VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

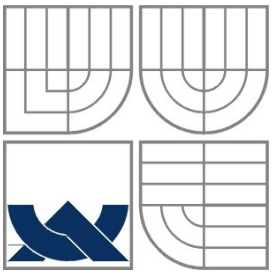
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

KLASIFIKÁCIA SIEŤOVÝCH TOKOV S VYUŽITÍM LOOK-UP PROCESSORA

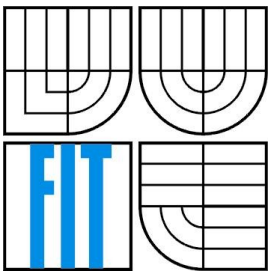
BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE JURAJ BLAHO
AUTHOR

BRNO 2007



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

KLASIFIKACE SÍŤOVÝCH TOKŮ S VYUŽITÍM LOOK-UP PROCESORU

NETWORK FLOW CLASSIFICATION USING LOOK-UP PROCESSOR

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE — JURAJ BLAHO
AUTHOR

VEDOUCÍ PRÁCE — ING. JAN KOŘENEK
SUPERVISOR

BRNO 2007

Zadání bakalářské práce/5827/2006/xblaho00

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačových systémů Akademický rok 2006/2007

Zadání bakalářské práce

Řešitel: **Blaho Juraj**
Obor: Informační technologie
Téma: **Klasifikace síťových toků s využitím look-up procesoru**
Kategorie: Alg. a datové struktury

Pokyny:

1. Seznamte se s architekturou look-up procesoru implementovanou na platformě COMBO6(X).
2. Navrhněte model filtrování síťových toků na základě údajů uvedených v unifikované hlavičce. Vytvořte syntax pro zápis filtrovacích pravidel look-up procesoru.
3. Popis filtrovacích pravidel rozložte do dvou vrstev. První vrstva bude určena pro převod pravidel zapsaných v syntaxi stávajících open-source filtrovacích nástrojů do filtrovacího jazyka look-up procesoru. Následně bude provedena transformace pravidel do hardware.
4. Ověřte funkcionalitu navrženého řešení. Proveďte kontrolní měření a srovnejte možnosti čistě SW filtrování s hardwarově akcelerovaným filtrováním pomocí look-up procesoru.

Literatura:

- Dle konzultace s vedoucím

Při obhajobě semestrální části projektu je požadováno:

- Splnění prvních 2 bodů zadání.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním paměťovém médiu (disketa, CD-ROM), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Kořenek Jan, Ing.**, UPSY FIT VUT
Datum zadání: 1. listopadu 2006
Datum odevzdání: 15. května 2007

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačových systémů a sítí
612 66 Brno, Božetěchova 2

L.S.

doc. Ing. Zdeněk Kotásek, CSc.
vedoucí ústavu

LICENČNÍ SMLOUVA
POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami

1. Pan

Jméno a příjmení: **Juraj Blaho**
Id studenta: 84457
Bytem: Rozkvet 2045/91, 017 01 Považská Bystrica
Narozen: 19. 10. 1985, Považská Bystrica
(dále jen "autor")

a

2. Vysoké učení technické v Brně

Fakulta informačních technologií
se sídlem Božetěchova 2/1, 612 66 Brno, IČO 00216305
jejímž jménem jedná na základě písemného pověření děkanem fakulty:

.....
(dále jen "nabyvatel")

Článek 1
Specifikace školního díla

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):
bakalářská práce

Název VŠKP: Klasifikace síťových toků s využitím look-up procesoru
Vedoucí/školitel VŠKP: Kořenek Jan, Ing.
Ústav: Ústav počítačových systémů
Datum obhajoby VŠKP: .....

VŠKP odevzdal autor nabyvateli v:

tištěné formě počet exemplářů: 1
elektronické formě počet exemplářů: 2 (1 ve skladu dokumentů, 1 na CD)

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.
3. Dílo je chráněno jako dílo dle autorského zákona v platném znění.
4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

Článek 2
Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizování výpisů, opisů a rozmnoženin.
2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.
3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti:
   - ☒ ihned po uzavření této smlouvy
   - ☐ 1 rok po uzavření této smlouvy
   - ☐ 3 roky po uzavření této smlouvy
   - ☐ 5 let po uzavření této smlouvy
   - ☐ 10 let po uzavření této smlouvy

   (z důvodu utajení v něm obsažených informací)
4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/ 1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

Článek 3
Závěrečná ustanovení

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.
2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.
3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísni a za nápadně nevýhodných podmínek.
4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne: .....

....

Nabyvatel Autor

Abstrakt

Táto práca sa zaoberá návrhom a implementáciou softvéra určeného na konfiguráciu vyhľadávacieho procesora, ktorý je využívaný na klasifikáciu IPv4 a IPv6 paketov. Práca popisuje algoritmy použité na prevod vstupnej množiny pravidiel do konfigurácie hardvéru. Hlavným cieľom je nastavenie vyhľadávacieho procesora pre účely filtrovania v hardvérom urýchlenom filtri NIFIC, ktorý je založený na osobnom počítači s výkonnou akceleračnou kartou COMBO6. Vykonané testy ukázali, že s použitím vytvoreného softvéra je možné filtrovať sieťové toky pri gigabitových rýchlostiach.

Kľúčové slová

klasifikácia paketov, paketový filter, vyhľadávací procesor, FPGA, COMBO6

Abstract

This work is about the design and the implementation of the software that is intended to configure the look-up processor used for an IPv4 and IPv6 packet classification. This work describes the algorithms used for the transformation of a user defined ruleset into the hardware configuration. The main goal is to set up the hardware acclerated filter called NIFIC which is based on a personal computer and the powerful acceleration card COMBO6. Performed tests show that it is possible to filter multi-gigabit network flows by using this developed software.

Keywords

packet classification, packet filter, look-up processor, FPGA, COMBO6

Citácia

Blaho, J.: *Klasifikácia sieťových tokov s využitím look-up processora, bakalárska práca*. Brno, FIT VUT v Brně, 2007.

Klasifikácia sieťových tokov s využitím look-up processora

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením Ing. Jana Kořenka. Ďalšie informácie mi poskytli kolegovia z projektu Liberouter. Uviedol som všetky literárne pramene, z ktorých som čerpal.

Juraj Blaho

15.4.2007

Obsah

1 Úvod

Vzhľadom na neustále sa zvyšujúce nároky na priepustnosť počítačových sietí, je nutné sa zaoberať zlepšovaním výkonu sieťových prvkov, akými sú napríklad smerovače, analyzátory sieťových tokov, filtre, firewally a mnoho ďalších.

V súčasnosti sa uplatňuje niekoľko prístupov k tvorbe spomenutých sieťových zariadení. Jednou možnosťou je využitie osobného počítača s niekoľkými sieťovými rozhraniami. Pakety sú prenášané cez systémovú zbernicu do operačnej pamäte, kde sú analyzované procesorom a následne sú posielané na výstupné rozhranie. Výhodou tohto riešenia je nízka cena a jednoduchá rozšíriteľnosť. Nevýhodou je nízky výpočtový výkon univerzálneho procesora pre veľké rýchlosti sieťového prenosu. Druhou alternatívou je použitie špecializovaného hardvéru, ktorého nevýhodou je ale výrazne vyššia cena a obmedzené možnosti rozširovania. V rámci projektu Liberouter je využívané riešenie, kde je osobný počítač vybavený špeciálnou akceleračnou kartou COMBO, ktorá obsahuje programovateľné hradlové pole (FPGA) a niekoľko sieťových rozhraní. Táto karta umožňuje po naprogramovaní a nakonfigurovaní spracovávať pakety bez účasti procesora. Pakety, ktoré karta nie je schopná sama spracovať môžu byť zaslané na analýzu do softvéra. Využitím tejto architektúry je možné spracovávať sieťové toky pri rýchlosti niekoľko gigabitov za sekundu.

Vo väčšine sieťových zariadení je nutné klasifikovať paket na základe jeho parametrov a určiť spôsob jeho ďalšieho spracovania. V prípade filtra by sa napríklad jednalo o rozhodnutie, či má byť paket prepustený alebo nie. Na klasifikáciu sa v rámci projektu Liberouter využíva komponenta look-up processor. Tento procesor umožňuje klasifikovať rôzne typy sieťových paketov, medzi ktoré patria napríklad TCP a UDP pakety nad IPv4 alebo IPv6.

Účelom tejto práce je návrh a implementácia konfiguračného softvéra pre vyhľadávací procesor. Softvér má umožňovať čo najpresnejšie nastaviť vyhľadávací procesor a tým zabezpečiť filtrovanie na základe čo najväčšieho počtu atribútov. Výsledná konfigurácia musí byť vytvorená tak, aby bola dosiahnutá vysoká priepustnosť procesora. Ďalšou požiadavkou je, aby nastavovanie hardvérom urýchleného filtra pomocou tohto softvéra pripomínalo nastavovanie niektorého z existujúcich voľne dostupných softvérových filtrov.

Tento dokument je rozdelený do niekoľkých častí. Hneď za úvodom nasleduje teoretický rozbor stručne popisujúci problematiku filtrovania a klasifikácie paketov. Následne sú porovnané existujúce voľne dostupné softvérové riešenia filtrov. Ďalej je v teoretickom rozbore stručne popísaná architektúra a princíp činnosti vyhľadávacieho procesora, ktorý reprezentuje jednu z možností klasifikácie. Za teoretickým rozborom nasleduje hlavná časť tohto dokumentu, ktorá popisuje vytvorený softvér, vysvetľuje jeho celkovú architektúru a účel jednotlivých súčastí. Taktiež vysvetľuje princípy použitých algoritmov. Predposledná kapitola pojednáva o výsledkoch dosiahnutých použitím vytvoreného softvéra. V závere sú zhodnotené dosiahnuté ciele a navrhnuté možné vylepšenia do budúcnosti.

2 Teoretický rozbor

2.1 Klasifikácia paketov

Pre činnosť mnohých sieťových zariadení, akými sú napríklad smerovače alebo filtre, je potrebné vykonávať klasifikáciu paketov. Jedná sa o proces, pri ktorom sú jednotlivé pakety podľa svojich parametrov zaradené do určitej skupiny. Na základe tejto skupiny je následne s paketom vykonaná príslušná akcia, ktorá závisí na type sieťového zariadenia. V prípade filtra by akcia určovala napríklad zablokovanie alebo prepustenie paketu, v prípade iného zariadenia, akým je napríklad smerovač, by akciou bolo smerovanie paketu na určitý výstup.

V ďalšom texte budú často používané pojmy pravidlo a podmienka, preto je nutné ujednotiť si ich význam. Pravidlo určuje triedu, prípadne akciu, ktorá je paketu priradená, pokiaľ sú splnené všetky podmienky daného pravidla. Podmienka testuje určité vlastnosti paketu, a môže byť buď splnená alebo nesplnená. Paket vyhovuje pravidlu, práve vtedy ak sú splnené všetky podmienky.

Klasifikácia je riadená množinou pravidiel, ktorá by mala byť zostavená tak, aby každý paket vyhovoval aspoň jednému pravidlu, preto množina obvykle obsahuje pravidlo, ktorému vyhovujú úplne všetky pakety. Pokiaľ tomu tak nie je a nájde sa paket, ktorý nevyhovuje žiadnemu pravidlu z množiny, musí klasifikátor zaradiť paket podľa nejakého implicitného pravidla.

Môžeme rozlíšiť niekoľko druhov podmienok. Najjednoduchším typom na spracovanie je test na zhodu s konštantou. Ďalej existujú podmienky testujúce príslušnosť do určitého rozsahu alebo množiny hodnôt. Niektoré množiny, prípadne rozsahy, sa dajú s výhodou zapísať ako test na zhodu s konštantou pri použití bitovej masky. Toto môže mať výrazný vplyv na výkon klasifikácie. Napríklad podmienka na 16-bitové nezáporné číslo väčšie alebo rovné ako 0x8000 môže byť prevedená na test skúmajúci, či je nastavený najvyšší bit.

2.1.1 Priorita pravidiel

V prípade, že paket vyhovuje viacerým pravidlám, je nutné určiť, ktoré pravidlo bude skutočne použité. I keď existuje viacero riešení, obvykle je priorita pravidla odvodená od jeho poradia v zozname pravidiel. Dá sa rozlíšiť niekoľko prístupov k tomuto problému.

Filter typu first-match použije prvé pravidlo, ktorému paket vyhovuje. Zástupcom tohto typu je napríklad ipfw [8]. Príklad spracovania pravidiel filtrom typu first-match je demonštrovaný na nasledujúcej množine pravidiel.

ID	Zdrojová IP	Cieľová IP	Akcia
R1	147.229.200.10	147.229.0.0/24	Pass
R2	147.229.200.10	*	Block
R3	147.229.0.0/24	*	Pass
R4	*	*	Block

Tabuľka 1: Príklad filtra typu first-match

Podľa týchto pravidiel budú pakety vyhodnotené tak, ako je naznačené v tabuľke číslo 2. Tabuľka obsahuje parametre paketov, ktorými sú v tomto prípade zdrojové a cieľové adresy. Pre

každý paket je v poslednom stĺpci vyznačené použité pravidlo a výsledná akcia. Napríklad paket P3 nevyhovuje pravidlu R1 a R2, pretože má rozdielnu zdrojovú IP adresu. Prvé pravidlo, ktorému vyhovuje, je pravidlo R3, preto bude toto použité a výsledná akcia bude Pass.

ID	Zdrojová IP	Cieľová IP	Pravidlo/Akcia
P1	147.229.200.10	213.151.10.18	R2/Block
P2	147.229.200.10	147.229.147.2	R1/Pass
P3	147.229.30.5	147.229.147.2	R3/Pass

Tabuľka 2: Vyhodnotenie paketov vo filtri typu first-match

Naopak filter typu last-match spracováva postupne všetky pravidlá a použije posledné, ktorému paket vyhovoval. V tomto type filtra je ďalej možné použiť takzvané quick príznaky, ktoré znamenajú, že pokiaľ sa pri vyhodnocovaní narazí na takéto pravidlo a paket mu vyhovuje, je toto pravidlo považované za posledné vyhovujúce a automaticky sa použije bez vyhodnocovania ďalších. Príkladom takéhoto filtra je OpenBSD Packet Filter. Last-match filter, ktorý by obsahoval iba quick pravidlá je vlastne ekvivalentný s first-match filtrom. V tabuľke číslo 3 sa nachádza postupnosť pravidiel, na ktorej bude ukázaný postup vyhodnocovania filtrom typu last-match s quick. Je významom ekvivalentná tabuľke z predchádzajúceho príkladu o filtri first-match, teda na rovnaké pakety bude aplikovaná rovnaká akcia.

