybrané pravidlo ze seznamu.



Obrázek 7.7 Editační panel

Editační panel je rozdělen do tří záložek, jak je znázorněno na obrázku 7.7. Na každé záložce nalezneme prvky, které spolu určitým způsobem souvisí. Jedná se o obecné možnosti pravidla (jméno, popis, typ pravidla, akce), možnosti linkové

vrstvy a možnosti vrstvy síťové. Rozdělením do záložek dosáhneme lepší přehlednosti, než kdyby byly jednotlivé prvky na jednom panelu. Dále pokud je specifikací pravidla nutné nezpřístupnit některou sadu možností (jedná se především o možnosti síťové vrstvy při volbě neanalyzovatelného protokolu neseného linkovou vrstvou), jednoduše je záložka znepřístupněna a uživatel lépe a především ihned vidí a chápe danou situaci.

Jako základ pro editační panel byla použita třída `QTabWidget` představující právě záložkové panely. Nad ní pak byl vytvořen potomek s názvem `RuleEditWidget`. Ten pak v sobě obsahuje všechny prvky grafické prvky nutné k umožnění editace pravidla, které také rozděluje do správných záložek.

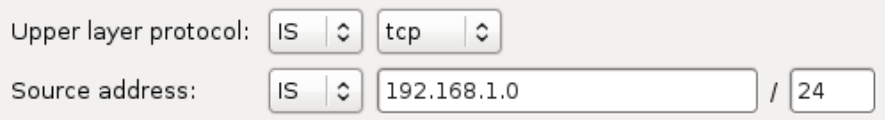
Koncept editace je následující. Uživatel vybere ze seznamu pravidel jedno konkrétní, to je mechanismem signálů a slotů (kapitola 7.3.3) indikováno právě editačnímu panelu, který si vybere ze seznamu pravidel ono vybrané (v signálu se předává index vybraného pravidla) a jeho parametry zobrazí v grafických komponentách na příslušných záložkách. Po změnách pravidla uživatel tyto změny potvrdí tlačítkem / akcí hlavního okna, tj. mimo editační panel, a editační panel je tak požádán o zapsání všech provedených úprav z grafických komponent do daného pravidla. Je nutno dodat, že pokud uživatel nějakou změnu v pravidle provede, nemůže ze seznamu vybrat jiné pravidlo nebo provést jinou akci v hlavním okně, aniž by provedené změny potvrdil nebo naopak stornoval.

Třída editačního panelu tak kromě grafických prvků obsahuje i metody / sloty pro načtení pravidla do prvků v panelu, respektive pro zapsání změn do pravidla. Dále ve třídě nalezneme již pomocné metody, sloty a signály, které slouží pro vytváření grafických prvků stejného typu (například výběry negací) a eliminují tak nadbytečný kód, či které zajišťují interakci panelu na základě parametrů pravidla. Jinak řečeno metody a sloty, které jsou spojeny se signály vysílanými při určitých změnách v editačních prvcích a na jejichž základě se pak zakazuje editace určitých jiných prvků (například pokud není zadaná žádná IP adresa, nelze zadat ani její masku).

Aplikace podporuje detailnější analýzu protokolů síťové vrstvy. Jedná se o protokoly *IPv4* a *IPv6*. Právě pokud jeden z těchto protokolů vybere uživatel, je mu umožněn přístup na třetí záložku editačního panelu, kde může specifikovat parametry síťové vrstvy (zdrojovou a cílovou adresu, nesený protokol). Je nutné dodat, že v obou případech se zobrazuje jedna a ta samá záložka se stejnými grafickými prvky, které jsou využitelné pro oba typy podporovaných síťových protokolů. To je bráno jakožto výhoda, kdy je konfigurace pro uživatele v obou případech jednotná, což vede k celkovému většímu přehledu v uživatelském rozhraní.

V editačním panelu jsou také nadefinovány možné formáty síťových či linkových adres (MAC, IPv4, IPv6). Nadefinování je uskutečněno regulárními výrazy. Na základě těchto výrazů jsou vytvořeny validátory – `QRegExpValidator`, které jsou nastaveny prvkům pro zadání adres. Tím je docíleno toho, že již při samotném zadávání uživateli není povoleno odchýlit se od formátu adresy. Pokud chce uživatel zadat znak neodpovídající formátu, znak jednoduše do prvku není vložen.

Jak naznačuje obrázek 7.8, je specifikace jednoho parametru filtrovacího pravidla následující. Před každým parametrem je uveden nejprve jeho název / popis, dále je zde možnost negace parametru výběrem z voleb IS nebo NOT (negace parametru). Následuje pole pro zadání hodnoty parametru, případně parametrů (například adresa + maska) nebo výběr z předdefinované skupiny hodnot.



Obrázek 7.8 Zadání parametru filtrovacího pravidla

7.10.4 Vlastní prvky grafického uživatelského rozhraní

Pro realizaci funkce, kdy na základě nevyplnění některých parametrů filtrovacího pravidla při jeho editaci v editačním panelu je nutné do některých závislých prvků zakázat přístup, musely být nadefinovány celkem dva vlastní prvky uživatelského rozhraní.

Prvním je výběr negace, jenž je realizován takzvaným *combo boxem*. Tento prvek je možné vidět na obrázku 7.8. Jako základ posloužila třída `QComboBox`, od níž pak byla děděna třída `NegationComboBox`. Přidaná hodnota v této třídě je taková, že mechanismem signálů a slotů (kapitola 7.3.3) je combo box zaregistrován k prvku, který specifikuje hodnotu toho parametru, u kterého je combo box použit pro volbu negace. Pokud je hodnota v prvku změněna (změna textu, výběr jiného prvku z možností), je vyslán signál, který obsahuje změněnou hodnotu. Slot combo boxu tento signál zachytí a zkontroluje si jej s vlastním atributem definujícím takovou hodnotu, pro kterou je nutné zamezit do něj přístup. Tento atribut se nastavuje vlastní metodou. V praxi je pak zajištěno, že pokud máme například prázdné pole IP adresy nebo nevybrán žádný port zařízení, nelze toto negovat, tj. nelze zadat pravidlo typu NOT <all> nebo NOT <prázdná hodnota>.

Ze stejného důvodu bylo potřeba vytvořit i druhý vlastní prvek `MaskLineEdit`. Jedná se o rozšíření základního jednořádkového vstupu `QLineEdit`. Jak napovídá samotný název, je tento prvek určen pro zadávání masek u parametrů adres. Stejně jako v předchozím případě negací, i zde je nutné v situaci, kdy adresa zadána není, zakázat přístup a editaci této části parametru. Princip je shodný s negacemi.

7.10.5 Hlavní okno

Všechny výše uvedené komponenty grafického uživatelského rozhraní jsou v aplikaci seskupeny v jednom hlavním okně. Z hlediska kódu se pak jedná o třídu s názvem `MainWindow`.

Při vytváření třídy (v konstruktoru) se tedy vytvářejí všechny potřebné instance jednotlivých grafických komponent a případně právě zde jsou spojovány signály se sloty.

Hlavní okno také obsahuje drtivou většinu akcí a operací. Dá se říci, že právě do této třídy je agregována vrstva *controller* z architektonického vzoru MVC, jenž je

základem použité architektury *model / view* této aplikace. Více o architektuře v kapitole 7.3.

Zde tedy nalezneme hodně metod představující různé akce. K těmto metodám je možné se dostat ze dvou stran. Buďto kliknutím na nějakou položku v menu hlavního okna nebo kliknutím na nějaké z tlačítek umístěné v hlavním okně. Jednotlivé položky menu jsou specifikovány třídami `QAction`, které mohou u každé z nich definovat atributy jako klávesovou zkratku pro spuštění, ikonku, název, popisek apod. V případě spuštění akce (klávesovou zkratkou, kliknutím na položku v menu) emituje třída signál (mechanismus signálů a slotů je popsán v kapitole 7.3.3), čímž dává najevo spuštění. Tento signál je zachycen nějakým ze slotů hlavního okna. Slot pak požadovanou akci vykoná.

V případě zmáčknutí tlačítka emituje signál o zmáčknutí přímo tlačítko (nemusí se tlačítko akci přiřazovat). Opět jako v případě akcí z menu je signál zachycen k němu zaregistrovaným slotem hlavního okna a ten požadovanou akci vykoná. Pokud navíc tlačítko představuje stejnou akci, jako položka z menu, je v obslužném slotu stisku tlačítka zavolána pouze metoda slotu pro akci z menu. Tím se nemusí jedna akce definovat na dvou místech ve dvou slotech.

V hlavním okně jsou definovány tyto akce:

- nové filtrovací pravidlo,
- smazaní filtrovacího pravidla,
- duplikování filtrovacího pravidla,
- uložení změn / pravidel a jejich aplikování v systému,
- uložení změn v upravovaném pravidle (v editačním panelu),
- storno změn v upravovaném pravidle,
- ukončení aplikace,
- reset všech změn – znovunačtení pravidel ze souboru,
- vyvolání nastavení,
- vyvolání statistik,
- vyvolání ukázky výstupu pro Netfilter.

Operace nad pravidly není třeba nějak složitě popisovat. Vždy se z komponenty zobrazující seznam pravidel zjistí index vybraného pravidla a samotná operace je provedena nad modelovou třídou za použití indexu. Model si pak sám řekne o aktualizaci zobrazovací komponenty.