ID	Quick	Zdrojová IP	Cieľová IP	Akcia
R1		*	*	Block
R2	QUICK	147.229.200.10	147.229.0.0/24	Pass
R3	QUICK	147.229.200.10	*	Block
R4		147.229.0.0/24	*	Pass

Tabuľka 3: Príklad filtra typu last-match s quick

Pakety budú vyhodnotené tak, ako je zapísané v nasledujúcej tabuľke. Význam jednotlivých položiek je zhodný s ich významom v tabuľke 2. Paket P2 vyhovuje pravidlu R1, ale ako bolo už spomenuté, vyhodnotenie sa nájdením vyhovujúceho pravidla, ktoré nie je quick, neukončí hneď, ale toto pravidlo je iba zapamätané ako posledné vyhovujúce a pokračuje sa v klasifikácii. Paket P2 ďalej nevyhovuje pravidlu ani R2, ale vyhovuje R3. A keďže má pravidlo R3 nastavený príznak quick, bude považované za posledné vyhovujúce a vyhodnotenie sa hneď ukončí. Paket P1 nevyhovuje žiadnemu z pravidiel R2, R3 a R4, preto posledné vyhovujúce pravidlo je R1. Nakoniec paket P4 vyhovuje pravidlám R1 a R4, z ktorých ani jedno nie je quick, a teda posledné vyhovujúce je R4.

ID	Zdrojová IP	Cieľová IP	Pravidlo/Akcia
P1	213.151.10.18	147.229.200.10	R1/Block
P2	147.229.200.10	213.151.10.18	R3/Block
P3	147.229.200.10	147.229.147.2	R2/Pass
P4	147.229.147.2	213.151.10.18	R4/Pass

Tabuľka 4: Vyhodnotenie paketov vo filtri typu last-match

Všetky spomenuté typy určovania priority pravidla (first-match, last-match a last-match s quick) majú rovnakú vyjadrovaciu schopnosť a je možné ich vzájomne previesť na iný typ. V prípade čistých first-match a last-match filtrov prevod znamená iba obrátenie poradia pravidiel. Pri použití last-match s quick je prevod na first-match poradie o trochu zložitejší.

2.2 Používané paketové filtre

V súčasnosti neexistuje ustálená definícia pojmu paketový filter. Pre účely tohto dokumentu bude jeho činnosť definovaná ako blokovanie, prípadne prepúšťanie paketov najmä na základe informácií z ich hlavičiek. Ďalšími kritériami môže byť napríklad identifikátor rozhrania, na ktorom bol paket prijatý. Filtre vždy obsahujú nejaký klasifikačný mechanizmus, ktorý je riadený súborom pravidiel vytvoreným administrátorom.

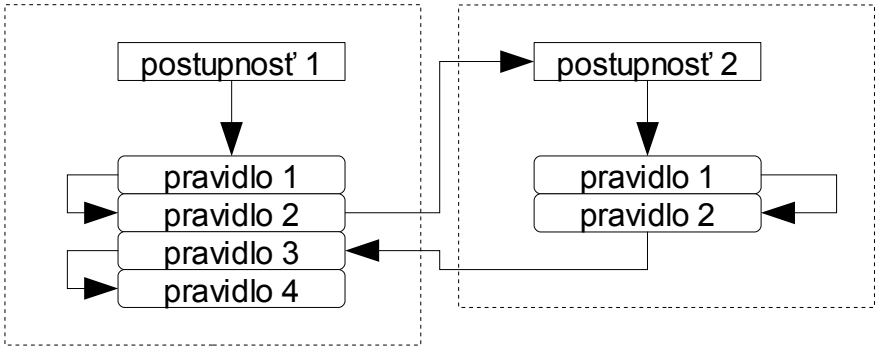
Pre nasadenie hardvérom urýchleného filtra do praxe je nutné zvoliť taký formát popisu pravidiel, ktorý by bol podobný nejakému inému používanému popisu, čím by sa skrátila doba potrebná na to, aby sa administrátor naučil konfigurovať tento filter. Medzi momentálne najpoužívanejšie paketové filtre patria iptables/ipchains [5] pracujúce na Linuxe, ipfw [8] na FreeBSD a Mac OS X a packet filer [6] na OpenBSD.

V tejto kapitole sú popísané základné princípy iptables a OpenBSD packet filtra. V závere sú formáty konfigurácie týchto filtrov porovnané z hľadiska vhodnosti pre popis hardvérom urýchleného filtra.

2.2.1 Iptables

Iptables je systém na filtrovanie paketov používaný na operačnom systéme Linux. Konfigurácia prebieha obvykle pomocou shellovských skriptov. Aktuálna konfigurácia sa dá uložiť (iptables-save) do súboru odkiaľ je ju potom možné obnoviť (iptables-restore).

Iptables klasifikuje pakety hlavne na základe IP hlavičiek, ale je schopný filtrovať aj podľa informácií z vyššej (TCP, UDP hlavičky, ...) alebo nižšej vrstvy (MAC adresa). Na rozhodnutie, čo sa má s paketom vykonať, sa využíva niekoľko tabuliek. V týchto tabuľkách sa nachádzajú pomenované postupnosti pravidiel, ktoré môže užívateľ vytvárať, meniť, premenovávať a mazať. Pravidlá sa skladajú z podmienky a akcie. Ak je podmienka úplne splnená, vyvolá sa akcia, ktorou môže byť napríklad prijatie alebo odmietnutie paketu, prípadne skok do inej tabuľky alebo postupnosti podmienok. Vyhodnocovanie pravidiel prebieha v poradí tak, ako je znázornené na obrázku až kým niektoré pravidlo neurčí akciu, ktorá sa má s paketom vykonať.



Obrázok 1: Príklad klasifikácie v iptables

Keďže sa pravidlá pridávajú pomocou parametrov programu iptables, nemajú fixné poradie atribútov. Odporúča sa ale používať nasledujúci predpis:

```
iptables [-t table] command [match] [-j target/jump]
```

-t table

Obvykle nie je nutné zadávať tabuľku, s ktorou sa pracuje a je implicitne použitá tabuľka filter. Táto je určená výhradne na filtrovanie a obsahuje tri postupnosti pravidiel: INPUT, FORWARD a OUTPUT. Ďalšími tabuľkami, ktoré je možné použiť, sú: nat, mangled a raw.

command

Parameter command určí, čo má iptables urobiť so zadaným pravidlom. Môže sa jednať o pridanie, odobranie alebo zmenu pravidla, prípadne o vytvorenie alebo zrušenie postupnosti pravidiel v danej tabuľke. Napríklad pre pridanie pravidla na koniec postupnosti INPUT sa použije príkaz -A INPUT.

match

Množina parametrov match obsahuje konkrétne podmienky, ktoré musí paket splniť, aby bolo pravidlo použité. Takýmito parametrami sú napríklad --dport a --sport, ktoré vyjadrujú podmienky na cieľový a zdrojový port.

-j target/jump

Poslednou časťou zápisu je akcia, ktorá sa má vykonať pokiaľ sú všetky podmienky splnené. Môže sa jednať určenie spracovania paketu alebo skok do inej postupnosti pravidiel.

Príklad pridania pravidla do postupnosti INPUT v tabuľke filter, ktoré prepustí tcp paket určený na cieľový port 22:

```
iptables -A INPUT -p tcp --dport 22 -j ACCEPT
```

2.2.2 OpenBSD Packet Filter

Tento filter bol pôvodne vyvinutý na OpenBSD, ale v súčasnosti je dostupný aj na iných distribúciach BSD (FreeBSD, NetBSD). Pravidlá bývajú uložené v textovej forme v súbore /etc/pf.conf, ale umiestnenie sa dá zmeniť.

Kritéria, ktoré sa používajú na kontrolu paketov, sú založené na hlavičkách z tretej (IPv4, IPv6) a štvrtej vrstvy (TCP, UDP, ICMP, ICMPv6) modelu ISO/OSI.

Filtrovacie pravidlá špecifikujú podmienky, ktoré musí paket splniť a výslednú akciu, ktorou je napríklad blokovať alebo prepustiť. Jedná sa o filter typu last-match, čo znamená, že pravidlá sú vyhodnocované v sekvenčnom poradí, od prvého po posledné. Ak podmienka neobsahuje kľúčové slovo quick, bude na určenie výslednej akcie použité posledné splnené pravidlo. V prípade, že vyhovujúce pravidlo toto kľúčové slovo obsahuje, bude toto pravidlo považované za posledné vyhovujúce. Pokiaľ žiadne pravidlo nesplňuje podmienku, bude paket implicitne prepustený.

Niektoré testované vlastnosti paketu je možné zadávať nie len pomocou konkrétnej hodnoty, ale aj pomocou rozsahu alebo zoznamu hodnôt. Zjednodušená syntax pravidla:

```
action [direction] ['log'] ['quick'] ['on' interface] [af] ['proto'
  protocol] ['from' src_addr ['port' src_port]] ['to' dst_addr
  ['port' dst_port]] ['flags' tcp_flags] [state]
```

Nasledujúci popis jednotlivých častí pravidla nevysvetľuje presnú syntax, ale slúži iba ako ukážka možností OpenBSD packet filtra.

action

Akcia, ktorá má byť vykonaná pokiaľ paket vyhovuje pravidlu. V prípade použitia kľúčového slova pass, bude paket prepustený na ďalšie spracovanie v operačnom systéme. Pokiaľ je akcia block, filter sa zachová podľa blokovacej politiky a paket buď ticho zahodí, alebo oboznámi odosielateľa, že paket bol odmietnutý.

direction

Smer, ktorým sa paket pohybuje na rozhraní, je určený buď kľúčovým slovom in alebo out.

'log'

Znamená, že pakety majú byť zaznamenávané pomocou démona pflogd.

'quick'

Pokiaľ paket vyhovuje pravidlu, ktorého zápis obsahuje kľúčové slovo quick, potom je toto pravidlo považované za posledné vyhovujúce, ihneď sa ukončí klasifikácia a vykoná sa príslušná akcia.

interface

Meno alebo skupina sieťových rozhraní.

af

Rodina adries, je určená slovom inet pre adresy typu IPv4, alebo slovom inte6 pre adresy IPv6. Packet filter obvykle dokáže rodinu adries určiť sám.

protocol

Protokol transportnej vrstvy. Môže byť zadaný buď názvom protokolu alebo číslom medzi 0 a 255. Názvy a k nim zodpovedajúce čísla sú na unixových operačných systémoch uložené v súbore /etc/protocols.

src_addr, dst_addr

Tieto parametre špecifikujú množiny zdrojových a cieľových IPv4 alebo IPv6 adries. Ako adresu je možné použiť aj prefix siete. Počet bitov prefixu je určený zápisom /width. Napríklad hodnota 192.168.1.0/24 je prefix siete so šírkou 24 bitov. Kľúčové slovo any znamená ľubovoľnú adresu.

src_port, dst_port

Množiny zdrojových a cieľových portov z hlavičiek TCP a UDP paketov. Porty môžu byť zadané aj pomocou mena služby. Zoznam názvov služieb a príslušných portov je uložený v súbore /etc/services.

tcp_flags

Špecifikácia príznakov z TCP hlavičiek. Príznaky sú zadané zápisom nastavené/overované. Takže zápis S/SA znamená kontrolu stavu príznakov SYN a ACK, kde iba SYN je nastavený.

state

Určuje, či sa má udržiavať stav pre pakety vyhovujúce danému pravidlu.

Nasledujúci príklad je zápis pravidla umožňujúceho prechod paketu smerom von na rozhraní dc0, a to iba z adresy 192.168.0.1 na adresy zadané pomocou prefixu siete 192.168.0.0/24:

```
pass out on dc0 from 192.168.0.1 to 192.168.0.0/24
```

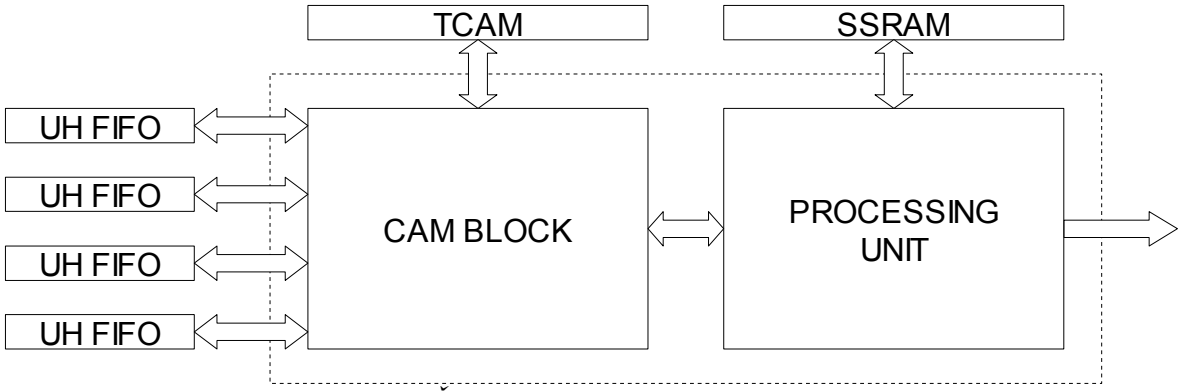
2.2.3 Porovnanie

Nevýhodou iptables oproti popisu OpenBSD Packet Filtra je, že konfigurácia prebieha pomocou skriptov operačného systému a neexistuje žiadna pevná syntax pravidiel. Ďalšou nevýhodou je, že poradie vyhodnotenia pravidiel je určené niekoľkými tabuľkami. Spracovanie takéhoto popisu filtra podobného iptables by bolo príliš komplikované. Z tohto dôvodu bola vybraná ako vstupný formát podmnožina jazyka pre popis OpenBSD Packet Filtra.

2.3 Vyhľadávací procesor

Vyhľadávací procesor (Look-Up Processor - LUP) je hardvérová komponenta implementovaná v FPGA, slúžiaca na klasifikáciu paketov. Je využívaný v rôznych projektoch Liberouteru. Dokáže klasifikovať na základe informácií z druhej, tretej a štvrtej vrstvy modelu ISO/OSI a je schopný pracovať s IPv4 aj IPv6. Jeho vstupom je štruktúra nazvaná unifikovaná hlavička vytvorená komponentou header file extractor (HFE). Výstupom klasifikácie je záznam určujúci triedu paketu, a tým aj akciu, ktorá sa s paketom následné vykoná. Význam konkrétnych hodnôt záznamu je rôzny v závislosti na aplikácii.

Ďalší obrázok znázorňuje základnú blokovú štruktúru vyhľadávacieho procesora a s ním súvisiacich komponent, ktorými sú štyri vstupné fronty, asociatívna a statická pamäť. Šípky vyjadrujú toky dát, prípadne požiadavkov medzi komponentami. Samotný procesor je vyznačený v obdĺžniku ohraničenom prerušovanou čiarou. Skladá sa z bloku asociatívnej pamäte (Content-Addressable Memory - CAM) a sekvenčnej jednotky (Processing Unit). Obidve tieto časti sú konfigurovateľné zo softvéra.