V případě vyvolání nastavení, statistik či ukázky výstupu se vždy zobrazují nová okna / dialogy. Jednotlivé dialogy jsou popsány dále v textu.

Uložení či storno změn upravovaného pravidla jsme si již popsali v kapitole věnující se editačnímu panelu (7.10.3).

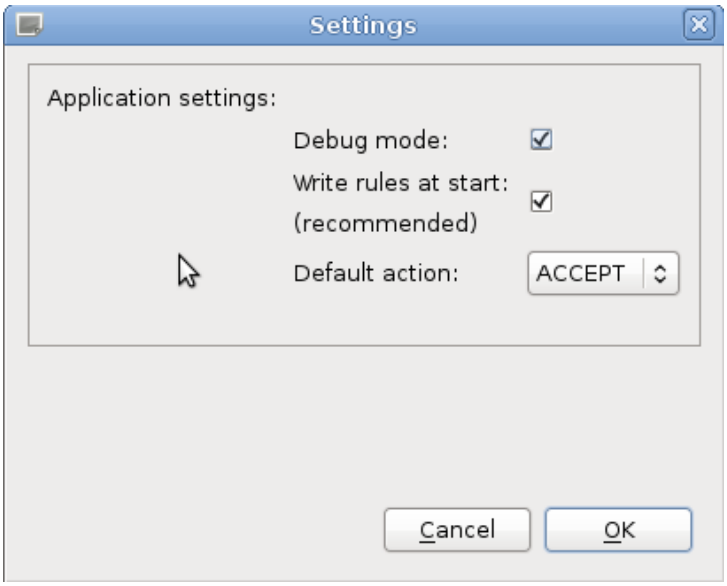
Při aplikaci a ukládání pravidel se používají knihovní třídy. Modelová třída seznamu pravidel poskytne seznam filtrovacích pravidel a na jeho základě jsou pravidla uložena do souboru a zapsána do frameworku Netfilter. Podrobněji se tomuto tématu věnují kapitoly 7.4 a 7.5. Zde si pouze uvedeme, že hlavní okno si udržuje stav změn, tedy pokud uživatel provedl nějaké změny v pravidlech,

případně nastavení. Pokud ano, teprve tehdy je uživateli umožněno změny aplikovat a uložit. V případě, že uživatel změny provedl, ale neuložil, je na tuto skutečnost při zavírání aplikace upozorněn.

Podobné jako při ukládání je chování aplikace také v případě načítání pravidel. To se standardně provádí při startu aplikace, avšak možnost znovunačtení je uživateli umožněna i při jejím běhu. Pouze však v případě, že uživatel nějaké změny provedl. Někdy se může stát, že se uživatel bude chtít vrátit do výchozího stavu před změnami. Tato akce umožňuje návrat k poslední uložené konfiguraci pravidel.

7.10.6 Dialog nastavení

Nastavení je věcí, která by se přímo na hlavním panelu objevit neměla. Lepším řešením je otevřít uživateli nové okno, kde lze nastavení měnit. Dialogové okno je znázorněno na obrázku 7.9.



Obrázek 7.9 Dialog nastavení

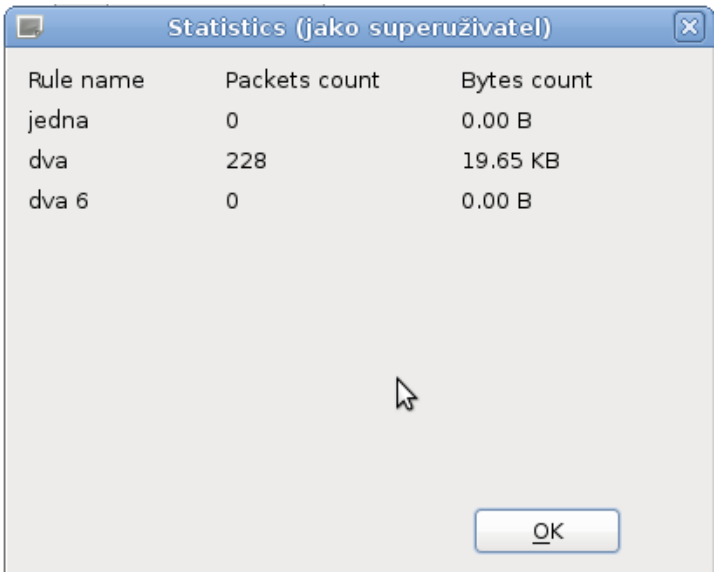
Okno s dialogem nastavení je implementováno třídou `SettingsDialog`, jenž je potomkem třídy `QDialog`, která slouží jako základ pro zobrazení dialogu. Tím se nemusíme zaobírat složitějším vytváření dialogového okna, pouze byla rozšířena funkčnost základního.

Dialogové okno nastavení v sobě obsahuje atributy v podobě grafických prvků, kterými zobrazuje uživateli aktuální nastavení a umožňuje ho změnit. Jelikož je nastavení aplikace jednoduché a obsahuje tři položky (výchozí akce, zápis pravidel při startu, ladící režim), je zobrazení nastavení realizováno dvěma prvky, pokud nebereme v úvahu popisky prvků. Konkrétně se jedná o `QCheckBox` pro ladící režim a zápis při startu a `QComboBox` pro výběr výchozí akce. Dále pak dialog obsahuje dvojici dialogových tlačítek `QDialogButtonBox` pro potvrzení či storno změn.

Při vytváření dialogu se z hlavního okna předá v konstruktoru ukazatel na strukturu obsahující strukturu aplikace. Dialog tak zná aktuální nastavení a může nastavit své prvky na správné hodnoty. Pokud uživatel provede v dialogu nějaké změny a dialog potvrdí stiskem tlačítka OK, jsou změny v prvcích zaznamenány a vloženy do struktury nastavení. Nastavení je pak také neprodleně uloženo do XML souboru (kapitola 7.4). Poté je vyslán signál informující o akceptaci dialogu a dialogové okno zavřeno. Na vyslaný signál reaguje k němu zaregistrovaný slot hlavního okna. Hlavní okno se totiž tímto způsobem „dozví“ o novém nastavení, na které patřičným způsobem zareaguje. Při změně výchozí akce si nastaví stav o neuložených změnách, při změně ladícího módu pak zobrazí nebo naopak skryje okno výstupu z logu.

7.10.7 Dialog statistik

O sběru statistik jsme se již zmínili v kapitole 7.5.3.



Obrázek 7.10 Dialog statistik

Statistiky se uživateli zobrazují podobně jako nastavení ve zvláštním dialogovém okně, viz obrázek 7.10. Implementace dialogu statistik je ve třídě `StatisticsDialog`, která je opět potomkem základní třídy dialogových oken `QDialog`.

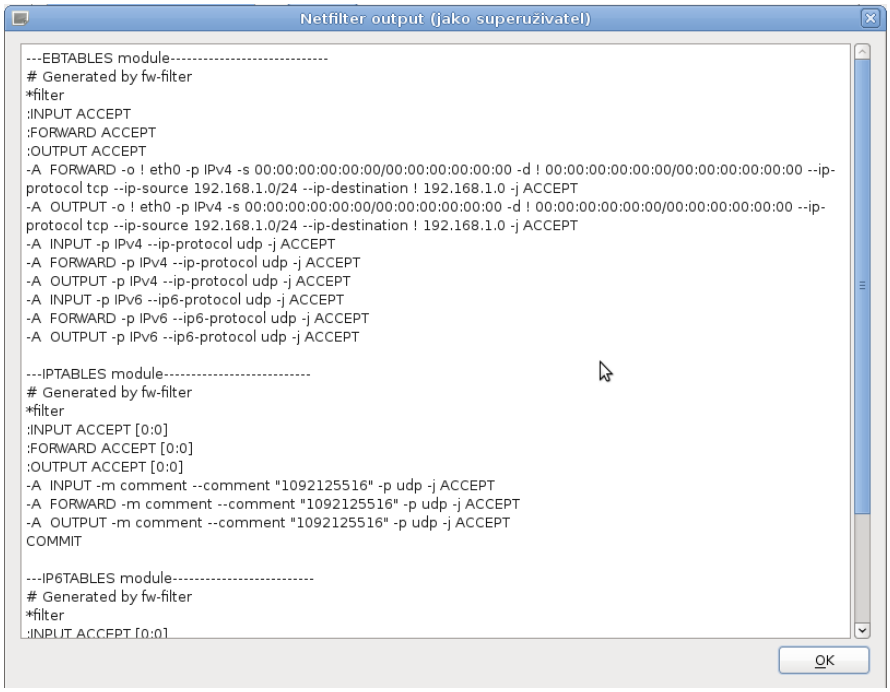
Po kliknutí na položku menu v hlavním okně se tedy uživateli zobrazí okno se statistikami. V konstruktoru je třídě dialogového okna předán seznam pravidel. Tento seznam je pak dále dialogovým oknem použit při volání knihovní třídy, která zajišťuje načítání statistik. Tato třída k jednotlivým pravidlům uloží i statistické informace, tj. doplní o ně seznam pravidel. Samotné dialogové okno pak pouze použije prvky grafického rozhraní (`QLabel`), ve kterých tyto informace uživateli vypíše.