Obrázok 2: Štruktúra vyhľadávacieho procesora

2.3.1 Unifikovaná hlavička

Unifikovaná hlavička je štruktúra zložená z 16-bitových registrov. Obsahuje dôležité informácie z hlavičiek paketov. Medzi tieto informácie patria napríklad zdrojové a cieľové MAC adresy, IP adresy, porty, informácie o triede protokolu, číslo protokolu a ďalšie. Jeden register s daným číslom môže obsahovať rôzne informácie, napríklad časť adresy IPv4 alebo adresy IPv6, prípadne môže byť jeho hodnota nedefinovaná, v závislosti na triede protokolu paketu. Konkrétne štruktúry unifikovanej hlavičky pre rôzne typy sieťového toku sú znázornené v prílohách v tabuľke č. 9.

2.3.2 Sekvenčná jednotka

Sekvenčná jednotka je riadená programom uloženým v jej statickej pamäti. Tento program prevažne obsahuje inštrukcie podmieneného skoku na základe porovnania jednotlivých registrov unifikovanej hlavičky. Okrem toho je možné niekoľkokrát vyvolať vyhľadanie v asociatívnej pamäti. Výsledok vyhľadania vždy určí adresu inštrukcie, od ktorej má program ďalej pokračovať. Analýza končí inštrukciou EXE, ktorej parametrom je číslo definujúce ďalšie spracovanie paketu. Sekvenčná jednotka umožňuje použiť tieto tieto inštrukcie:

CAM Set

Touto inštrukciou je vyvolané vyhľadanie v asociatívnej pamäti. Parameter Set určuje sadu registrov unifikovanej hlavičky, ktorých hodnoty majú byť vyhľadávané. Po vyhodnotení je nastavený obsah programového čítača v závislosti na čísle prvého riadka asociatívnej pamäti, na ktorom došlo k zhode.

JMP Addr

Inštrukcia nepodmieneného skoku. Nahradí obsah programového čítača hodnotou parametra Addr. Tým zabezpečí pokračovanie programu na inštrukcii nachádzajúcej sa na danej adrese.

CMP Step, Reg, Mask, Const

Inštrukcia testu na zhodu. Na register Reg sa aplikuje bitová maska Mask a výsledná hodnota sa porovná s konštantou Const. V prípade zhody sa hodnota programového čítača zväčší o hodnotu Step, v opačnom prípade sa čítač inkrementuje iba o jednotku. Hodnota parametra Step nie je ľubovoľná a riadi sa tabuľkou číslo 11.

Jxxx Step, Reg, Const

Jedná sa o skupinu inštrukcií podmieneného skoku. V prípade, že porovnanie registra Reg s konštantou Const dopadne podľa zadanej podmienky, je k hodnote programového čítača pripočítaná hodnota parametra Step. V opačnom prípade je programový čítač inkrementovaný o jedna. Pre parameter Step platí rovnaké obmedzenie ako v prípade inštrukcie CMP, teda sú možné iba hodnoty uvedené v tabuľke číslo 11. Názov konkrétnej inštrukcie Jxxx dostaneme podľa nasledujúcej tabuľky nahradením písmen xxx.

	celý 16-bitový register	spodných 8-bitov registra
väčší	GTH	BGT
väčší alebo rovný	GTE	BGE
menší	LTH	BLT
menší alebo rovný	LTE	BLE

Tabuľka 5: Názvy inštrukcií podmieneného skoku

EXE Rec

Touto inštrukciou sa ukončuje analýza paketu. Parameter Rec je číslo určujúce ďalšie spracovanie paketu.

2.3.3 Asociatívna pamäť

V asociatívnej pamäti sa vyhľadáva kľúč vytvorený na základe hodnoty unifikovanej hlavičky a čísla sady porovnávaných registrov. Asociatívna pamäť je zložená z riadkov o šírke 272 bitov. Ku každému riadku dát náleží riadok bitovej masky s rovnakou šírkou. To umožňuje určiť bity dát, ktoré sa majú pri porovnaní brať do úvahy, a ktoré sa majú ignorovať. Porovnávajú sa iba bity, na ktorých pozícií je maska nastavená na jednotku. Ostatné sa považujú automaticky za zhodné.

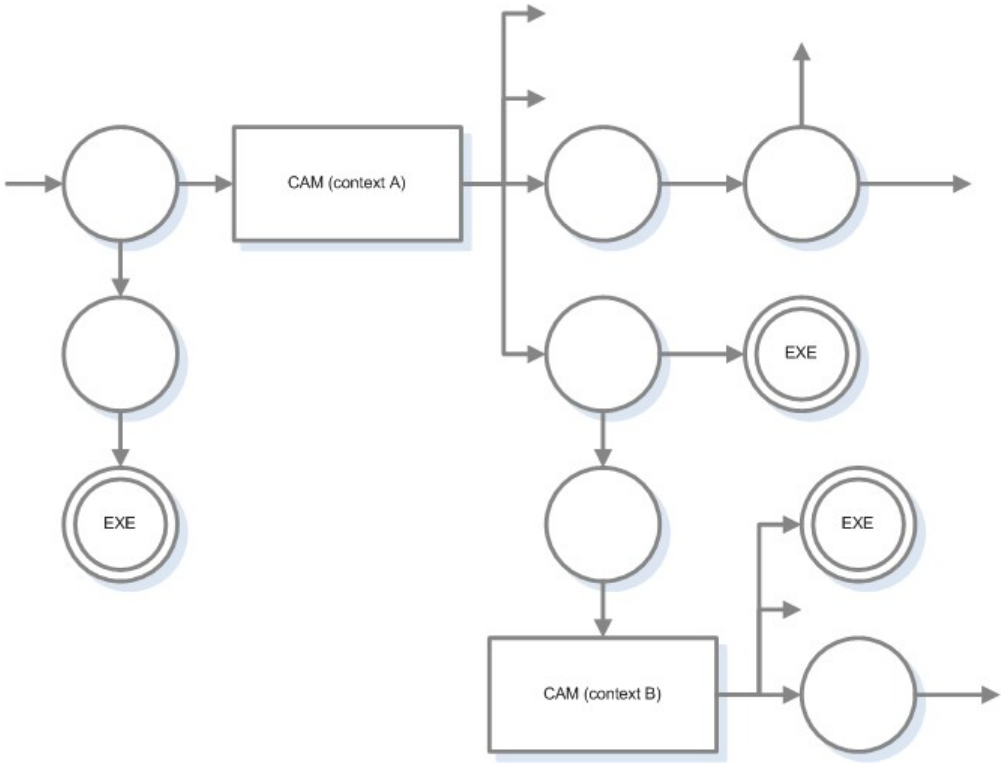
Z celkovej šírky riadka je pre porovnávané dáta vyhradených 256 bitov. Vzhľadom na to, že v asociatívnej pamäti sa dajú registre unifikovanej hlavičky adresovať po dvojiciach, teda po 32 bitoch, umožňuje táto šírka vyhodnotiť naraz až 8 takýchto 32-bitových blokov. Dajú sa adresovať iba zarovnané dvojice, čo znamená, že adresa nižšieho 16-bitového registra je deliteľná dvomi. To, ktoré registre unifikovanej hlavičky sa budú porovnávať, s ktorými časťami riadka asociatívnej pamäte, je nastaviteľné v registroch vyhľadávacieho procesora. Dá sa nastaviť niekoľko takýchto sád porovnávaných blokov unifikovanej hlavičky, takzvaných kontextov. Tým je umožnené viacnásobné využitie asociatívnej pamäte a porovnávanie zakaždým iných parametrov paketu. Na určenie sady, s ktorou sa má daný riadok porovnávať, je na riadku vyhradených 8 bitov obsahujúcich číslo príslušnej sady. Vyhľadávací procesor zabezpečí, že na danom mieste hľadaného kľúča sa tiež nachádza číslo sady, a teda nie je možné, aby došlo k úplnej zhode kľúča s riadkom, ktorý má na danom mieste inú hodnotu.

Vyhľadanie v asociatívnej pamäti je vyvolané vždy len zo sekvenčnej jednotky inštrukciou CAM. Výsledkom vyhľadania je číslo prvého riadka, na ktorom došlo k zhode. Toto číslo je vynásobené konštantou a výsledok sa použije ako adresa inštrukcie, od ktorej bude pokračovať vyhodnocovanie v sekvenčnej jednotke. Keďže je unifikovaná hlavička konštantná, nemá význam opakované vyhľadanie s použitím jednej sady registrov a spôsobilo by zacyklenie, pretože zakaždým by bol nájdený rovnaký riadok.

2.3.4 Princíp klasifikácie

Vyhodnotenie každej unifikovanej hlavičky začína v sekvenčnej jednotke odkiaľ je možné vyvolať niekoľko vyhľadaní v asociatívnej pamäti. Je vhodné klasifikáciu rozdeliť medzi spomenuté komponenty tak, že testy na zhodu budú vyhodnocované pomocou asociatívnej pamäte a kontroly rozsahov programom v sekvenčnej jednotke.

Princíp klasifikácie je znázornený na nasledujúcom obrázku. Obdĺžniky znázorňujú vyhľadanie v asociatívnej pamäti, kruhy a šípky spracovanie v sekvenčnej jednotke. Kruhy s dvojitým okrajom značia ukončenie klasifikácie. Vyhľadanie v asociatívnej pamäti predstavuje n-árne vetvenie a naproti tomu spracovanie inštrukcie v sekvenčnej jednotke vetví klasifikáciu binárne. Ako vidno na obrázku, jednotlivé uzly grafu, znázorňujúceho priebeh klasifikácie, môžu byť takmer ľubovoľne pospájané s podmienkou, že výsledný graf nebude obsahovať cykly. Tie sú neprípustné, pretože počas vyhodnotenia jedného paketu sú všetky hodnoty registrov unifikovanej hlavičky konštantné, a teda aj výsledky porovnaní sa nemenia. Z tohto dôvodu by bol každý cyklus vykonávaný donekonečna.



Obrázok 3: Princíp klasifikácie pomocou vyhľadávacieho procesora

2.3.5 Konfigurácia vyhľadávacieho procesora

Vytvorený graf klasifikácie je transformovaný na obsah asociatívnej pamäte a program sekvenčnej jednotky. Samotnú konfiguráciu vyhľadávacieho procesora môžeme rozdeliť do troch častí. Prvou časťou je nastavenie sekvenčnej jednotky, ktoré zahŕňa nahratie programu do statickej pamäte a nastavenie adresy prvej inštrukcie.

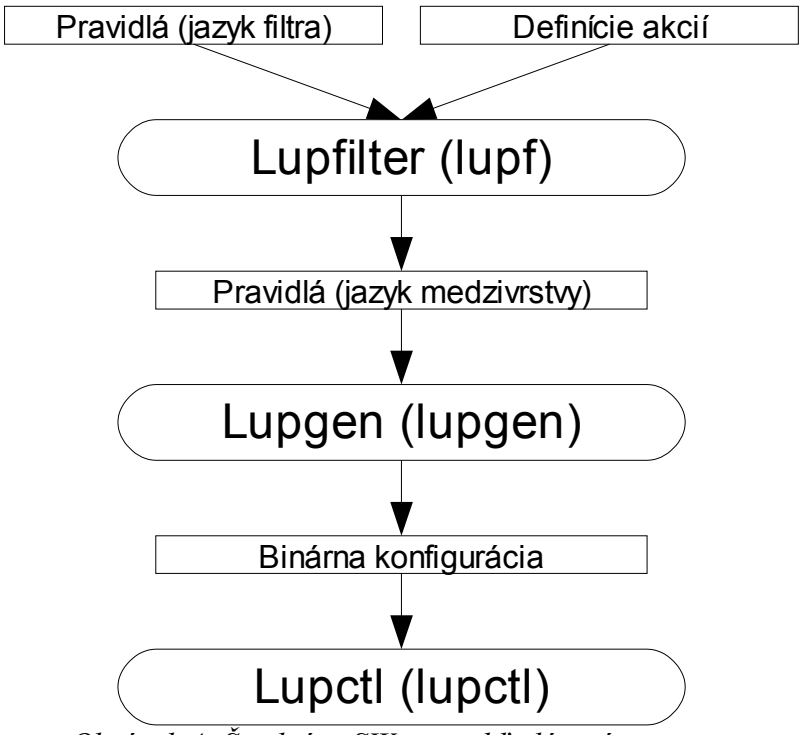
Druhá časť je nastavenie bloku asociatívnej pamäte. Tu je potrebné nahrať potrebné dáta a masky do asociatívnej pamäte a nastaviť kontextové registre určujúce, ktoré časti unifikovanej hlavičky sa budú vyhľadávať. Ostáva ešte inicializovať kontrolnú pamäť, indexové pamäte a nastaviť adresu, kam budú ukladaná identifikácia paketu.

Posledná fáza je samotné spustenie bloku asociatívnej pamäte a sekvenčnej jednotky.

3 Konfiguračný softvér

V rámci tejto práce bol vytvorený softvér, ktorý na základe vstupnej množiny filtrovacích pravidiel vytvorí konfiguráciu pre vyhľadávací procesor a následne túto konfiguráciu nahrá do príslušných pmätí.

Kvôli lepšej znovupoužiteľnosti bol softvér rozdelený do troch vrstiev. Podľa miery abstrakcie sú názvy jednotlivých vrstiev od najvyššej po najnižšiu: lupfilter, lupgen a lupctl. Výstupom každej vrstvy je konfiguračný súbor pre nižšiu vrstvu. Toho sa dá využiť tak, že stačí vždy napísať konfiguračné súbory iba pre jednu vrstvu a postupne transformovať túto konfiguráciu. Výstupom najnižšej vrstvy lupctl je nastavený vyhľadávací procesor. Na nasledujúcom obrázku je znázornená štruktúra konfiguračného softvéra. Obdĺžniky znázorňujú konfiguračné súbory. Ovály reprezentujú príslušné vrstvy, kde v zátvorke je uvedený názov konkrétneho programu pracujúceho na danej vrstve. Šípky naznačujú, čo je vstupom a výstupom jednotlivých programov/vrstiev.



Obrázok 4: Štruktúra SW pre vyhľadávací procesor

Lupfilter

Vrstva lupfilter je navrhnutá s ohľadom na jednoduchosť nastavovania filtra a slúži ako pohodlné užívateľské rozhranie. Kvôli tejto špecializácii však pomocou nej nie je možné nastaviť vyhľadávací procesor ľubovoľným spôsobom, a tak, v prípade potreby využitia procesora pre iný účel ako filtrovanie, je nutné použiť nižšiu vrstvu – lupgen.

Lupfilter zabezpečuje zjednodušenie pravidiel do disjunktívnej normálnej formy, čo znamená prevod zložitých pravidiel obsahujúcich logickú spojku 'alebo' (OR) na pravidlá výhradne s logickými spojkami 'a zároveň' (AND). Prakticky sa jedná o to, aby výsledné podmienky mohli

byť zapísané pomocou testu na zhodu alebo pomocou jednoduchého porovnania, teda aby podmienkou nebol test príslušnosti do nejakej množiny. Vrstva ďalej zabezpečuje prevod postupnosti pravidiel na poradie typu first-match.

Transformačný program pracujúci na tejto vrstve je nazvaný lupf. Na vstupe vyžaduje dva súbory. Jedným je popis filtrovacích pravidiel, ktoré sú zapísané v jazyku podobnom OpenBSD PF. V druhom sa nachádza definícia akcií, ktoré sa môžu s rôznymi paketmi vykonať a konkrétne číselné hodnoty týchto akcií zodpovedajúce významom a hodnotám záznamov na výstupe vyhľadávacieho procesora. Podrobnosti o formátoch týchto súborov je možné nájsť v prílohách.

Lupgen (medzivrstva)

Lupgen je tiež označovaný pojmom medzivrstva. Slúži ako univerzálne rozhranie na generovanie konfigurácie vyhľadávacieho procesora. Vďaka tomu je možné vytvárať nad touto vrstvou rôzne nadstavby, akou je napríklad lupfilter. Najzložitejšou úlohou lupgenu je zaručiť sekvenčné a zároveň efektívne vyhodnotenie pravidiel. Lupgen takisto zabezpečuje výber registrov, ktoré budú vyhodnocované v asociatívnej pamäti, a ktoré v sekvenčnej jednotke. Nakoniec zaisťuje tiež generovanie binárnej konfigurácie procesora.

Pre zápis pravidiel je vytvorený špeciálny jazyk, nazvaný jazyk medzivrstvy. Ten musí umožniť adresovanie a porovnávanie všetkých registrov unifikovanej hlavičky a taktiež testovanie jednotlivých bitov týchto registrov. Podrobný popis jazyka je v prílohách. Vstupom programu lupgen je jediný súbor obsahujúci klasifikačné pravidlá zapísané v tomto jazyku. Výstupný súbor obsahuje textový zápis binárnej konfigurácie, ktorú je možné nástrojom lupctl priamo, bez ďalších transformácií, nahrať do vyhľadávacieho procesora.

Lupctl

Najnižšia vrstva konfiguračného softvéra je reprezentovaná nástrojom lupctl. Pre svoju činnosť vyžaduje podporné nástroje ssramctl a tcamctl slúžiace na nahrávanie obsahov statickej pamäte sekvenčnej jednotky a asociatívnej pamäte. Ovládanie týchto nástrojov je ale pod plnou kontrolou lupctl, a preto sa užívateľ nimi nemusí zaoberať.

Okrem nahrávania obsahov jednotlivých pamätí, slúži lupctl na zapisovanie a čítanie kontrolných, stavových a ladiacich registrov, a tým na zadávanie príkazov procesora a zisťovanie jeho stavu. Medzi možné príkazy patria napríklad zastavenie alebo spustenie procesora.

Vstupný súbor obsahujúci binárny obraz konfigurácie je nevhodné akokoľvek ručne upravovať alebo vytvárať, z dôvodu vysokej pravdepodobnosti vzniku chyby a mal by byť vždy generovaný nejakým nástrojom, napríklad lupgenom.

3.1 Spracovanie jazyka filtra

Transformácia na tejto vrstve je rozdelená do niekoľkých fáz. Najskôr sú analyzované pravidlá z vstupného súbora. Následne sú pravidlá preusporiadané a zjednodušené do takej formy aby boli spracovateľné nižšou vrstvou.

3.1.1 Reprezentácia pravidiel

Zoznam pravidiel je usporiadaná n-tica $R=[r_1, r_2, ..., r_n]$. Priorita pravidiel je typu last-match s quick. Jednotlivé pravidlá popisujúce filter sú reprezentované trojicou $r=[a, q, C]$, kde a je akcia, ktorá sa vykoná pokiaľ sú splnené všetky podmienky $c \in C$. Príznak q, určuje či je pravidlo

quick. Podmienky sú dvojice $c=[attr, RANGES]$, kde $RANGES$ je neprázdna množina rozsahov. Rozsah $range \in RANGES$ je interval celých čísel $x \in range$. Môže byť predstavovaný testom na rovnosť $x = const/width$ alebo nerovnosť $x \neq const/width$. V týchto prípadoch je možné určiť počet bitov zľava $width$, ktoré sa budú porovnávať. Toto sa využíva najmä na zadávanie prefixov sietí. Rozsah môže byť taktiež reprezentovaný jednostranným obmedzením $x > const$, $x \geq const$, $x < const$, $x \leq const$ alebo obojstranným obmedzením $const_{low} < x < const_{hi}$, $const_{low} \leq x \leq const_{hi}$. Podmienka je splnená, ak $attr$ patrí aspoň do jedného rozsahu z danej množiny $\exists range, range \in RANGES \wedge attr \in range$. Symbol $attr$ zastupuje testovanú vlastnosť paketu. Tou môže byť zdrojová a cieľová IP adresa. Sú podporované adresy IP verzie 4 aj 6. Ďalej je nožné porovnávať zdrojový a cieľový port, číslo protokolu a číslo vstupného rozhrania. V pravidle sa daná vlastnosť vyskytuje maximálne v jednej podmienke, teda platí:

$$\forall (c_i=[attr_i, RANGES_i]) \in C, \forall (c_j=[attr_j, RANGES_j]) \in C : i \neq j \Rightarrow attr_i \neq attr_j .$$

3.1.2 Tabuľky

Tabuľka je na vstupe zapísaná množinou konštánt so zadanou šírkou porovnávania. Ku každej konštante náleží operátor rovnosti alebo nerovnosti. Príkladom tabuľky IP adries je množina $T=\{=172.16.0.0/16, \neq 172.16.5.0/24, =172.16.5.100/32\}$. Podmienka určená tabuľkou $c=[attr, T]$ vyhodnotí atribút paketu vzhľadom na najširšiu zhodnú konštantu. Spomenutá podmienka c je napríklad splnená pre IP adresu 172.16.1.254, kde najširšia zhodná konštanta je 172.16.0.0/16. Pre 172.16.5.100 je najširšia zhodná konštanta 172.165.5.100, a preto táto hodnota tiež splňuje podmienku. Adresa 192.168.15.1 nevyhovuje, lebo neexistuje žiadna hodnota v tabuľke, s ktorou by sa zhodovala. Adresa 172.16.5.99 taktiež nevyhovuje, lebo najširšia zhodná konštanta 172.16.5.0/24 je vylúčená operátorom nerovnosti.

Z predchádzajúceho textu vyplýva, že spracovanie podmienok určených tabuľkami je rozdielne od podmienok určených množinou rozsahov. Kvôli jednotnému spracovaniu je nutné tabuľku transformovať. Na začiatku sa vytvorí prázdna množina rozsahov $RANGES$. Pre každú konštantu $const \in T$ s operátorom rovnosti sa vytvorí zodpovedajúci rozsah určený obojstranným obmedzením. Napríklad pre konštantu 172.16.0.0/16, vznikne rozsah určený obmedzením $172.16.0.0 \leq x \leq 172.16.255.255$. Následne sú z vytvoreného rozsahu odstránené tie hodnoty, ktoré vyhovujú konštantám s operátorom nerovnosti a s väčšou porovnávanou šírku ako $const$.

Pri odstraňovaní hodnôt z rozsahu môže nastať niekoľko situácií. Ak je potrebné odstrániť hodnoty zo spodnej alebo hornej časti, stačí upraviť spodné alebo horné obmedzenie. Pokiaľ sú hodnoty zo stredu, rozsah sa rozdelí na dva a ďalej je nutné pracovať so všetkými takto vzniknutými rozsahmi. Posledná situácia, keď rozsah je podmnožinou odstraňovaných hodnôt, môže nastať v prípade, že daný rozsah vznikol rozdelením. Táto situácia je riešená úplným odstránením tohto rozsahu. Nakoniec sú všetky vzniknuté rozsahy pridané do množiny $RANGES$, ktorá môže byť potom použitá v podmienke nejakého pravidla.

Transformácia tabuľky na zoznam rozsahov

```
1:       Ranges := new empty set
2:
3:       for each Constant in Table
4:       where Operator of Constant is Equal:
5:         RangesTmp := set { createRange(Constant) }
6:
7:         for each ConstantExcl in Table
```

```
8:          where Operator of ConstantNeg is NonEqual
9:          and Width of ConstantExcl >= Width of Constant:
10:
11:           RangeExcl := createRange(ConstantExcl)
12:
13:           for each RangeTmp in RangesTmp:
14:             if min(RangeExcl) <= min(RangeTmp)
15:             and max(RangeExcl) >= max(RangeTmp):
16:               delete RangeTmp from RangesTmp
17:
18:             else if min(RangeExcl) <= max(RangeTmp)
19:             and max(RangeExcl) >= max(RangeTmp):
20:               min(RangeTmp) := max(RangeExcl) + 1
21:
22:             else if min(RangeExcl) <= max(RangeTmp)
23:             and max(RangeExcl) >= max(RangeTmp):
24:               max(RangeTmp) := max(RangeExcl) - 1
25:
26:             else if min(RangeExcl) >= min(RangeTmp)
27:             and max(EangeExcl) <= max(RangeTmp):
28:               RangeTmp2 := copy(RangeTmp)
29:               max(RangeTmp) := min(RangeExcl) - 1
30:               min(RangeTmp2) := max(RangeExcl) + 1
31:               add RangeTmp2 to RangesTmp
32:
33:         append RangesTmp to Ranges
```

3.1.3 Prevod pravidiel do jazyka medzivrstvy

Pravidlá sú v zozname R v takom poradí v akom boli vo vstupnom súbore. Ich priorita je určená spôsobom last-match s quick. Pre spracovanie lupgenom je nutné R transformovať na zoznam typu first-match R'.

Zoznam R je rozdelený na dva zoznamy R'_0 a R'_1. Všetky pravidlá $r \in R$, ktoré majú nastavený príznak q, sú uložené v rovnakom poradí do R'_0. Ostatné sú v obrátenom poradí vložené do R'_1. Výsledný zoznam R' vznikne spojením R'_0 a R'_1.

Transformácia last-match s quick na first-match

```
1:        R'0 := empty list
2:        R'1 := empty list
3:
4:        for each Rule in R:
5:          if Rule is Quick:
6:            add Rule to the end of R'0
7:          else:
8:            add Rule to the beginning of R'1
9:
10:       R' := R'0
11:       append R'1 to R'
```

Pomocou množiny rozsahov je v podmienke vyjadrená logická spojka alebo. Takže napríklad podmienku $c=[attr,\{range_1, range_2, ...\}]$, je možné bez použitia množiny interpretovať ako $attr \in range_1 \vee attr \in range_2 \vee ...$. Lupgen ale neumožňuje v pravidle používať spojku alebo. Riešením tohto problému je rozklad pravidla $r=[a,\{c_1, c_2, ..., c_n\}]$, kde $c_i=[attr_i, RANGES_i]$, tak, že každé vzniknuté pravidlo je určené jednou z možných kombinácií rozsahov. Daná kombinácia obsahuje práve jeden rozsah z každej množiny $RANGES_i$. Celkový počet kombinácií m je súčin veľkostí množín rozsahov určujúcich jednotlivé podmienky.

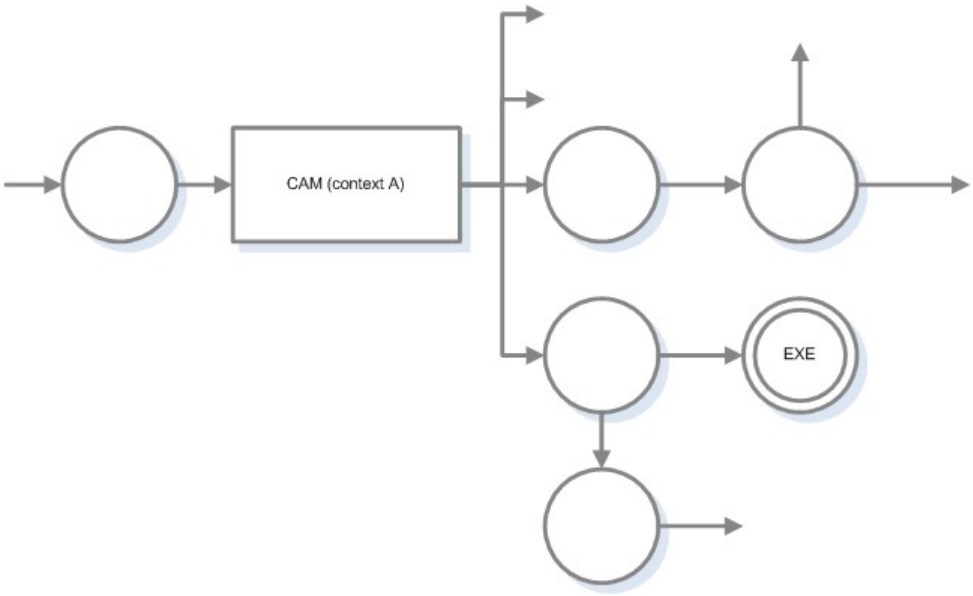
$$m=\prod_{i=1}^{n} |RANGES_i|$$

Napríklad z pravidla $r=[a,\{[attr_1,\{=1,>2\}],[attr_2,\{>5\}],[attr_3,\{=3,\geq 10\}]\}]$ by rozkladom vzniklo $m=2*1*2=4$ pravidiel:

$r'_1=[a,\{[attr_1,\{=1\}],[attr_2,\{>5\}],[attr_3,\{=3\}]\}]$
$r'_2=[a,\{[attr_1,\{>2\}],[attr_2,\{>5\}],[attr_3,\{=3\}]\}]$
$r'_3=[a,\{[attr_1,\{=1\}],[attr_2,\{>5\}],[attr_3,\{\geq 10\}]\}]$
$r'_4=[a,\{[attr_1,\{>2\}],[attr_2,\{>5\}],[attr_3,\{\geq 10\}]\}]$

3.2 Spracovanie jazyka medzivrstvy

Spracovanie medzivrstvy pozostáva z načítania a zjednodušenia pravidiel zo vstupného súbora, vytvorenia obsahu asociatívnej pamäte, zostavenia sekvenčných programov a nakoniec zápisom konfigurácie do výstupného súbora. Týmito problémami sa zaoberajú nasledujúce podkapitoly. Hlavnou myšlienkou je vyhodnocovať testy na zhodu v asociatívnej pamäti a zbytok pravidla v sekvenčnej jednotke. V súčasnosti nie sú využité všetky možnosti vetvenia klasifikácie vo vyhľadávacom procesore. Na začiatku vyhodnotenia je vždy vyvolané jedno vyhľadanie v asociatívnej pamäti a na základe jeho výsledku sa klasifikácia dokončí v príslušnom programe v sekvenčnej jednotke. Jednotlivé programy patriace k rôznym riadkom asociatívnej pamäte sa neprekrývajú.