Dialogové okno v sobě obsahuje časovač (`QTimer`), který při jeho otevření vytváří a po načtení statistik pak spouští. Časovač odpočítává nastavený interval a vždy při

jeho „vytikání“ emituje signál. Při vytváření časovače se k jeho signálu zaregistruje slot dialogového okna, který tak vždy zareaguje tím, že přes knihovní třídu opět načte statistiky z Netfilteru. Tím je realizována průběžná aktualizace statistik, pokud je okno otevřeno. V případě zavření okna se časovač vypne.

7.10.8 Ukázka výstupu pro Netfilter

Posledním dialogem, který se zobrazuje mimo hlavní okno je ukázka výstupu pro framework Netfilter. Dialog je implementován třídou `NfOutputDialog`, která je opět potomkem třídy `QDialog`.



Obrázek 7.11 Ukázka výstupu pro Netfilter

Při spuštění této akce z hlavního okna je nejprve knihovní třídou vygenerován výstup (popsáno v kapitole 7.5). Ten je ve formě řetězce (tentokrát se neukládá do souboru, ale je v paměti) předán v konstruktoru právě dialogovému oknu. Samotné okno pak obsahuje pouze víceřádkový textový prvek `QTextEdit`, ve kterém tento výstup uživateli zobrazí, viz obrázek 7.11. Uživatel výstup nemůže nijak editovat, pouze okno prostřednictvím tlačítka OK (`QDialogButtonBox`) zavřít.

8 Ověřování funkčnosti

8.1 Systémové požadavky

Aplikace byla vyvíjena a primárně testována na notebooku Acer Aspire 3690 s procesorem Intel Celeron M 420 (1,6GHz, 533 MHz Front Side Bus, 1MB L2 cache). Operační paměť stroje 2,50 GB DDR2, operační systém Debian Linux 6.0.7, verze jádra 2.6.32.

Kvůli použití frameworku Netfilter je možné aplikaci provozovat na jádře operačního systému ***Linux verze 2.6 a vyšší***, ačkoliv existuje i patch komponenty ebtables pro poslední jádra verze 2.4. Při použití pouze komponenty iptables by použití verze 2.4, od které je framework Netfilter již standardně součástí jádra, možné bylo. V tomto případě je však limitujícím prvkem komponenta ebtables.

Dále je nutné mít v operačním systému nainstalovány všechny tři používané programy uživatelského prostředí, přes které se přistupuje ke komponentám Netfilteru. Jedná se o stejnojmenné programy:

- ***iptables*** (při vývoji a testování použita verze *1.4.8* standardně dostupná z repozitářů OS Debian),
- ***ip6tables*** (při vývoji a testování použita verze *1.4.8* standardně dostupná z repozitářů OS Debian),
- ***ebtables*** (při vývoji a testování použita verze *2.0.10 – 4*).

Při standardní instalaci ebtables z repozitářů OS Debian se ukázalo, že stažený a nainstalovaný program neobsahoval příkaz / program *ebtables-restore*, z toho důvodu bylo nutné ručně stáhnout poslední verzi ze stránek projektu (http://ebtables.sourceforge.net) a tuto verzi nainstalovat. Zdrojové soubory použité verze 2.0.10 – 4 jsou přiloženy na CD s celou aplikací.

Pro účely práce z XML soubory je ve zdrojových kódech použita knihovna ***libxml2***. Na vývojovém systému byla tato knihovna nainstalována balíkem *libxml2-dev* a při překladu zdrojových kódů je zapotřebí tuto knihovnu v systému mít zavedenou. Dále je nutné zkontrolovat, zda se knihovna nainstalovala do správného umístění - */usr/include/libxml*. Je možné a často se děje, že se knihovna nainstaluje do umístění */usr/include/libxml2/libxml*. V tom případě je nutné vytvořit symbolický odkaz v systému */usr/include/libxml > /usr/include/libxml2/libxml*.

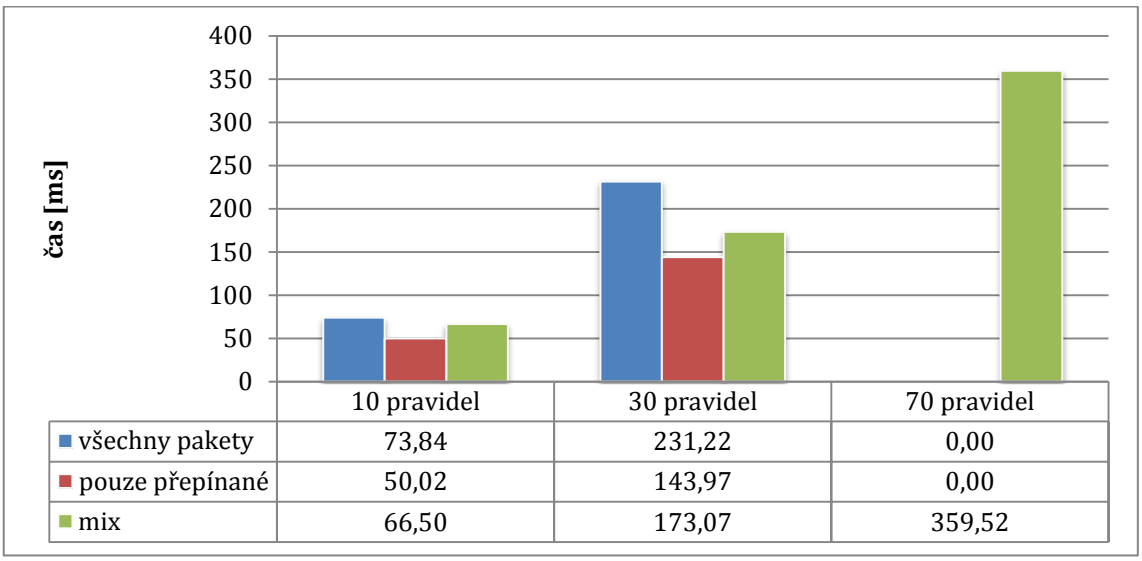
Pro správné přeložení zdrojových souborů je dále nutné mít v systému nainstalované vývojové knihovny ***Qt***. Při vývoji byl použit balík *qt4-dev* obsahující verzi 4.8.

Jelikož aplikace pracuje s jádrem operačního systému (programy iptables, ip6tables a ebtables), jsou vyžadována administrátorská oprávnění. Z tohoto důvodu je potřeba aplikaci spouštět jako uživatel *root* nebo příkazem *sudo*. V opačném případě aplikace sama vypíše při jejím startu varování.

8.2 Doba zápisu filtrovacích pravidel

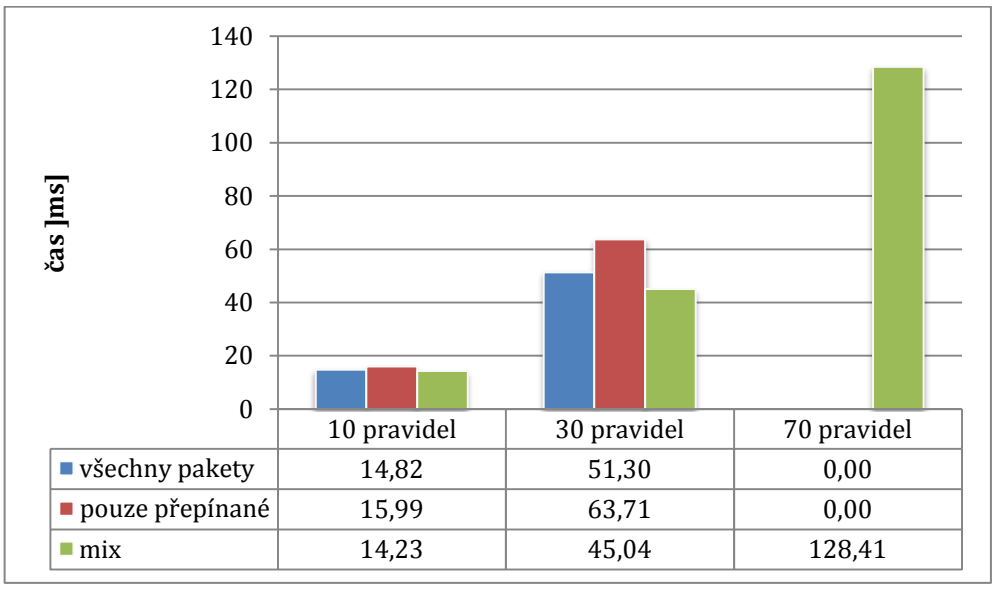
Průměrné doby zápisu pravidel do jádra operačního systému a do XML souboru jsou znázorněny na následujících grafech.

Uvedené grafy znázorňují naměřené hodnoty doby zápisu v případě 10, 30 a 70 pravidel v aplikaci. Měření bylo prováděno zvlášť pro oba dva typy pravidel a také pak společně. V případě 70 pravidel se provádělo pouze měření situace, kdy jsou v aplikaci používány oba dva typy pravidel společně. Zobrazené hodnoty jsou průměry z několika sekvenčních měření.



	10 pravidel	30 pravidel	70 pravidel
všechny pakety	73,84	231,22	0,00
pouze přepínané	50,02	143,97	0,00
mix	66,50	173,07	359,52

Obrázek 8.1 Graf doby zápisu pravidel do systému



	10 pravidel	30 pravidel	70 pravidel
všechny pakety	14,82	51,30	0,00
pouze přepínané	15,99	63,71	0,00
mix	14,23	45,04	128,41

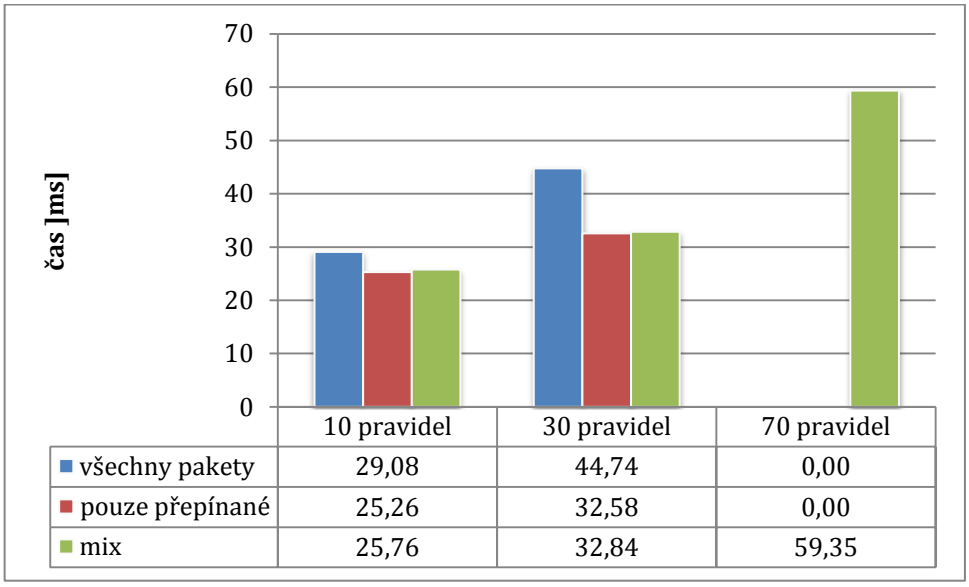
Obrázek 8.2 Graf doby zápisu pravidel do XML souboru

Z výše uvedených grafů lze konstatovat, že obě operace ukládání a zápisu probíhají plynule.

8.3 Doba načítání filtrovacích pravidel

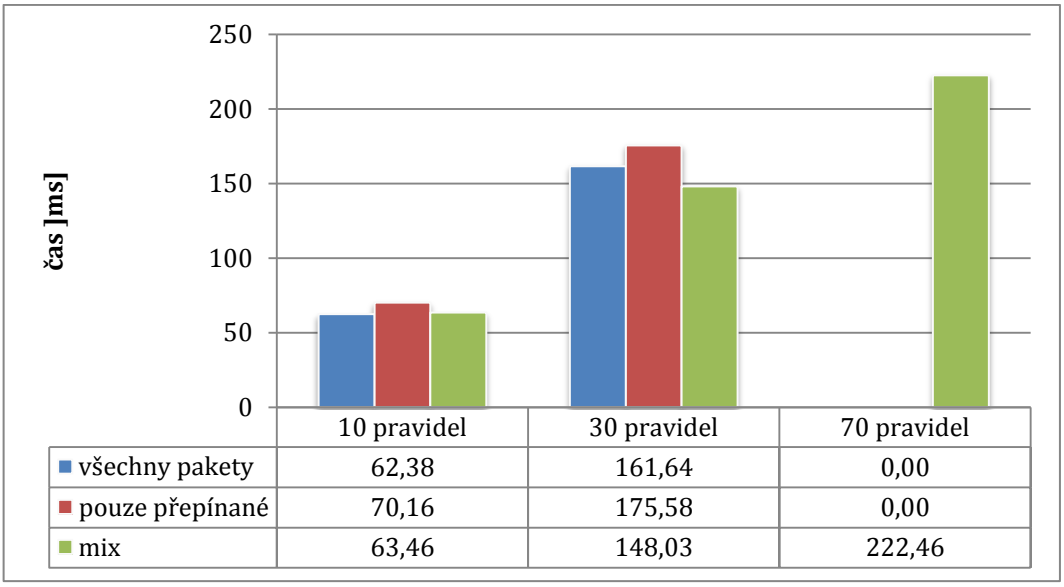
Průměrné doby načtení statistických informací z jádra operačního systému a také načtení filtrovacích pravidel z XML souboru jsou znázorněny na následujících grafech.

Způsob měření je shodný jako v případě zápisu filtrovacích pravidel, viz předchozí odstavec.



	10 pravidel	30 pravidel	70 pravidel
všechny pakety	29,08	44,74	0,00
pouze přepínané	25,26	32,58	0,00
mix	25,76	32,84	59,35

Obrázek 8.3 Graf doby načtení statistik



	10 pravidel	30 pravidel	70 pravidel
všechny pakety	62,38	161,64	0,00
pouze přepínané	70,16	175,58	0,00
mix	63,46	148,03	222,46

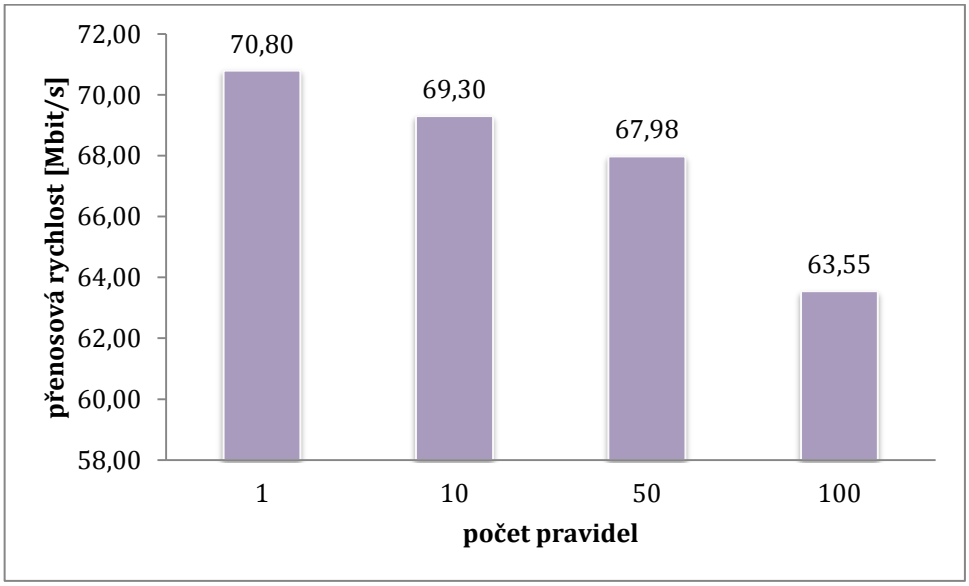
Obrázek 8.4 Graf doby načtení pravidel z XML souboru

Z výše uvedených grafů lze opět konstatovat, že načítání dat nepředstavuje v aplikaci žádné časové zdržení či omezení.

8.4 Propustnost

K měření propustnosti systému byla použita utilita *Iperf* (http://iperf.sourceforge.net/), která se používá k tomuto účelu. Utilita se skládá ze dvou částí (server a klient), které jsou umístěny na dvou různých zařízeních. Klient se připojí na server a poté je spuštěn test, jehož výsledkem je maximální přenosová rychlost na síťovém spoji mezi klientem a serverem.

Při měření propustnosti se vždy měřila komunikace v obou směrech. Výsledky znázorněné v grafu na obrázku 8.5 představují průměr naměřených hodnot z více testů (sekvenčně za sebou pouštěných). Firewall byl při měření nakonfigurovaný tak, aby propouštěl pouze pakety měřící utility. Počet pravidel pak představuje počet pravidel, které byly ve firewallu před pravidly povolující komunikaci měření, tj. počet pravidel, „přes“ které pakety musely „projít“ při filtračním rozhodování.



Obrázek 8.5 Graf propustnosti

Z výše uvedených výsledků je patrný vliv firewallu na propustnost počítačové sítě, který plně koresponduje s výhodami a nevýhodami filtrování síťové komunikace popsanými v kapitole 2.3.

8.5 Odezva systému

Systém se na výše popsaném stroji (kapitola 8.1) chová plynule a při dalších spuštěných programech (cca 2-4 programy). Odezva systému na příkazy uživatele je plynulá.

9 Závěr

Cílem této diplomové práce bylo seznámit se s problematikou síťového provozu, porovnat možnosti filtrování síťového provozu na linkové a síťové vrstvě, nalézt vhodné API pro možnost filtrování na obou vrstvách na platformě Linux, navrhnout architekturu pro univerzální SW firewall a řešení v praxi ověřit implementací včetně grafického uživatelského rozhraní.