Obrázok 5: Konfigurácia s použitím Lupgenu

3.2.1 Reprezentácia a spracovanie pravidiel

Zoznam pravidiel je reprezentovaný usporiadanou n-ticou $R=[r_1, r_2, \ldots, r_n]$. Pravidlá sú v zozname usporiadané v poradí, v akom sa nachádzajú vo vstupnom súbore. Priorita pravidiel je typu first-match. Konkrétne pravidlo je možné reprezentovať ako dvojicu $r=[a, C]$, kde C je množina podmienok $C=\{c_1, c_2, c_3, \ldots, c_n\}$, v ktorej musia byť všetky podmienky splnené aby mohla byť vykonaná akcia a. Podmienka je určená premennou, operátorom, hodnotou a v prípade testu na zhodu aj bitovou maskou. Jazyk medzivrstvy umožňuje zápis viac-registrových premenných, ale spracovanie vo vyhľadávacom procesore je možné len po jednom 16-bitovom registri. Je teda potrebné rozložiť podmienky tak, aby boli všetky určené práve jedným registrom.

Podmienka c_a, ktorú je možné zapísať v jazyku medzivrstvy, a ktorá testuje zhodu danej premennej a konštanty pri aplikovaní bitovej masky, je matematicky vyjadrená zápisom $c_a: REG=V/M$. $REG=[reg_1, reg_2, \ldots, reg_n]$ je premenná zložená z niekoľkých registrov unifikovanej hlavičky, kde reg_i je číslo konkrétneho 16-bitového registra. Konštanta $V=[v_1, v_2, \ldots, v_n]$ je zložená z 16-bitových hodnôt v_i. Bitová maska $M=[m_1, m_2, \ldots, m_n]$ je podobne ako konštanta rozdelená na 16-bitové bloky m_i. Princíp zjednodušenia spočíva v rozložení podmienky na množinu elementárnych podmienok, v ktorých je vždy porovnávaný práve jeden register. Počet výsledných podmienok je zhodný s počtom registrov, z ktorých je zložená premenná REG. Ak pre pravidlo $r=[a, C]$ platí, že $c_a \in C$, potom zjednodušením podmienky c_a dostaneme nové pravidlo $r'=[a, C']$, kde:

$$C'=(C \setminus \{c_a\}) \cup \{[reg_1]=[v_1]/[m_1], [reg_2]=[v_2]/[m_2], \ldots, [reg_n]=[v_n]/[m_n]\}.$$

Podmienka porovnania veľkostí hodnôt premennej a konštanty c_b sa zapíše $c_b: REG * V$, kde význam symbolov REG a V je totožný s ich významom v predchádzajúcom texte. Zápis $a * b$ reprezentuje jedno z porovnaní $a<b$, $a \leq b$, $a>b$, $a \geq b$ alebo $a \neq b$. Zjednodušením tejto podmienky dostaneme z pravidla $r=[a, C]$, kde $c_b \in C$, toľko pravidiel, koľko je registrov v premennej REG. Výsledné pravidlá $r'_i=[a, C'_i]$ sú určené nasledujúcimi množinami podmienok:

$$C'_1=(C \setminus \{c_b\}) \cup \{[reg_1] * [v_1]\}$$
$$C'_2=(C \setminus \{c_b\}) \cup \{[reg_1]=[v_1], [reg_2] * [v_2]\}$$
$$\ldots$$
$$C'_n=(C \setminus \{c_b\}) \cup \{[reg_1]=[v_1], [reg_2]=[v_2], \ldots, [reg_n] * [v_n]\}$$

Načítavanie pravidiel začína vytvorením prázdneho zoznamu R. Každé pravidlo je zjednodušené, prípadne rozložené, hneď po tom, ako je celé načítane zo súbora a výsledné pravidlá sú pridané na koniec zoznamu R. Podmienky $reg=v/0$, $reg \geq 0$ a $reg \leq 0$xFFFF sú vždy splnené, nemajú teda žiaden vplyv na vyhodnotenie, a preto sa do pamäťovej reprezentácie pravidiel vôbec neukladajú.

3.2.2 Zostavenie asociatívnej pamäte (CAM)

V tejto časti je popísaný princíp vytvorenia obsahu asociatívnej pamäte (Content-Addressable Memory - CAM). Pre názornosť je postup demonštrovaný na podmienkach, ktoré majú masku nastavenú tak, že sa porovnáva hodnota celého registra. V prípade iných masiek je princíp rovnaký a jediný rozdiel je v tom, že podmienky spracované v asociatívnej pamäti nepracujú s celými registrami, ale s jednotlivými bitmi týchto registrov.

Riadok asociatívnej pamäte nie je dostatočne široký na porovnanie všetkých registrov unifikovanej hlavičky naraz. Preto je nutné určiť, ktoré registre sa v nej budú vyhodnocovať. Ako je uvedené v časti popisujúcej vyhľadávací procesor, je možné zvoliť osem 32-bitových blokov unifikovanej hlavičky. Vybrané sú tie bloky, ktoré sa najčastejšie vyskytujú v podmienkach testujúcich zhodu s konštantou.

Ďalej je potrebné ošetriť situáciu, keď by jeden paket mohol vyhovovať rôznym riadkom pamäte. Takéto riadky budú označené ako konfliktné. Vyhľadanie by totiž vždy vrátilo iba prvý konfliktný riadok a ďalšie by sa stali nedostupné. Problém je vysvetlený na nasledujúcom príklade.

Sú dané dve pravidlá:

$$r_1=(A_1,\{[reg_1]=[1],[reg_2]=[2],[reg_4]>[4]\})$$
$$r_2=(A_2,\{[reg_2]=[2],[reg_3]=[3],[reg_4]<[4]\})$$

Zodpovedajúce riadky asociatívnej pamäte CAM_x a k nim patriace sekvenčné programy by mohli chybne vyzerať takto:

$$(CAM_1=\{reg_1=1,reg_2=2\})\rightarrow if\ reg_4>4\ do\ A_1\ else\ do\ A_0$$
$$(CAM_2=\{reg_2=2,reg_3=3\})\rightarrow if\ reg_4<4\ do\ A_2\ else\ do\ A_0\text{'}$$

Symbol $\rightarrow$ reprezentuje prechod na príslušný program v sekvenčnej jednotke. Pre paket v ktorom by platilo, že $reg_1=1,reg_2=2,reg_3=3,reg_4=1$, by vyhľadanie vrátilo zhodu s CAM_1, ale dokončením vyhodnotenia v sekvenčnej jednotke by sa zistilo, že paket nevyhovuje zodpovedajúcemu pravidlu r_1 a bude preto vykonaná nejaká implicitná akcia A_0. Správne však má byť vykonaná akcia A_2.

Dva riadky CAM_a a CAM_b sú navzájom konfliktné ak neexistuje žiadny register, ktorý by bol testovaný v týchto dvoch riadkoch na rozdielnu hodnotu. Teda riadky sú konfliktné, ak pre všetky podmienky, ktoré porovnávajú rovnaký register na obidvoch riadkoch, sú porovnávané konštanty zhodné: $\forall x,(reg_x=a_x)\in CAM_a\wedge(reg_x=b_x)\in CAM_b : a_x=b_x$

Jednou možnosťou je vyriešenie konfliktu čisto programom v statickej pamäti. Potom by správne spracovanie vyzeralo nasledovne:

$$(CAM_1=\{reg_1=1,reg_2=2\})\rightarrow if\ reg_4>4\ do\ A_1\ else\ if\ reg_3=3\wedge reg_4<4\ do\ A_2\ else\ do\ A_0$$
$$(CAM_2=\{reg_2=2,reg_3=3\})\rightarrow if\ reg_4<4\ do\ A_2\ else\ do\ A_0$$

Lupgen však využíva inú metódu. Efektívnejšie je totiž konflikt vyriešiť predradením ďalšieho riadka $CAM_{ab}=CAM_a\cup CAM_b$ pred existujúce a odstrániť prípadné duplicitné riadky. Predchádzajúci príklad s pravidlami r_1 a r_2 je možné previesť na:

$$(CAM_{12}=\{reg_1=1,reg_2=2,reg_3=3\})\rightarrow if\ reg_4>4\ do\ A_1\ else\ if\ reg_4<4\ do\ A_2\ else\ do\ A_0$$
$$(CAM_1=\{reg_1=1,reg_2=2\})\rightarrow if\ reg_4>4\ do\ A_1\ else\ do\ A_0$$
$$(CAM_2=\{reg_2=2,reg_3=3\})\rightarrow if\ reg_4<4\ do\ A_2\ else\ do\ A_0$$

Najskôr je z každého pravidla vygenerovaný jeden riadok, ktorý obsahuje všetky tie podmienky na zhodu patriace k danému pravidlu, ktoré porovnávajú registre vybrané na spracovanie v asociatívnej pamäti. Napríklad ak sú v asociatívnej pamäti spracovávané registre reg_1 až reg_4, potom pre pravidlo r_1 bude vygenerovaný riadok CAM_1. Toto bude v ďalšom texte označované zápisom $CAM_1=GenerateCamRow(r_1)$.

Potom sa riešia konflikty medzi vygenerovanými riadkami. Kvôli efektivite spracovania sú konflikty rozdelené do niekoľkých skupín. Taktiež sa predchádza vloženiu duplicitných riadkov, aby ich vzápätí nebolo potrebné odstraňovať. Čísla riadkov budú a a b, kde CAM_a leží pred CAM_b. V prípade, že $CAM_b \subset CAM_a$, netreba robiť nič, pretože predradením $CAM_{ab} = CAM_a \cup CAM_b = CAM_a$, by vznikla duplicita. Ak $CAM_a \subset CAM_b$, riadky sa navzájom vymenia. V ostatných prípadoch bude pridaný riadok $CAM_{ab} = CAM_b \cup CAM_b$ na prvé miesto, pokiaľ takýto riadok ešte neexistuje. Algoritmus opakovane prechádza všetky dvojice riadkov a rieši konflikty medzi nimi. Ukončí sa v prípade, že pri celom prechode už nedošlo k žiadnej zmene zoznamu riadkov. Touto metódou vznikne zoznam riadkov zoradených zostupne podľa špecifickosti. Viac špecifické riadky, ktorým vyhovuje menšia množina paketov, sú pred menej špecifickými, ktorým vyhovuje viac paketov. Algoritmus riešenia konfliktov využívajúci na uloženie zoznamu riadkov CAM jednosmerne spájaný zoznam je možné v pseudojazyku zapísať:

Riešenie konfliktov v CAM

```
1:       Solved := False
2:       while not Solved:
3:         Solved := True
4:
5:         CamRowA := first Of CamRows
6:         while CamRowA:
7:           CamRowB := next CamRowA
8:
9:           while CamRowB:
10:            if CamRowA in conflict with CamRowB:
11:              if CamRowB is subset of CamRowA:
12:                do nothing
13:
14:              else if CamRowA is subset of CamRowB:
15:                swap CamRowA with CamRowB
16:                Solved := False
17:
18:              else:
19:                NewCamRow := CamRowA union CamRowB
20:
21:                if NewCamRow not in CamRows:
22:                  add NewCamRow to the beginning of CamRows
23:                  Solved := False
24:
25:            CamRowB := next CamRowB
26:          #end of "while CamRowB"
27:
28:          CamRowA := next CamRowA
29:        #end of "while CamrowA"
```

3.2.3 Zostavenie sekvenčných programov

Po zostavení asociatívnej pamäte sa vytvárajú sekvenčné programy zodpovedajúce jednotlivým riadkom. Pre každé pravidlo sa vytvorí blok inštrukcií. Ten je zostavený na základe podmienok, ktoré nie sú vyhodnocované v asociatívnej pamäti. Pokiaľ sú splnené všetky podmienky vyhodnocované

v bloku alebo v bloku nie sú vyhodnocované žiadne podmienky, klasifikácia sa ukončí s nastavením výstupného záznamu vyhľadávacieho procesora na hodnotu určenú daným pravidlom. V opačnom prípade sa vykoná skok za poslednú inštrukciu bloku. Z takýchto blokov uložených za seba sa potom vytvoria kompletné programy zodpovedajúce jednotlivým riadkom asociatívnej pamäte. Dôležitou úlohou je určiť, ktorým pravidlám môže vyhovovať paket, pokiaľ by bol výsledkom vyhľadania daný riadok CAM. Pre riadok CAM_x sú na zostavenie sekvenčného programu vybrané tie pravidlá r_y, pre ktoré platí, že $GenerateCamRow(r_y) \subseteq CAM_x$. Bloky inštrukcií vytvorené na základe týchto pravidiel budú vo výslednom programe pre riadok CAM_x v takom poradí, v akom sú pravidlá vo vstupnom súbore. Tým sa zabezpečí, že pravidlá budú vyhodnocované v poradí typu first-match.

Transformácia last-match s quick na first-match

```
1:       for each Rule in Rules:
2:         RuleProgramBlock=generateRuleProgramBlock(Rule)
3:         RuleDefaultCam=generateCamRow(rule)
4:
5:         for each CamRow in CamRows:
6:           if RuleDefaultCam is subset of CamRow:
7:             append RuleProgramBlock to Program for CamRow
```

Bloky vygenerované pre jednotlivé pravidlá pozostávajú z menších skupín inštrukcií, ktoré vyhodnocujú jednotlivé podmienky alebo ukončujú klasifikáciu. Podmienky typu $reg > const$, $reg \geq const$, $reg < const$ a $reg \leq const$ majú zastúpenie v inštrukčnej sade a sú reprezentované skupinou dvoch inštrukcií:

```
Jxxx step=2, reg, const
JMP behind_rule_block
```

Pre podmienku $reg \neq const$ je využitá inštrukcia testujúca zhodu s plne nastavenou bitovou maskou. Celú skupinu tvoria tri inštrukcie:

```
CMP step=2, reg, const, mask=0xFFFF
JMP current+2
JMP behind_rule_block
```

Najzložitejšie je vytvorenie skupiny inštrukcií reprezentujúcej podmienku typu $reg = const / mask$, pretože inštrukcia CMP umožňuje zadávať iba masku so zhodnými dvojicami bitov a je nutné toto obmedzenie obísť, aby sa dala použiť ľubovoľná maska. Zápis $a[i]$ bude v nasledujúcom texte znamenať i-ty bit hodnoty a, kde bity sú číslované od 0 do 15. Ak sa má testovať podmienka $c: reg = const / mask$, najskôr sa testuje, či je splnená menej špecifická podmienka $c'_0: reg = const / mask'_0$, kde pre i deliteľné dvomi platí $mask'_0[i] = mask'_0[i+1]$ a $mask[i] = 1 \wedge mask[i+1] = 1 \Leftrightarrow mask'_0[i] = 1$.