Výsledkem je univerzální softwarový firewall implementovaný na platformě Linux, jenž k účelu filtrování používá v dnešní době nejrozšířenější framework, který se na platformě Linux k tomuto účelu používá, framework Netfilter. Netfilter je součástí jádra operačního systému Linux od verze 2.4. Podrobnější analýza však zjistila absenci API pro konfiguraci a zápis pravidel ve frameworku. Z tohoto důvodu musely být použity příkazy jednotlivých použitých komponent – ebtables, iptables, ip6tables. Tento fakt představuje asi nejslabší místo výsledného řešení, nicméně na druhou stranu přesně toto řešení je doporučeno vývojáři Netfilteru do té doby, než přijde jeho nová verze s implementovaným API. Na tuto skutečnost se při návrhu univerzálního firewallu myslelo a ten je tak z hlediska architektury připraven na změny v této oblasti. Díky knihovním třídám, které zajišťují komunikaci a přístup právě k Netfilteru tak stačí provést změny těchto tříd, přičemž další zásah napříč celou aplikací není nutný.

Aplikace umožňuje filtrování na linkové i síťové vrstvě, přičemž je možné filtrovat provoz v režimu síťových mostů. Tato funkčnost tak umožňuje z aplikace udělat transparentní síťový filtr, který je útočníky velmi obtížně identifikovatelný a který představuje mocnou zbraň z hlediska bezpečnosti počítačové sítě.

Aplikací lze filtrovat síťový provoz na linkové úrovni pouze na sítích typu Ethernet. Toto je způsobeno kódem jádra operačního systému Linux, který umí lépe pracovat s rámci na linkové úrovni pouze tohoto typu sítě. Na síťové úrovni lze podrobně analyzovat pakety protokolů IPv4 a IPv6. Společně s Ethernetem se jedná o nejčastější kombinaci protokolů, která se dnes používá ve všech standardních počítačových sítích.

Pro jednoduchost nastavení a konfigurace bylo implementováno grafické uživatelské prostředí za použití knihovny pro tvorbu grafických rozhraní - *Qt*. Právě jednoduchost a přehlednost byly při návrhu GUI hlavními body. Pokud se uživatel aplikace v grafickém rozhraní snadno orientuje, eliminuje se tím počet chyb, které může při konfiguraci filtrovacích pravidel udělat. Navíc samotná aplikace se snaží pomoct uživateli a grafické rozhraní tak například při specifikaci filtrovacího pravidla neumožňuje uživateli přístup do sekcí nebo k parametrům, které nemají z hlediska již nakonfigurovaných parametrů dále význam. Dále pak samozřejmě aplikace hlídá, zda uživatel všechny provedené změny zapsal do operačního systému. Zápis se provádí na žádost z toho důvodu, aby uživatel při změnách pravidel mohl nejprve provést všechny změny a ty najednou do systému zapsat a uvést v platnost.

Aplikace také poskytuje určitou zpětnou vazbu ve formě statistik. Pro každé filtrovací pravidlo je možné zobrazit počet ovlivněných paketů a bytů. To nabízí uživateli jednak informaci, že systém opravdu „něco dělá“, a také jaká pravidla jsou

nejvíce používána, což lze také chápat jako fakt vypovídající o struktuře síťového provozu.

Pro aplikaci byla zvolena architektura model / view, která vychází z architektonického vzoru Model View Controller a pro kterou jsou již v knihovně Qt připraveny základní třídy. Navíc v případě použití navrženého řešení zadavatelskou firmou Kerio Technologies Inc., jejíž síťový produkt používá konfiguraci prostřednictvím webových stránek běžících na daném zařízení, lze oddělit ze současné view vrstvy akce patřící do vrstvy controller a dosáhnout tak MVC architektury, která se běžně používá ve webových aplikacích.

Systém byl vytvářen dle standardních postupů od analýzy až po testování vyvinuté aplikace. Při tvorbě byl použit verzovací systém GIT s hlavním repozitářem ve službě *GitHub* (http://github.com/mpetrak/fw-filter).

Vyvinutý systém je připraven pro další rozvoj. Kromě již zmíněné případné změny komunikace s frameworkem Netfilter při vyvinutí API lze pak další rozšíření implementovat především v dalších možnostech filtrování. Systém je možné rozšířit zejména o podporu dalších protokolů především síťové vrstvy. Nabízí se možnost protokolu ARP, přičemž je možné využít komponentu Netfilteru arptables. Dále je možné z hlediska filtrace postupovat i do vyšších vrstev a filtrovat tak na základě čísel portů apod.

Přehled zkratek a použitého značení

abc Jemné zdůraznění, důležité pojmy.

abc Intenzivní zdůraznění, velmi důležité pojmy.

`abc` Názvy tříd, zdrojový kód.

API	Aplikační rozhraní, které slouží pro komunikaci mezi dvěma aplikacemi / systémy.
ARP	Address Resolution Protocol, protokol používaný k překladu IP adres na odpovídající MAC adresy.
DHCP	Dynamic Host Configuration Protocol, protokol používaný k automatické konfiguraci počítačů v počítačové síti.
DNS	Domain Name System, systém doménových jmen obsahující DNS servery a komunikační protokol.
DOS	Denial Of Service, typ útoků, které mají za cíl nedostupnost dané služby. Většinou se realizují zahlcením sítě / stroje požadavky tak, že cíl útoku je prakticky nedostupný.
Ethernet	Typ počítačové sítě (vrstvy síťového rozhraní v zásobníku TCP/IP), definuje mimo jiné i strukturu rámců na linkové vrstvě.
FDDI	Fiber distributed data interface, typ počítačové sítě.
HW	Hardware.
ICMP	Internet Control Message Protocol, protokol používaný k řízení počítačové sítě Internet, posílá řídící zprávy.
IPv4	IP protokol verze 4.
IPv6	IP protokol verze 6.
ISO/OSI	International Standards Organization“/„Open System Interconnection, referenční komunikační model používaný v počítačových sítích.
MAC	Media Access Control. Podvrstva vrstvy síťového rozhraní v zásobníku TCP/IP, MAC adresa jednoznačně identifikuje síťové rozhraní, přiřazená výrobcem zařízení.
MIME	Multipurpose Internet Mail Extension, původně v elektronické poště pro rozlišení typu příloh, dnes se používá i v dalších protokolech (např. HTTP) a aplikacích.

MTU	Maximum Transmission Unit, maximální možná velikost IP paketu. Více např. ve článku: http://www.root.cz/clanky/velke-trable-s-malym-mtu/.
MVC	Model View Controller, architektonický vzor pro tvorbu aplikací.
NAT	Network Address Translation, překlad síťových adres formou změny zdrojové nebo cílové adresy.
NF	Netfilter, zkratka použitá v této práci.
TCP/IP	Protokolový zásobník, který se dnes používá v Internetu a jemu obdobných počítačových sítích.
RMON	Remote Network Monitoring, standard pro monitorování počítačových sítí. Specifikuje různé monitorovací zařízení a výměnu naměřených dat s aplikacemi zobrazujícími naměřená (mohou ležet v jiné síti).
SW	Software.
TCP	Transmission Control Protocol, spojově orientovaný transportní protokol představující spolehlivý přenos dat.
UDP	User Datagram Protocol, nespojovaný transportní protokol, nezaručuje ztrátu paketů, nehlídá změnu pořadí jednotlivých paketů.
XML	Formát souborů a zkratka značkovacího jazyka Extensible Markup Language.

Seznam obrázků

Použité zdroje

[1] **Zwicky, Elizabeth D., Cooper, Simon a Chapman, Brent.** *Building Internet Firewalls, Second Edition.* Sebastopol : O'Reilly Media, 2000. ISBN 1-5652-871-7.

[2] Model ISO/OSI. *Počítačové sítě SITE.THE.CZ.* [Online] [Citace: 19. 03 2013.] http://site.the.cz/index.php?id=4.

[3] **ITU-T.** *X.200 : Information technology - Open Systems Interconnection - Basic Reference Model: The basic model.* [PDF Dokument] Červenec 1994.

[4] **Reed, Darren.** IP Filter. *IP Filter - TCP/IP Firewall/NAT Software.* [Online] [Citace: 02. 12 2012.] http://coombs.anu.edu.au/~avalon/.

[5] **Russel, Rusty.** Linux IPCHAINS-HOWTO. *The netfilter.org project.* [Online] 12. 07 2000. [Citace: 02. 12 2012.] http://people.netfilter.org/~rusty/ipchains/HOWTO.html.

[6] **X/OS Experts in Open Systems BV.** IP firewall administration. *X/OS Experts in Open Systems BV.* [Online] 2006. [Citace: 02. 12 2012.] http://www.xos.nl/resources/ipfwadm.

[7] **Borkman, Daniel.** netsniff-ng - the packet sniffing beast. *netsniff-ng - the packet sniffing beast.* [Online] 2009-2012. [Citace: 02. 12 2012.] http://netsniff-ng.org/.

[8] **McCanne, Steven a Jacobson, Van.** *The BSD Packet Filter: A New Architecture for User-level Packet Capture.* [PDF Dokument] 1992.

[9] **Jengelh.** File:Netfilter-components.svg. *Wikipedia, The Free Encyclopedia.* [Online] 13. 12 2008. [Citace: 02. 04 2013.] http://en.wikipedia.org/wiki/File:Netfilter-components.svg.

[10] **Russell, Rusty a Welte, Harald.** Linux netfilter Hacking HOWTO. *The netfilter.org project.* [Online] 2. 07 2002. [Citace: 2. 04 2013.] http://www.netfilter.org/documentation/HOWTO//netfilter-hacking-HOWTO.html#toc9.

[11] **Schuymer, Bart De a Fedchik, Nick.** ebtables/iptables interaction on a Linux-based bridge. *ebtables - Linux Ethernet bridge firewalling.* [Online] 9. 11 2003. [Citace: 2. 04 2013.] http://ebtables.sourceforge.net/br_fw_ia/br_fw_ia.html.

[12] **Welte, Harald.** netfilter/iptables FAQ. *The netfilter.org project.* [Online] 04. 06 2007. [Citace: 11. 11 2012.] http://www.netfilter.org/documentation/FAQ/netfilter-faq.html.

[13] Bridge-nf Frequently Asked Questions. *ebtables - Linux Ethernet bridge firewalling.* [Online] 04. 12 2004. [Citace: 11. 11 2012.] http://ebtables.sourceforge.net/misc/brnf-faq.html.

[14] **Digia plc.** Model/View Programming. *Qt Project.* [Online] [Citace: 01. 12 2012.] http://qt-project.org/doc/qt-4.8/model-view-programming.html.

[15] —. Signals & Slots. *Qt Project.* [Online] [Citace: 28. 04 2013.] http://qt-project.org/doc/qt-4.8/signalsandslots.html.

[16] Man page of EBTABLES. *ebtables - Linux Ethernet bridge firewalling.* [Online] 10. 07 2011. [Citace: 05. 01 2013.] http://ebtables.sourceforge.net/misc/ebtables-man.html.

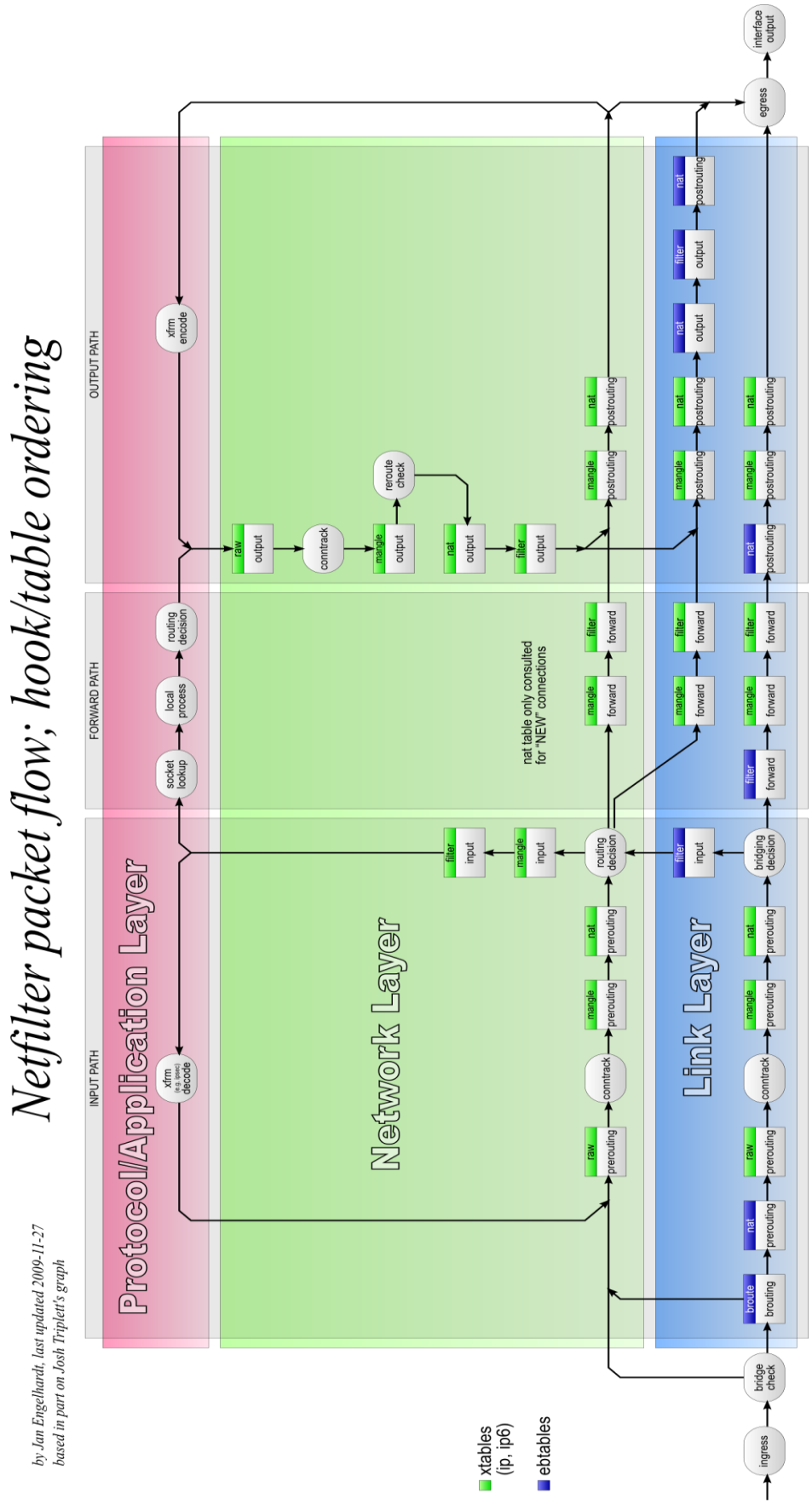
[17] Man page of IPTABLES. *The netfilter.org project.* [Online] 20. 10 2008. [Citace: 06. 01 2013.] http://ipset.netfilter.org/iptables.man.html.

[18] **Szabo, Andras Kis-Szabo.** *Man page of IP6TABLES.*

[19] **Jenkins, Bob.** A hash function for hash Table lookup. [Online] [Citace: 03. 19 2013.] http://www.burtleburtle.net/bob/hash/doobs.html.

[20] **Hare, Chris a Siyan, Karanjit.** *Internet firewalls and network security.* 2. vydání. Indianapolis : New Riders Publishing, 1996. ISBN 1-56205-632-8.

Příloha A: Graf průchodů paketů / rámců skrze kód Netfilter



Zdroj: http://en.wikipedia.org/wiki/File:Netfilter-components.svg

Příloha B: Uživatelská příručka

FW filter: Softwarový firewall pro filtrování na síťové a linkové vrstvě

B.1 Instalace a spuštění programu

B.1.1 Systémové požadavky

Aplikace pro svou činnost používá framework Netfilter. Z toho důvodu je možné aplikaci provozovat na jádře operačního systému ***Linux verze 2.6 a vyšší***.

Dále je nutné mít v operačním systému nainstalovány následující programové balíčky:

- ***iptables*** (doporučena verze *1.4.8*)
- ***ip6tables*** (doporučena verze *1.4.8*)
- ***ebtables*** (doporučena verze *2.0.10 – 4*).

Všechny balíčky lze do systému nainstalovat ručně, je však doporučeno použít standardní balíčkovací systém konkrétní distribuce operačního systému Linux.

Je nutné, aby programové balíčky byly kompletní, tj. včetně rozšíření typu *iptables-restore*, respektive *ip6tables-restore* či *ebtables-restore*. Může se vyskytnout problém zejména u balíčku ebtables, který v případě, že přes balíčkovací systém nebude nainstalován kompletně, je nutný nainstalovat ručně stažením zdrojových souborů a jejich překladem programem *make*. Zdrojové soubory ebtables verze 2.0.10-4 jsou přiložené k programu.

Dále je pro překlad a sestavení aplikace nutné mít v operačním systému knihovnu ***libxml2*** a ***Qt***. Zde postačí standardní instalace balíčkovacím systémem. V případě distribuce Debian Linux a balíčkovacího systému *apt* se jedná o balíky *qt4-dev* a *libxml2-dev*. Bez těchto dvou knihoven nepůjdou zdrojové kódy přeložit.

V případě knihovny libxml2 je nutné zkontrolovat, existenci knihovny ve standardním umístění */usr/include/libxml*. Je možné a často se tak děje, že se knihovna nainstaluje do umístění */usr/include/libxml2/libxml*. V tom případě je nutné vytvořit symbolický odkaz v systému */usr/include/libxml > /usr/include/libxml2/libxml*.

B.1.2 Instalace a překlad zdrojových kódů

Aplikace je distribuována ve formě zdrojových kódů. Po stažení složky zdrojových kódů do požadovaného umístění je nutné jejich překlad.