Následne sa pre všetky i deliteľné dvomi, kde $mask[i] \neq mask[i+1]$, testuje, či je splnená aspoň jedna z podmienok $c'_{i0}: reg = const'_{i0} / mask'_i$ alebo $c'_{i1}: reg = const'_{i1} / mask'_i$. Bity $const'_{i0}[j] = 0$ a $const'_{i1}[j] = 1$, kde j je index nulového bitu masky $mask$. Teda platí $j \in \{i, i+1\}$ a $mask[j] = 0$. Ostatné bity konštánt $const'_{i0}$ a $const'_{i1}$ sú zhodné s bitmi $const$. Maska $mask'_i$ má nastavené iba bity $mask'_i[i] = mask'_i[i+1] = 1$ a ostatné bity sú nulové.

Podmienka c'_0 je reprezentovaná pomocou skupiny dvoch inštrukcií:

```
CMP step=2, reg, const, mask'0
JMP behind_rule_block
```

Za ňou nasledujú skupiny štyroch inštrukcií tvoriacich podmienky $c'_{i0} \vee c'_{i1}$:

```
CMP step=4, reg, const'i0, mask'i
CMP step=2, reg, const'i1, mask'i
JMP behind_rule_block
JMP current+1
```

Napríklad, ak sú pre vyhodnotenie v asociatívnej pamäti vybrané registre reg_1 až reg_3, potom pre riadok $CAM_1 = \{reg_1 = 1, reg2 = 2\}$ a tri pravidlá $r_1 : (a_1, \{reg_1 = 1, reg_3 > 5, reg_4 > 4\})$, $r_2 : (a_1, \{reg_2 = 2, reg_3 != 3\})$ a $r_3 : (a_3, \emptyset)$, bude výsledný program:

```
CAM_1:
r1:
  JGTH step=2, reg3, const=5
  JMP behind_r1

  JTLH step=2, reg4, const=4
  JMP behind_r1

  EXE rec=a1

behind_r1:
r2:
  CMP step=2, reg3, const=3, mask=0xFFFF
  JMP behind_r2
  JMP current+1

  EXE rec=a2

behind_r2:
r3:
  EXE rec=a3
```

Adresa statickej pamäte, na ktorej začína program je určená poradovým číslom daného riadka asociatívnej pamäte vynásobeným konštantou. Mohlo by sa stať, že počet inštrukcií programu je väčší ako hodnota tejto konštanty. V tom prípade sa daný riadok CAM, N-krát zopakuje. Pretože vyhľadanie vráti vždy iba prvý zo zhodných riadkov, zopakované riadky nebudú nikdy vyhľadané. Výsledný priestor pre program sa týmto spôsobom N-krát zväčší.

4 Dosiahnuté výsledky

Rýchlosť klasifikácie pomocou vyhľadávacieho procesora záleží z veľkej časti na jeho konfigurácii. Tá pozostáva zo sekvenčného programu a obsahu asociatívnej pamäte. V tejto kapitole sú skúmané vlastnosti konfigurácie, ktorá vznikne za pomoci vytvoreného softvéra. Nakoniec je porovnaný výkon softvérového riešenia paketového filtra s možnosťami harvérom urýchleného filtrovania.

Sledovanými vlastnosťami generovanej konfigurácie je počet obsadených 272-bitových riadkov asociatívnej pamäte a čas najdlhšieho vyhodnotenia paketu, ktorý je možné vyjadriť počtom vykonaných inštrukcií. Pre každý paket sú vykonané minimálne dve inštrukcie. Prvá vyvolá vyhľadanie v asociatívnej pamäti, druhá ukončí klasifikáciu a určí ďalšie spracovanie paketu. Ďalšou skúmanou vlastnosťou je vážený priemer najdlhších vyhodnotení pre jednotlivé riadky asociatívnej pamäte. Váha je určená na základe počtu hodnôt, ktoré môžu zodpovedať danému riadku. Pokiaľ maska obsahuje n nulových, potom výsledná váha je $w = 2^n$.

Priepustnosť vyhľadávacieho procesora tiež záleží na dĺžke analyzovaných paketov. Najmenšia dĺžka ethernetového paketu, ktorý neobsahuje žiadne dáta, je 64B. Pri spracovaní takýchto paketov sa celková priepustnosť procesora pracujúceho na frekvencii 100MHz pohybuje na hranici 12Gb/s, čo je približne 3Gb/s pre každé zo štyroch rozhraní. Pri tejto rýchlosti je možné vykonať jedno vyhľadanie v asociatívnej pamäti a spracovať päť inštrukcií. Pre nižšiu rýchlosť 4*2Gb/s je možné vykonať 2 vyhľadania a 8 inštrukcií, pre 4*1Gb/s až 4 vyhľadania a 16 inštrukcií.

4.1 Závislosť konfigurácie na vstupnej množine pravidiel

Nasledujúca tabuľka číslo 6 obsahuje vlastnosti konfigurácie v závislosti na počte a type porovnávaných atribútov. Sú vytvorené štyri typy množín obsahujúce vždy 250 pravidiel. Prvý typ množiny (T1) obsahuje pravidlá testujúce iba zhodu zdrojovej adresy s prefixom siete. Druhý typ (T2) pridáva k existujúcim pravidlám test na príslušnosť zdrojového portu do rozsahu hodnôt. Tretí typ (T3) testuje zdrojové aj cieľové adresy a zdrojový port. Posledný typ (T4) testuje obidve adresy a obidva porty. Je vykonaných 50 experimentov a následne sú vypočítané priemerné hodnoty sledovaných vlastností. Dĺžka vyhodnotenia je udávaná v počte vykonaných inštrukcií. Na základe tohto počtu je určená celková priepustnosť procesora pre pakety s dĺžkou 64B.

typ pravidiel		T1	T2	T3	T4
počet riadkov CAM		227,62	228,72	448,12	448,12
najdlhšie vyhodnotenie	počet inštrukcií	2,00	63,92	15,38	24,30
	priepustnosť [Gb/s]	12,8	1,0	4,2	2,6
vážený priemer dĺžok vyhodnotení	počet inštrukcií	2,00	37,93	5,92	8,54
	priepustnosť [Gb/s]	12,8	1,7	10,8	7,5

Tabuľka 6: Závislosť konfigurácie na počte a type porovnávaných atribútov

Pre prvý typ množiny sú všetky podmienky vyhodnocované ako testy na zhodu s použitím bitovej masky. Keďže je možné všetky porovnávané atribúty vyhľadať na jeden raz v asociatívnej pamäti, tak na každé vyhodnotenie paketu sú potrebné práve dve inštrukcie. Výsledná konfigurácia

obsahuje menej riadkov ako je počet pravidiel. To je spôsobené tým, že z rovnakých riadkov, ktoré vznikli na základe rôznych pravidiel sú všetky okrem prvého odstránené.

Príčinou mierneho nárastu počtu riadkov asociatívnej pamäte pre druhý typ množiny sú sekvenčné programy dlhšie ako je rozdiel adries statickej pamäte, na ktorých začína sekvenčné vyhodnotenie dvoch po sebe idúcich riadkov asociatívnej pamäte.

Pridaním testu na cieľovú adresu do pravidiel v treťom type množín sa výrazne zvýšilo využitie asociatívnej pamäte a zároveň výrazne klesol počet inštrukcií sekvenčných programov. Dôvodom je možnosť riešenia konfliktov medzi niektorými dvojicami riadkov vytvorením nového riadka. Toto v predchádzajúcich prípadoch nebolo možné, lebo z konfliktných riadkov bol vždy jeden nadmnožinou druhého.

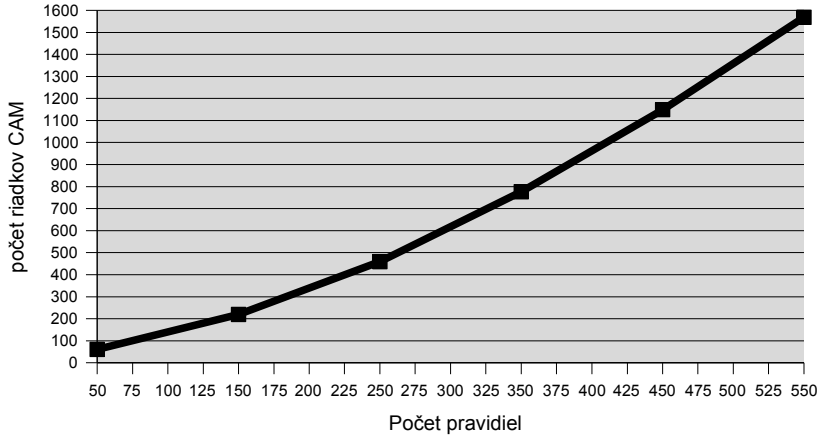
V poslednom type množiny sa využitie asociatívnej pamäte nezvýšilo, ale kvôli pridaniu ďalšej podmienky testujúcej príslušnost portu do intervalu, ktorá musí byť vyhodnotená v sekvenčnej jednotke, sa zvýšil počet inštrukcií sekvenčných programov.

Pre zistenie závislostí na veľkosti množiny sú vygenerované náhodné množiny, ktoré obsahujú 50 až 550 pravidiel. Kvôli lepšej presnosti výsledku je pre každý počet vygenerovaných 50 množín. Jednotlivé pravidlá testujú, či zdrojová a cieľová IPv4 adresa vyhovujú danému prefixu siete, a ďalej, či zdrojový a cieľový port patria do určitého intervalu hodnôt. Tabuľka 7 zobrazuje priemerné hodnoty skúmaných vlastností v závislosti na počte pravidiel vo vstupnej množine.

počet pravidiel		50	150	250	350	450	550
počet riadkov CAM		60,80	219,68	459,36	776,42	1148,64	1568,70
najdlhšie vyhodnotenie	počet inštrukcií	13,70	20,78	25,00	30,00	34,40	38,00
	priepustnosť [Gb/s]	4,7	3,0	2,6	2,1	1,9	1,7
vážený priemer dĺžok vyhodnotení	počet inštrukcií	7,13	7,76	8,51	10,28	10,67	11,94
	priepustnosť [Gb/s]	9,0	8,3	7,5	6,2	6,0	5,4

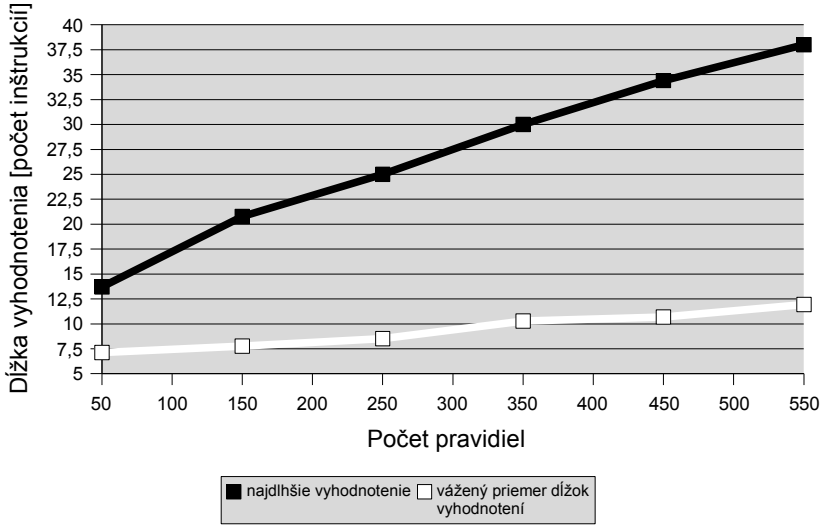
Tabuľka 7: Závislosť konfigurácie od počtu pravidiel

Na nasledujúcom obrázku vidno, že závislosť počtu riadkov CAM je o niečo horšia ako lineárna. Dôvodom je spôsob riešenia konfliktov, pretože ku každej dvojici konfliktných riadkov je s určitou pravdepodobnosťou vytvorený nový riadok.



Obrázok 6: Závislosť počtu riadkov CAM na počte pravidiel

Počet inštrukcií potrebných na vyhodnotenie paketu rastie v závislosti na počte pravidiel.

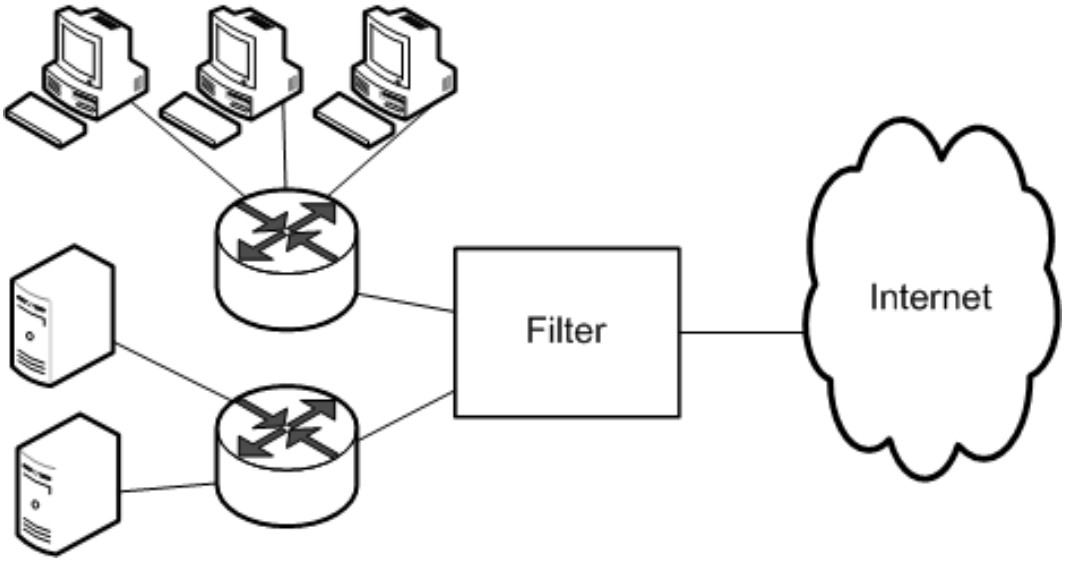


Obrázok 7: Závislosť dĺžky vyhodnotenia na počte pravidiel

4.2 Typická množina pravidiel

Na otestovanie vlastností konfigurácie vyhľadávacieho procesora v reálnom prípade je vytvorená množina filtrovacích pravidiel pre fiktívnu organizáciu, pre ktorú platí:

- Organizácia je pripojená na internet a jej sieť obsahuje servery a pracovné stanice. Je použitý paketový filter s minimálne tromi rozhraniami, kde na prvé je pripojený internet, na druhé servery a na treťom všetky pracovné stanice. Zapojenie je znázornené na obrázku 8.
- Kvôli bezpečnosti sú servery v inej podsieti ako pracovné stanice.
- Niektoré služby serverov sú dostupné iba z pracovných staníc, iné sú dostupné aj z internetu.
- Všetky pracovné stanice okrem zablokovaných majú prístup na internet.