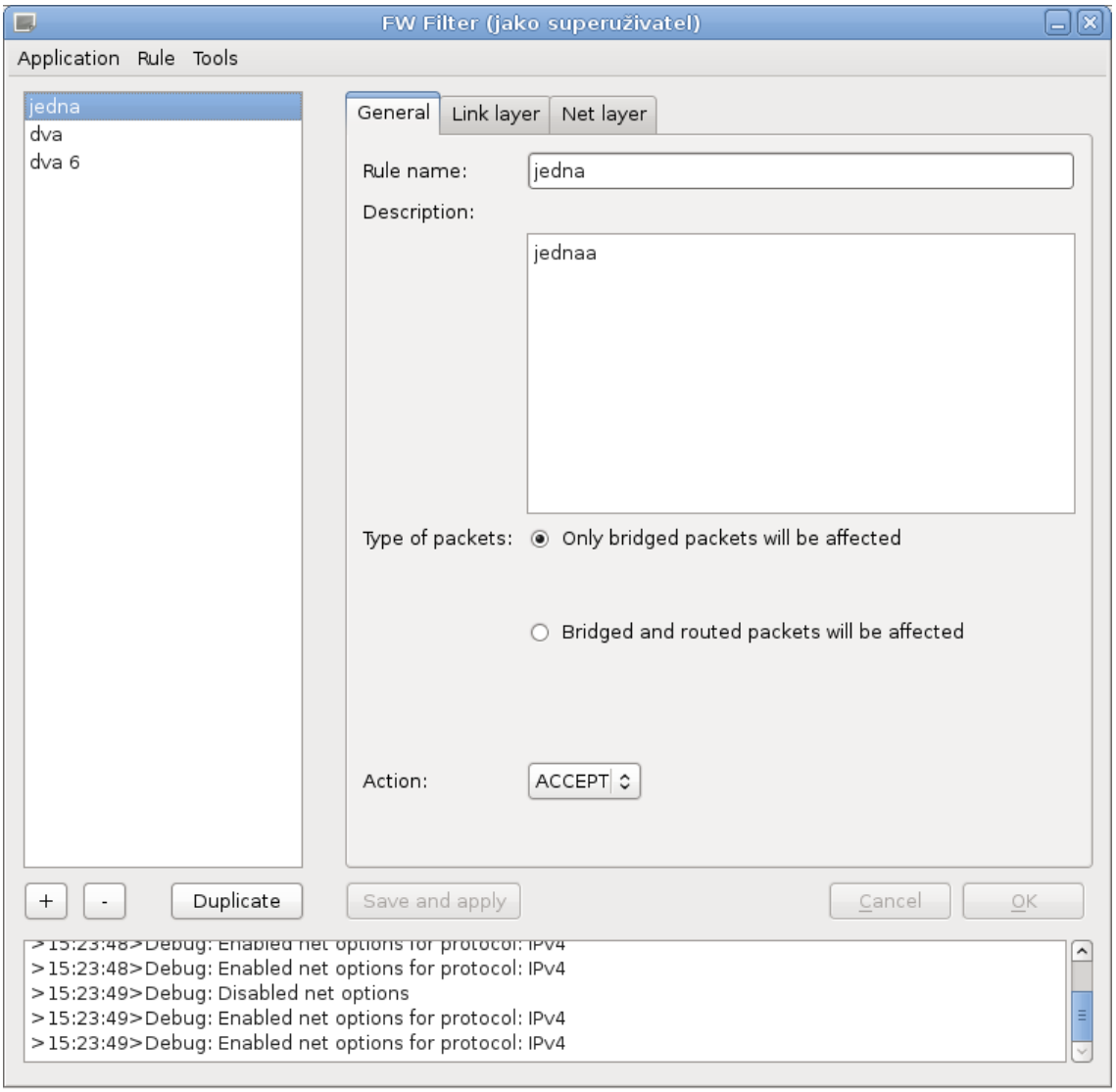
Pro přeložení zdrojových kódů je v hlavním adresáři *Makefile*. Aplikaci tak přeložíme zavoláním příkazu *make*. Tím proběhne samotný překlad a v hlavním adresáři se vytvoří spustitelný soubor s názvem aplikace – *fw-filter*.

B.1.3 Spuštění

Aplikaci lze spustit souborem ***fw-filter*** v hlavním adresáři. Pro práci aplikace vyžaduje přístup k nastavení operačního systému. Z toho důvodu je nutné aplikaci spouštět s administrátorskými oprávněními (jako *root*, případně příkazem *sudo*). V opačném případě nelze zapsat filtrovací pravidla do systému a aplikace na to sama při startu upozorní.

B.2 Popis ovládání

B.2.1 Hlavní okno



Hlavní okno (znázorněné na obrázku výše) poskytuje veškerou hlavní funkčnost aplikace. V pravé části je seznam filtrovacích pravidel, v levé části editační panel, ve kterém lze vybrané pravidlo ze seznamu editovat.

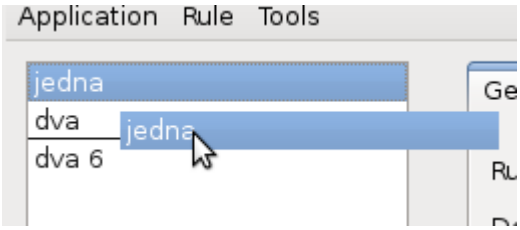
B.2.2 Přidání, duplikování, odebrání pravidla

K tomuto účelu slouží trojice tlačítek s označením *+* (přidání), *-* (odebrání) a *Duplicate* (duplikace). Akce je platná vždy na vybrané pravidlo. V případě přidávání pravidel se pak pravidlo přidá bezprostředně za vybrané pravidlo.

Při duplikaci jsou zkopírovány všechny parametry mateřského pravidla, pouze ke jménu je přidán dodatek *copy*.

B.2.3 Řazení pravidel

Řazení seznamu pravidel probíhá mechanismem *drag and drop*. Jednoduše tak lze pravidlo myší „přetáhnout“ do požadované pozice, jak ukazuje následující obrázek.



B.2.4 Editace filtrovacího pravidla

Po výběru jednoho pravidla ze seznamu se jeho parametry zobrazí v editačním panelu. Editační panel obsahuje celkem tři záložky, které odpovídají skupině parametrů patřící k sobě.

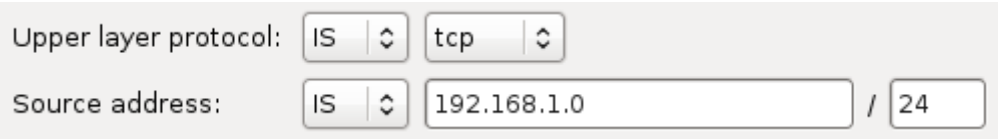
Na prvním záložce lze nastavit základní informace jako:

- jméno pravidla (bude zobrazeno v seznamu pravidel),
- popis pravidla (detailnější text),
- typ pravidla,
- akce.

Typ pravidla udává, jaké další možnosti pro jeho editaci lze uvést a hlavně jaké pakety bude pravidlo ovlivňovat. V případě volby přepínaných a směrovaných paketů (*Bridged and routed packets will be affected*), tj. tedy všech paketů procházející zařízením, lze editovat dále parametry pouze na síťové vrstvě. Je tak nutné vybrat typ síťového protokolu na druhé záložce (IPv4 nebo IPv6), případně i vstupní a výstupní rozhraní, a poté vyplnit parametry síťové vrstvy na třetí záložce.

Pokud vybereme volbu pouze přepínaných paketů (*Only bridge packets will be affected*) lze editovat na linkové vrstvy kromě vstupního a výstupního portu také zdrojové a cílové adresy této vrstvy (HW adresy). Kromě toho je na výběr větší množství protokolu. Pokud je z protokolů vybrán IPv4 nebo IPv6 protokol, můžeme detailněji parametrizovat tuto volbu na třetí v tu chvíli zpřístupněné záložce týkající se síťové vrstvy.

B.2.4.1 Forma zadání parametrů



U filtrovacího pravidla lze zadat jeho parametry dvojím způsobem. Oba dva způsoby jsou znázorněny na obrázku výše.

První skupinou parametrů jsou výběry z možností, kde uživateli není dovoleno specifikovat vlastní hodnotu parametru.

Druhou skupinou je zadání hodnoty parametru ve formě textového řetězce. Tento typ tvoří zejména různé typy adres včetně jejich masek. Každý typ adresy (MAC, IPv4, IPv6) má nadefinovaný formát a pole tak umožňuje uživateli zadat pouze

tento formát. Nelze tak zadat nesprávné znaky nebo správné znaky na nesprávných místech. Takové znaky nebudou do textu zapsány!

U adres se mohou použít také masky. Pokud masky nejsou použity, aplikace chápe parametr jako konkrétní adresu. S maskou se zadávají zejména síťové rozsahy ve formátu adresa sítě / maska. Pole pro masku tedy nemusí být použito a lze jej nechat nevyplněné. Nelze však zadat masku, aniž by předtím nebyla zadána hodnota adresy.

Celý parametr lze také negovat. K tomuto účelu slouží na začátku přepínač hodnot IS / NOT. Hodnota NOT znamená negaci celého parametru. Jako příklad můžeme tedy zadat jedním parametrem všechny pakety nesoucí jiný, než TCP protokol tím, že vybereme v nabídce na síťové vrstvě TCP protokol společně s přepínačem negace nastaveným na hodnotu NOT.

B.2.5 Ukládání změn, aplikování pravidel

Každou editaci jednoho pravidla je nutné před dalšími úkony uložit tlačítkem *Ok* na hlavní obrazovce vpravo pod editačním panelem. Provedené změny lze také zrušit vedle umístěným tlačítkem *Cancel*.

Po uložení změn se pravidlo ihned do systému neaplikuje! Je nutné provedené a uložené změny aplikovat (a uložit seznam pravidel) tlačítkem *Save and apply* v dolní části hlavního okna. To je z toho důvodu, že pokud se mění větší počet pravidel, než jedno, je vhodné nejprve změnit celou konfiguraci, která se pak najednou aplikuje, aby nedošlo k chybám při postupných změnách.