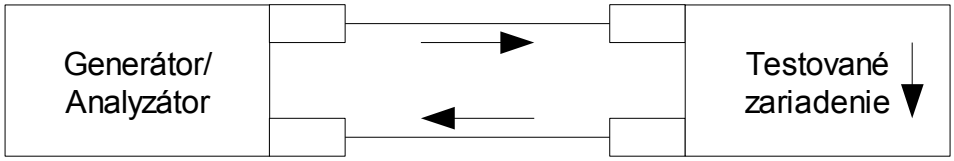


Obrázok 8: Zapojenie filtra

Konkrétny príklad takejto množiny pravidiel zapísanej vo vstupnom jazyku filtra sa nachádza v prílohe F. Zápis obsahuje 3 tabuľky a 14 pravidiel. Po spracovaní programom lupf, vznikne súbor v jazyku medzivrstvy obsahujúci 27 pravidiel. Výsledná konfigurácia vytvorená programom lupgen zaberá 137 riadkov asociatívnej pamäte a na klasifikáciu ľubovoľného paketu potrebuje jedno vyhľadanie v asociatívnej pamäti a dve inštrukcie. Z toho plynie, že celková priepustnosť procesora pre túto množinu pravidiel je 12Gb/s.

4.3 Porovnanie HW a SW filtrovania

Pre otestovanie možností filtrovania v softvéri, bola zvolená najjednoduchšia možná konfigurácia, kedy všetky pakety prichádzajúce na jedno rozhranie boli kopírované na druhé rozhranie. Zapojenie je naznačené na obrázku číslo 9. Testované zariadenie obsahovalo dva procesory Intel(r) Xeon(tm) 3.00GHz, 1GB RAM. Bol použitý filter iptables na operačnom systéme Linux 2.6.19.1.



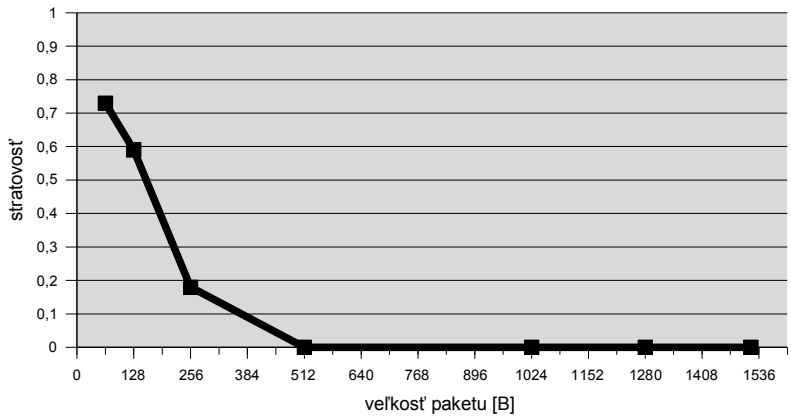
Obrázok 9: Zapojenie testovaného zariadenia

Testy ukázali, že aj pre takúto jednoduchú množinu pravidiel softvérový filter pre krátke pakety nie je schopný spracovať sieťový tok s rýchlosťou 1Gb/s. Naproti tomu priepustnosť vyhľadávacieho procesora umožňuje aj pre najkratšie možné pakety pracovať so sieťovými tokmi pri rýchlosti 12Gb/s. Tabuľka číslo 8 a graf na obrázku 10 znázorňujú stratovosť softvérového filtra pre rýchlosť 1Gb/s v závislosti na dĺžke paketu. Stratovosť je definovaná pomerom zahodených paketov k celkovému počtu paketov:

$$stratovosť = \frac{počet\ generovaných\ paketov - počet\ prijatých\ paketov}{počet\ generovaných\ paketov}$$

veľkosť paketu [B]	64	128	256	512	1024	1280	1518
stratovosť	0,73	0,59	0,18	0,00	0,00	0,00	0,00

Tabuľka 8: Stratovosť pre 1Gb/s



Obrázok 10: Stratovosť pre 1Gb/s

5 Záver

Počas práce na téme tejto bakalárskej práce bol navrhnutý a implementovaný softvér určený na konfiguráciu vyhľadávacieho procesora implementovaného na platforme COMBO6(X), ktorý slúži na klasifikáciu paketov.

Pre získanie informácií o architektúre vyhľadávacieho procesora a o štruktúre unifikovanej hlavičky boli naštudované dokumenty vytvorené v rámci projektu Liberouter popisujúce danú problematiku. Taktiež bolo nutné sa zoznámiť s existujúcimi voľne dostupnými paketovými filtrami a metódami klasifikácie. Načerpané znalosti boli použité pri písaní teoretického rozboru.

Na základe informácií získaných pri študovaní problematiky bol navrhnutý jazyk medzivrstvy ktorý umožňuje využiť pre klasifikáciu všetky údaje uvedené v unifikovanej hlavičke.

Vytvorený softvér je rozložený do niekoľkých vrstiev, čo zvyšuje znovupoužiteľnosť jednotlivých častí. Transformácia vstupnej množiny pravidiel zapísanej v jazyku podobnom OpenBSD PF na konfiguráciu hardvéru prebieha v niekoľkých oddelených krokoch.

Funkčnosť navrhnutého riešenia bola overená na jednoduchých množinách pravidiel. Taktiež bola skúmaná pamäťová náročnosť výslednej konfigurácie a celková teoretická priepustnosť procesora v závislosti na počte a type pravidiel. Zistilo sa, že pomocou tohto softvéra sa dajú vytvoriť také konfigurácie vyhľadávacieho procesora, ktoré umožňujú spracovávať sieťové toky pri rýchlsti 12Gb/s. Testy ďalej ukázali, že osobný počítač nie je dostatočne výkonný na filtrovanie sieťových tokov pre gigabitové rýchlosti a je nutné použiť harvérovú akceleráciu.

Vytvorený softvér bude v projekte Liberouter používaný na konfiguráciu hardvérom urýchleného filtra NIFIC. Softvér je možné ďalej vylepšiť optimalizovaním generovaných sekvenčných programov. Ďalším zdokonalením by mohlo byť využitie viacnásobného vyhľadania v asociatívnej pamäti, čo by malo vplyv na zlepšenie výkonu pre pravidlá pracujúce s IPv6 adresami.

6 Použitá literatúra

[1] Kořenek, J.: *Diplomová práce*. Brno, FIT VUT v Brně, 2003.

[2] Málek, T.: *Návrh a implementace procesoru pro klasifikaci IPv4/IPv6 paketu, bakalářská práce*. Brno, FIT VUT v Brně, 2006.

[3] Pospíšil, M.: *SCAMPIDUMP - monitorování provozu sítě v hardwaru, diplomová práce*. Brno, Masarykova univerzita, 2005.

[4] Antoš, D.: *Hardware-constrained Packet Classification, thesis*. Brno, Masaryk University, 2006.

[5] Andreasson, O.: *Iptables Tutorial*. Dokument dostupný na URL http://iptables-tutorial.frozentux.net/iptables-tutorial.html

[6] *PF: The OpenBSD Packet Filter*. Dokument dostupný na URL http://www.openbsd.org/faq/pf/

[7] Hansteen, P. N. M.: *Firewalling with OpenBSD's PF packet filter*. Dokument dostupný na URL http://www.bgnett.no/~peter/pf/en/

[8] *Ipfirewall*. Dokument dostupný na URL http://en.wikipedia.org/wiki/Ipfw

[9] *Packet Filter*. Dokument dostupný na URL http://en.wikipedia.org/wiki/Packet_filter

[10] *IPv6*. Dokument dostupný na URL http://en.wikipedia.org/wiki/Ipv6

[11] Málek, T., Kořenek, J., Tobola, J.: *Look-up processor*. Dokument dostupný na URL http://www.liberouter.org/vhdl_design/generated/HEAD_lup_info.php

[12] Hažmuk, I.: *Specification of Unified Header for IPv6 accelerated ethernet card Combo6*. Liberouter, 2007.

7 Zoznam netlačených príloh

Príloha 1. CD obsahujúce zdrojové texty, programovú dokumentáciu a technickú správu

Príloha A Unifikovaná hlavička

Štruktúra unifikovanej hlavičky záleží na type protokolu konkrétneho paketu. Sú rozlíšené tri možnosti: IPv4, IPv6 a ARP.

	IPv6				**IPv4**				**ARP**			
register	**15-12**	**11-8**	**7-4**	**3-0**	**15-12**	**11-8**	**7-4**	**3-0**	**15-12**	**11-8**	**7-4**	**3-0**
0	L2_REG				L2_REG				L2_REG			
1	L3_REG				L3_REG				L3_REG			
3-5	DST_MAC				DST_MAC				DST_MAC			
6-8	SRC_MAC				SRC_MAC				SRC_MAC			
9	802.1p	802.1q VLAN TAG			802.1p	802.1q VLAN TAG			802.1p	802.1q VLAN TAG		
10				IF_ID				IF_ID				IF_ID
11-12	SRC_IPv6 addr				SRC_IPv4 addr							
13-18												
19-20	DST_IPv6 addr				DST_IPv4 addr							
21-26												
27	SRC_PORT				SRC_PORT							
28	DST_PORT				DST_PORT							
29-36	INTER_ADDR											
37	PLEN				PLEN				PLEN			
38	TCP FLAGS	PROTOCOL			TCP FLAGS	PROTOCOL						
...												
63	PACKET_ID				PACKET_ID				PACKET_ID			

Tabuľka 9: Štruktúra unifikovanej hlavičky

Príloha B Kódovanie inštrukcií vyhľadávacieho procesora

Program sekvenčnej jednotky je uložený v statickej pamäti so šírkou slova 36 bitov. Tomu je tiež prispôsobená šírka inštrukcie, teda všetky inštrukcie majú rovnakú veľkosť a zaberajú práve jedno slovo v tejto pamäti. Ich binárne kódovanie je uvedené v nasledujúcej tabuľke.

Symbolický zápis	Operačný kód
CAM Set	00xxxxxxxxxxxxxxxxxxxxxxxxxxSSSSSSSS
JMP Addr	01xxxxxxxx01xxxxAAAAAAAAAAAAAAAAAAAA
CMP Step, Reg, Mask, Const	10SSRRRRRR1xMMMMMMMMCCCCCCCCCCCCCCCC
JLTH Step, Reg, Const	10SSRRRRRR0xxxxx0101CCCCCCCCCCCCCCCC
JLTE Step, Reg, Const	10SSRRRRRR0xxxxx1101CCCCCCCCCCCCCCCC
JGTH Step, Reg, Const	10SSRRRRRR0xxxxx0011CCCCCCCCCCCCCCCC
JGTE Step, Reg, Const	10SSRRRRRR0xxxxx1011CCCCCCCCCCCCCCCC
JBLT Step, Reg, Const	10SSRRRRRR0xxxxx0100CCCCCCCCCCCCCCCC
JBLE Step, Reg, Const	10SSRRRRRR0xxxxx1100CCCCCCCCCCCCCCCC
JBGT Step, Reg, Const	10SSRRRRRR0xxxxx0010CCCCCCCCCCCCCCCC
JBGE Step, Reg, Const	10SSRRRRRR0xxxxx1010CCCCCCCCCCCCCCCC
EXE Rec	11xxPPPPPPPPPPPPPPPPPPPPPPPPPPPPPP

Tabuľka 10: Operačné kódy inštrukcií

Maska je v inštrukčnom slove zakódovaná na 8 bitov tak, že jeden bit kódu určuje dva susedné bity masky. Z tohto dôvodu nie je možné priamo porovnávať jednotlivé bity. Parameter step pri podmienených skokoch je kódovaný podľa nasledujúcej tabuľky. Preto sa nedá použiť ľubovoľná hodnota tohto parametra.

Binárny kód parametra step	Veľkosť relatívneho skoku
00	2
01	8
10	32
11	128

Tabuľka 11: Kódovanie parametra step

Príloha C Značenie syntaxe

Pre popis syntaxe navrhnutých jazykov bola zvolená forma popisu podobná BNF. V tejto kapitole je vysvetlený význam jednotlivých znakov nasledovaný príkladom použitia.

Terminálne symboly sú ohraničené apostrofami. Môžu obsahovať špeciálne znaky podobné ako reťazce jazyka C, akými sú napríklad tabelátor „\t“ alebo nový riadok „\n“.

```
'123'
'\nAB\tCD'
```

Neterminálne symboly sú vyjadrené alfanumerickými reťazcami začínajúcimi písmenom, ktoré môžu naviac obsahovať znak „_“.

```
ipv4_constant
```

Existuje niekoľko možností na vyjadrenie násobnosti výskytu daného symbolu. Znak „*“ za symbolom znamená nula až nekonečno opakovaní, znak „+“ za symbolom jedno až nekonečno opakovaní.

```
'123'+
ipv4_constant*
```

Sekvencia určitého počtu rovnakých symbolov sa značí číslom, ktoré určuje počet, nasledovaným znakom „*“ a daným symbolom.

```
5*'A'
```

Pre označenie nepovinnej skupiny symbolov, teda nula alebo jeden výskyt, sa používajú hranaté zátvorky „[ ]“.

```
[ ipv4_constant ]
```

Ľubovoľná sekvencia symbolov môže byť uzavretá do jednoduchých zátvoriek „( )“, čím sa umožní s ňou pracovať ako s jedným symbolom.

```
( constant ',' )* constant
```

Na vyjadrenie výberu z rôznych variant slúži operátor „|“. Použitie tohto znaku znamená, že vo výsledku sa môže vyskytovať práve jeden z takto oddelených symbolov.

```
'a' | 'b' | 'c' | 'd'
```

Pre skrátenie predchádzajúceho zápisu je možné použiť operátor „-“, pomocou ktorého sa dá vyjadriť výber zo sekvencie symbolov.

```
'a' - 'd'
```

Pravidlá sú vytvárané pomocou operátora „:=“, kde na ľavej strane sa musí vždy nachádzať práve jeden neterminálny symbol a na pravej výraz zložený z terminálnych aj neterminálnych symbolov, operátorov a zátvoriek. Nasledujúci príklad definuje symbol alfanum, ktorý je zložený z alfanumerických znakov.

```
alfanum := ('a' – 'z' | '0' - '9') [ alfanum ]
```

Príloha D Vstupný jazyk filtra

Náplňou tejto kapitoly je syntaktický a sémantický popis vstupného jazyka pre popis filtra. Nad syntaktickým popisom jednotlivých konštrukcií je vysvetlený ich význam. Jazyk je podobný jazyku OpenBSD packet filtra a nerozlišuje veľkosť písmen. Zapísané syntaktické pravidlá sú nadmnožinou sémanticky správnych konštrukcií.

D.1 Konštanty

Desiatkové číslice sú znaky '0' až '9'.

```
dec_digit := '0' – '9'
```

Šestnástkové číslice sú znaky '0' – '9' a písmená 'a' až 'f'.

```
hexa_digit := dec_digit | ('a' – 'f')
```

Desiatkové konštanty sú zapísané ako sekvencia číselných znakov.

```
dec_const := dec_digit+
```

Konštanty v šestnástkovej sústave majú úvodný znak '$' nasledovaný sekvenciou hexadecimálnych číslic.

```
hexa_const := '$' hexa_digit+
```

Môžu byť zadávané IP adresy verzie 4 alebo 6.

```
ip_const := ipv4_const | ipv6_const
```

IP adresa verzie 4 je sekvencia štyroch čísle v rozsahu od 0 do 255 oddelených bodkou. Voliteľne môže byť adresa nasledovaná lomítkom a desiatkovým číslom určujúcim počet bitov zľava, ktoré sa porovnávajú.

```
ipv4_const := dec_const 3*('.' dec_const) ['/' dec_const]
```

Príklady správnych IP adries verzie 4:

```
10.0.0.0 / 8
10.1.2.3
```

Adresa Ipv6 je obvykle zapisovaná ako osem skupín hexadecimálnych číslic oddelených dvojbodkou. Ľubovoľný počet po sebe idúcich skupín samých núl môže byť nahradený dvomi dvojbodkami. Rovnako ako adresa Ipv4 môže byť nasledovaná lomítkom a desiatkovým číslom určujúcim počet bitov zľava, ktoré sa porovnávajú.

```
ipv6_const := [hexa_digit+ (':' hexa_digit)* ] ['::'] [hexa_digit+
  (':' hexa_digit)* ] ['/' dec_const]
```

Príklady navzájom ekvivalentných adries IP verzie 6:

```
500f:0a23:0000:0000:0000:0000:fdc4:0056
500f:a23:0:0:0:0:fdc4:56
500f:a23::fdc4:56
```

D.2 Zoznamy

Zoznam môže obsahovať iba IP adresy alebo desiatkové 16-bitové konštanty.

```
list_constant := dec_const | ipv4_const | ipv6_const
```

Zoznam je neprázdna množina konštánt a negovaných konštánt, prípadne vnorených zoznamov. Zapisuje sa pomocou zložených zátvoriek. Jednotlivé položky môžu byť oddelené čiarkami alebo bielymi znakmi. V zozname sa môže naraz vyskytovať iba jeden druh konštánt.

```
list := '{' ( ( (['!'] list_constant) | list) [','])+ '}'
```

V prípade použitia zoznamu, je z jedného pravidla vytvorených toľko podpravidiel, koľko má zoznam prvkov. Teda z pravidla:

```
pass in on 0 from { 10.0.0.0/8, !10.1.2.3 }
```

sú vytvorené dve popravidlá, ktoré prepustia paket z ľubovoľnej adresy:

```
pass in on 0 from 10.0.0.0/8
pass in on 0 from !10.1.2.3
```

Preto je vhodné použiť radšej tabuľky a v zoznamoch vôbec nepoužívať operátor '!'.

D.3 Tabuľky

Tabuľky je narozdiel od zoznamov možné definovať a následne použiť pomocou identifikátora. Meno tabuľky je alfanumerický reťazec začínajúci písmenom uzavretý v lomených zátvorkách.

```
table_name := '<' 'a'-'z' ('a'-'z' | '0'-'9' | '_')* '>'
```

Definícia pravidla začína kľúčovým slovom table nasledovaným menom tabuľky a zoznamom konštánt.

```
table_definition := 'table' table_name list
```

Tabuľky majú taktiež inú sémantiku ako zoznamy. Atribút paketu je vyhodnotený vzhľadom na najviac špecifikovanú zhodnú konštantu. Teda napríklad tabuľke:

```
table <my_table1> { 172.16.0.0/16 !172.16.5.0/24 172.16.5.100 }
```

vyhovujú IP adresy 172.16.1.254 alebo 172.16.5.100. Adresa 192.168.15.1 nevyhovuje, pretože neexistuje žiadna hodnota v tabuľke, s ktorou by sa zhodovala. Adresa 172.16.5.99 taktiež nevyhovuje, lebo najviac špecifická zhodná konštanta je !172.16.5.0/24.

D.4 Pravidlá

Pravidlá špecifikujú kritéria, ktoré musí paket splniť a výslednú akciu, ktorá sa vykoná, ak sú tieto kritéria splnené. Teda napríklad, či je paket prepustený alebo blokovaný. Syntax pravidla je definovaná nasledujúcim predpisom. Význam jednotlivých parametrov je popísaný ďalej.

```
rule := action ['quick'] ['on' interface] ['proto' protocol] ['from'
  src_addr ['port' src_port]] ['to' dst_addr ['port' dst_port]]
  ['flags' [set_flags] '/' checked_flags]
```

Akcia je v pravidle určená jej názvom. Názvy akcií sú definované v osobitnom súbore, ktorého formát je popísaný v kapitole 5.2.

```
action := action_name
```

Kľúčové slovo quick znamená, že pravidlo má nastavený príznak quick.

Rozhranie je určené desiatkovou konštantou. S použitím daného hardvéru to môže byť číslo od 0 do 3.

```
interface := ['!'] dec_const
```

Protokol je zadaný číslom, tak ako sú uvedené v súbore /etc/protocols.

```
protocol := dec_const | const_list
```

Zdrojová a cieľová adresa môžu byť zadané jednou IP adresou, adresou negovanou pomocou operátora '!', zoznamom konštánt, tabuľkou alebo kľúčovým slovom 'any', ktoré znamená ľubovoľnú adresu.

```
src_addr := dst_addr := ( ['!'] ip_const) | list | table_name |
  'any'
```

Zdrojový a cieľový port sú špecifikované desiatkovou konštantou, zoznamom konštánt, tabuľkou prípadne rozsahom.

```
src_port := dst_port := dec_const | list | table_name | range
```

V rozsahu je možné použiť unárne operátory nerovná sa ('!='), menší ('<'), väčší ('>'), menší alebo rovný ('<='), väčší alebo rovný ('>=') a binárne operátory v rozsahu okrem hraníc ('><'), mimo rozsah ('<>') a v rozsahu vrátane hraničných hodnôt (':').

```
range := (('!=' | '<' | '>' | '<=' | '>=') dec_const) | (dec_const
  ('><' | '<>' | ':') dec_const)
```

TCP príznaky sú zadávané formou nastavené/kontrolované. Jednotlivé príznaky sú určené počiatočným písmenom. Zápis S/SA znamená kontrolovať príznaky SYN a ACK, kde práve iba SYN je nastavený.

```
set_flags := checked_flags := ('F' | 'S' | 'R' | 'P' | 'A' | 'U' |
  'E' | 'C')+
```

D.5 Popis filtra

Pravidlá sa vyhodnocujú sekvenčne od prvého po posledné, v poradí, v akom sú zapísané v súbore. V prípade, že paket vyhovuje pravidlu s kľúčovým slovom quick, je aplikovaná akcia podľa tohto pravidla. Inak sa pravidlá vyhodnocujú až po koniec súbora a akciu určuje posledné, s ktorým bola zistená zhoda.

Súbor s popisom filtra obsahuje každé pravidlo, prípadne definíciu tabuľky na samostatnom riadku. V prípade, že je riadok príliš dlhý je možné použiť znak '\' ktorý ruší význam bezprostredne nasledujúceho znaku '\n'. Definícia tabuľky musí predchádzať je použitie.

```
filter := row*
row := [table_definition | rule] '\n'
```

V popise filtra je možné použiť komentáre začínajúce znakom '#' a končiace novým riadkom alebo koncom súbora.

Príklad jednoduchého filtra, ktorý prepustí iba pakety zo známych adries smerujúcich na porty 0 až 1024 a všetky pakety na tieto adresy.

```
#known hosts
table <known_hosts> {147.229.0.0/16 213.15.18.3}

#implicit block
block

#pass traffic from known hosts to ports lesser than 1024
pass from <known_hosts> to any port <1024
#pass all traffic to known hosts
pass to <known_hosts>
```

D.6 Definícia akcií

Súbor s definíciami akcií obsahuje na jednotlivých riadkoch názov akcie a číselnú hodnotu výstupného záznamu vyhľadávacieho procesora.

```
action_definitions := action_definition*
action_definition := action_name value
action_name := ('a'-'z' | '_') ('a'-'z' | '0'-'9' | '_')*
value := dec_const | hexa_const
```

Príklad definícií:

```
block 0
pass 1
forward_1 $10
```

Príloha E Jazyk medzivrstvy

V tejto kapitole je popísaná syntax a sémantika jazyka medzivrstvy. Jazyk nerozlišuje veľké a malé písmená.

E.1 Konštanty

Je možné použiť 32-bitové nezáporné konštanty zapísané v desiatkovej alebo šestnástkovej sústave.

```
constant := dec_const | hexa_const
```

Desiatková konštanta je zložená zo znakov '0' až '9':

```
dec_const := dec_digit+
dec_digit := '0' - '9'
```

Šestnástková konštanta je znak 'h' nasledovaný sekvenciou šestnástkových číslic.

```
hexa_const := 'h' hexa_digit+
hexa_digit := dec_digit | 'a' - 'f'
```

E.2 Pravidlá

Pravidlo určuje akciu, ktorá sa vykoná ak sú splnené všetky zadané podmienky, teda akoby medzi jednotlivými podmienkami bola logická spojka „a zároveň“ (AND).

```
rule := action condition* ';'
```

Akcia je reprezentovaná 32-bitovou hodnotou, ktorá je výstupom vyhľadávacieho procesora. Táto hodnota môže byť zložená z niekoľkých konštánt. Jednotlivé hodnoty sa zapisujú zľava od najviac významnej. Je možné určiť koľko bitov sprava sa z jednotlivých konštánt použije na popis akcie. Implicitne je použitých celých 32 bitov šírky. Zápis { h1 h2/4 h34/8 } znamená 32-bitovú akciu s hodnotou 0x00001234.

```
action := '{' ( constant ['/' width] )+ '}'
width := constant
```

Podmienkou môže byť test na zhodu alebo test na príslušnosť do intervalu.

```
condition := exact_match | range_match
```

V prípade testu na zhodu je možné pomocou bitovej masky definovať, ktoré bity premennej sa majú porovnávať, a ktoré sa majú pri porovnaní ignorovať.

```
exact_match := variable '=' constant ['/' bitmask]
bitmask := constant
```

Pri teste na príslušnosť do intervalu sa porovnáva hodnota premennej s danou konštantou.

```
range_match := variable range_operator constant
range_operator := '>' | '<' | '>=' | '<=' | '!='
```

Premenná je postupnosť mien registrov uzavrená v zložených zátvorkách. Hodnota premennej sa získa uložením bitov jednotlivých 16-bitových registrov za seba. Registere sú usporiadané zľava s klesajúcim významom pre hodnotu premennej. Príklad: Ak by sme mali premennú zloženú z dvoch registrov { r0 r1 }, kde r0 = 0x12AA a r1 = 0x0CBB, potom hodnota celej premennej bude 0x12AA0CBB.

```
variable := '{' register_name+ '}'
```

```
register_name := 'r' register_number
```

E.3 Definície

Definície slúžia na zjednodušenie písania a zlepšenie čitateľnosti pravidiel. Je možné definovať názvy premenných a konštánt. Tieto názvy je potom možné použiť v zápise pravidiel všade tam, kde je potrebné zadať premennú prípadne konštantu. Efektívna dĺžka názvu je 255 znakov.

```
definition := variable_name_definition | constant_name_definition
variable_name_definition := variable_name variable
variable_name := '@' ('a'-'z' | '0'-'9' | '_')+
constant_name_definition := constant_name constant
constant_name := '@' ('a'-'z' | '0'-'9' | '_')+
```

E.4 Program pre vyhľadávací procesor

Program pre filter je zložený z definičnej časti a zoznamu pravidiel.

```
program := definition_list rule_list
```

Definičná sekcia je označená reťazcom „:definitions“. Za ním nasledujú jednotlivé definície.

```
definition_list := ':definitions' definition*
```

Zoznam pravidiel začína reťazcom „:rules“. Akcia, ktorá sa s paketom vykoná, je určená na základe prvého pravidla, v ktorom sú splnené všetky podmienky. V prípade, že takéto pravidlo neexistuje, je akcia určená implicitnou hodnotou 0. To je možné si predstaviť tak, akoby bolo na konci súbora pravidlo „{ 0 };“.

```
rule_list := ':rules' rule*
```

Je možné používať komentáre začínajúce symbolom mriežka ('#') a končiace novým riadkom alebo koncom zdrojového súbora.

Príklad programu:

```
:definitions
#variables
@L2 { r0 }
@L3 { r1 }
@SRC_IPV4 { r11 r10 }
@DST_IPV4 { r19 r18 }

# actions
@BLOCK 0
@PASS 1

:rules
#pass traffic to the specified Ipv4 address
{ @PASS } @L3 = h0000/h8001 @DST_IPV4 = hD50F1203;

#default block
{ @BLOCK };
```

Príloha F Príklad typického filtra

```
#filter for organization with servers and workstations connected to
  internet

#port 0 = internet
#port 1 = servers
#port 2 = workstations

#tables
#servers
table <servers> { 10.0.1.3 10.0.1.4}
#workstations
table <workstations> { 10.0.2.0/24 }
#blocked internet access
table <blocked_workstations> { 10.0.2.15 10.0.2.20 }

#default block
block

#antispoof
block quick on !1 from <servers>
block quick on !2 from <workstations>

#block traffic between servers through filter
block quick on 1 from <servers> to <servers>
#block traffic between workstations through filter
block quick on 2 from <workstations> to <workstations>

#allow traffic for workstations to internet
pass_hw0 on 2 from <workstations>
pass_hw2 on 0 to <workstations>

#block internet to blocked workstations
block from <blocked_workstations>
block to <blocked_workstations>

#allow traffic for server services (HTTP, SSH) available from
  anywhere
pass_hw1 proto tcp to <servers> port {22,80}
pass_hw0 on 0 proto tcp from <servers>
pass_hw2 on 0 proto tcp from <servers> to <workstations>

#allow traffic for server services (NNTP) available from
  workstations
pass_hw1 on 2 proto tcp from <workstations> to <servers> port 119
pass_hw2 on 1 proto tcp from <servers> to <workstations>
```