Při změnách filtrovacích pravidel lze také všechny provedené změny zrušit a vrátit se k poslední uložené konfiguraci. Tato možnost je skryta v menu pod záložkou *Application – reset all changes*. a slouží jako případná záchranná brzda.

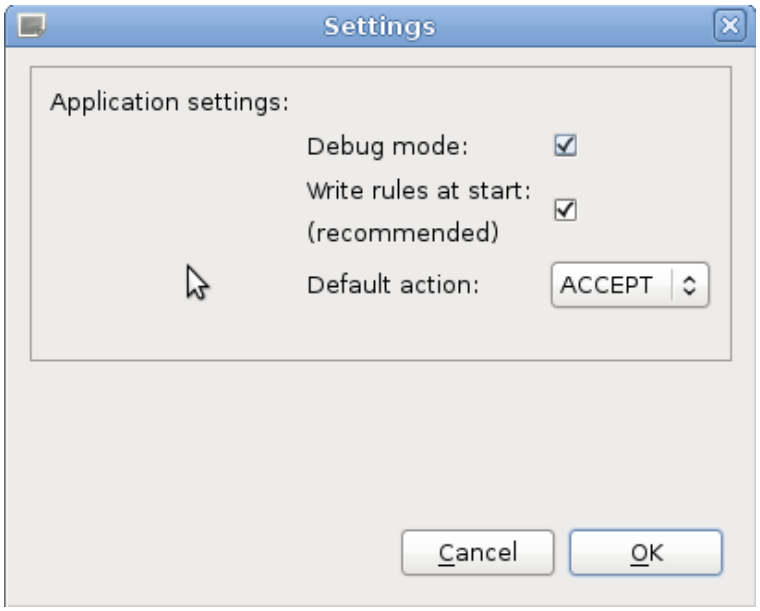
B.2.6 Klávesové zkratky, menu

Každou akci lze mimo tlačítka provést i z menu hlavního okna. V tomto menu se dále nacházejí další akce, které nemají zastoupení v hlavním okně jako tlačítka. Jedná se o akce, u nichž se jejich používání nepředpokládá tak často.

Kromě tlačítek a menu lze také ty nejčastější úkony spouštět klávesovými zkratkami. Zkratky jsou uvedeny vždy u příslušné akce v menu.

B.3 Nastavení

Dialogové okno zobrazující možnosti nastavení lze vyvolat přes menu *Tools* – *Settings*. Okno je znázorněno na následujícím obrázku.



V nastavení lze vypnout či zapnout ladící mód (*Debug mode*). Pouze při zapnutém módu obsahuje hlavní okno aplikace v jeho dolní části výstup s výpisy.

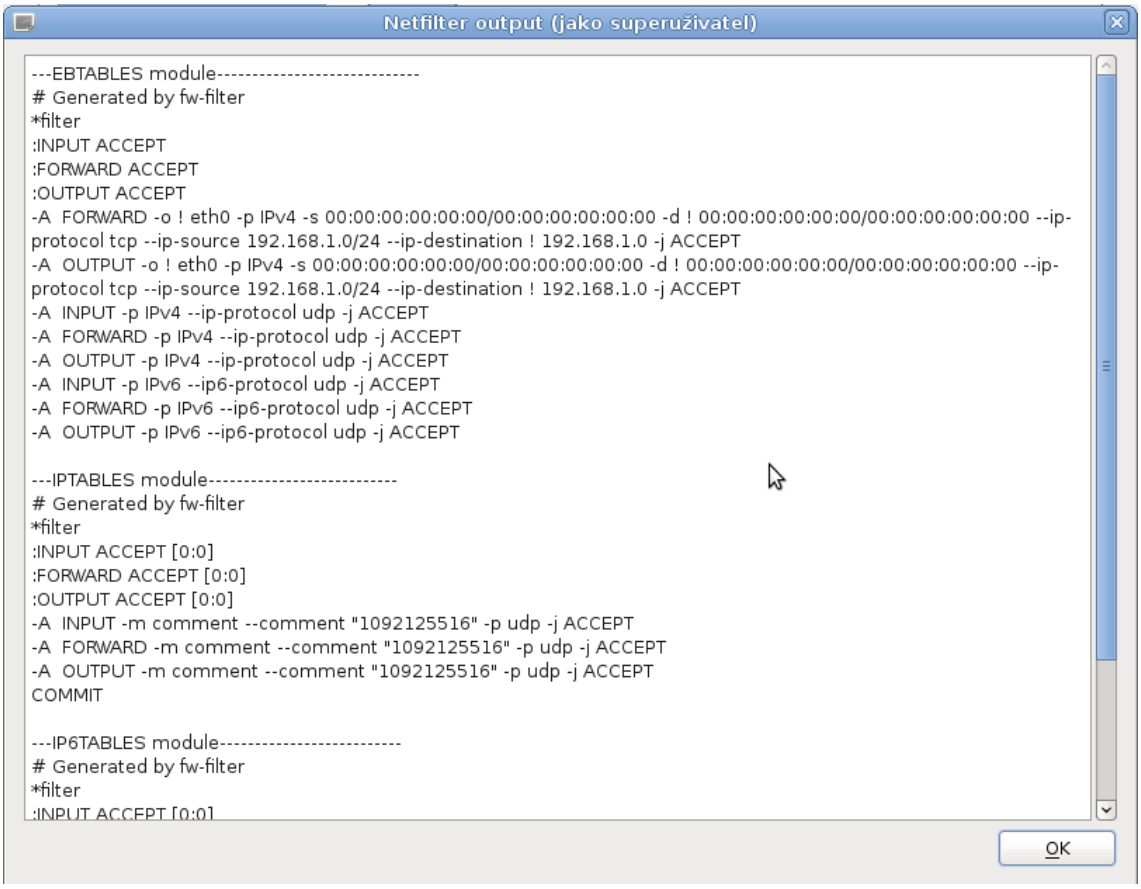
Dále lze nastavit, zda se při každém startu aplikace mají pravidla zapsat do operačního systému. Je vhodné ponechat tuto volbu zaškrtnutou a vypínat ji pouze v ojedinělých případech. Jde o to, že pokud bude operační systém vypnut nebo restartován, při následném startu se všechna pravidla z něj ztratí. Proto ihned po startu aplikace budou filtrovací pravidla do systému opět zapsána.

Poslední volbou nastavení je výchozí akce pro všechny pakety, které nejsou filtrovacími pravidly specifikovány. Ve výchozím nastavení se takové pakety dále propouštějí, aby po instalaci systém nepřerušil veškerou síťovou komunikaci.

B.4 Další nástroje

B.4.1 Ukázka výstupu pro Netfilter

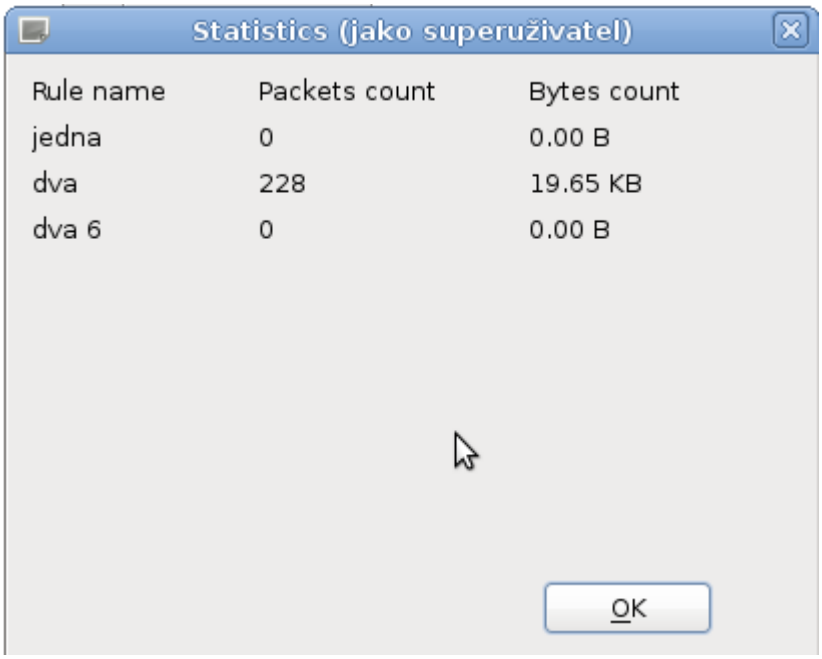
Při specifikaci pravidel je uživateli umožněno zobrazit si, jak bude vypadat výstup aplikace pro použitý framework Netfilter běžící v jádře operačního systému. Tato volba se skrývá v menu pod nabídkou *Tools – Netfilter output*. Uživateli je pak zobrazeno dialogové okno s výstupem, kde jsou výstupy pro každý modul / komponentu (iptables, ebtables, ip6tables) odděleny, viz následující obrázek.



B.4.2 Statistiky

Pro zobrazení funkčnosti jednotlivých pravidel slouží statistiky, kdy je u každého pravidla zobrazen počet „ovlivněných“ paketů a součet jejich bytů. Tato volba se skrývá v menu jako *Tools – Statistics*.

Statistiky jsou v dialogovém okně aktualizovány průběžně po 30 vteřinách.


Statistics (jako superuživatel)
Rule name
Packets count
Bytes count
jedna
0
0.00 B
dva
228
19.65 KB
dva 6
0
0.00 B
OK