VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

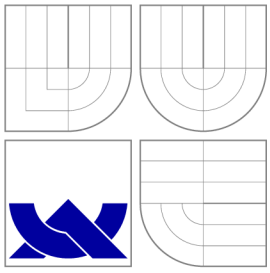
OPTIMALIZACE KLASIFIKAČNÍCH ALGORITMŮ ZALOŽENÝCH NA KARTÉZSKÉM SOUČINU

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

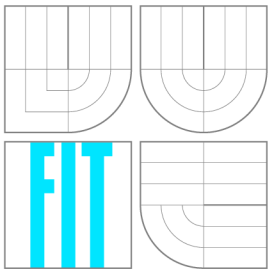
AUTOR PRÁCE
AUTHOR

MICHAL KAJAN

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

OPTIMALIZACE KLASIFIKAČNÍCH ALGORITMŮ ZALOŽENÝCH NA KARTÉZSKÉM SOUČINU

OPTIMIZATION OF CROSSPRODUCT-BASED CLASSIFICATION ALGORITHMS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — MICHAL KAJAN
AUTHOR

VEDOUCÍ PRÁCE — Ing. VIKTOR PUŠ
SUPERVISOR

BRNO 2009

Abstrakt

Tato diplomová práce se zabývá problematikou klasifikace paketů v počítačových sítích. Přibližuje problém klasifikace spolu s požadavky kladenými na klasifikační algoritmy. Popsány jsou různé přístupy ke klasifikaci paketů a jsou přiblíženy konkrétní příklady moderních algoritmů implementovatelných obvodově spolu s jejich vlastnostmi. Pozornost je věnována algoritmům kartézského součinu, jejichž výhodou je vysoká rychlost, ale mají problém s velkými paměťovými nároky. Představeny jsou metody optimalizace těchto algoritmů založené na prohledávání stavového prostoru a to redukcí původní sady filtrovacích pravidel jejich výběrem do asociativní paměti. Práce také ilustruje využití asociativní paměti jako flexibilní možnost ke klasifikaci a možnost implementace takovéto paměti přímo na čipu.

Klíčová slova

klasifikace paketů, algoritmy kartézskeho součinu, hardware, asociativní paměť, stavový prostor

Abstract

This thesis deals with the packet classification problem in computer networks. It introduces packet classification along with the demands on classification algorithms. Different approaches to packet classification and several concrete examples of modern classification algorithms with their properties are described. The aim is on algorithms which can be implemented in hardware. Crossproduct-based algorithms are described in more detail whose biggest advantage is classification speed, but their disadvantage consists in great memory requirements. Several optimization methods based on state space search are presented. These optimization methods are based on reduction of original ruleset by selecting a small number of rules to associative memory. Lastly, utilization of associative memory as a flexible part of classification is illustrated together with the potential hardware implementation of such memory directly on a chip.

Keywords

packet classification, crossproduct-based algorithms, hardware, associative memory, state space

Citace

Michal Kajan: Optimalizace klasifikačních algoritmů
založených na kartézském součinu, diplomová práce, Brno, FIT VUT v Brně, 2009

Optimalizace klasifikačních algoritmů založených na kartézském součinu

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Viktora Puše a uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Michal Kajan
26. května 2009

Poděkování

Chtěl bych vyjádřit poděkování svému vedoucímu Ing. Viktoru Pušovi za odbornou pomoc a neocenitelné rady při řešení této diplomové práce. Mé poděkování patří i kolegům z projektu Liberouter za cenné konzultace, především však Bc. Andrejovi Hankovi.

Obsah

Kapitola 1

Úvod

V posledných rokoch sme boli svedkami neuveriteľného rozvoja Internetu a počítačových sietí. Internet sa za posledné roky stal významnou infraštruktúrou použitou pre komerčné účely, čím sa aj výrazne zmenili požiadavky pripojených užívateľov v zmysle požadovaného výkonu, bezpečnosti a dostupných služieb.

So vzrastajúcim počtom užívateľov rástla aj potreba na čoraz vyššie prenosové rýchlosti z dôvodu množstva poskytovaných služieb a prenášaných dát. Pre koncových užívateľov sa dnes stávajú bežne dostupné pripojenia s rýchlosťami v rádovo desiatkach až stovkách Mb/s a na chrbticových sieťach poskytovateľov pripojenia pracujú linky na rýchlostiach až niekoľko desiatok Gb/s. Cieľom poskytovateľov služieb je využiť dostupnú zdieľanú sieťovú infraštruktúru pre poskytovanie rozmanitých služieb svojim klientom, ktoré by bolo možné spoplatňovať v závislosti na type služby či meniacich sa potrieb zákazníkov. Práve z toho dôvodu bolo potrebné navrhnúť spôsoby oddelenia sieťovej prevádzky týkajúcej sa jednotlivých užívateľov, mechanizmy zabraňujúce prieniku neautorizovaného prístupu do určitých častí využívanej siete či prispôsobovania dostupnej šírky pásma v súlade s požiadavkami a potrebami zákazníkov a účtovaním služieb.

Pre uvedenú diferenciáciu sieťovej prevádzky je nutné spracovať hlavičky prenášaných paketov a rozhodovať o príslušnej akcii vykonanej s každým paketom. Kľúčovým mechanizmom sa preto stala klasifikácia paketov (alebo tiež filtrovanie paketov). Pri klasifikácii paketov dochádza k porovnaniu obsahu hlavičiek paketov voči vytvorenej databáze pravidiel s cieľom nájsť pravidlo s najvyššou prioritou, ktorému paket zodpovedá a vykonať definovanú akciu. Takou môže byť zahadzovanie neželaných paketov prenášaných sieťou, presmerovanie sieťovej prevádzky, rozhodnutie o smerovaní paketu založené aj na iných položkách hlavičky ako cieľovej adrese, použitie systému QoS (Quality of Service) pre určenie priority rôznych typov prevádzky. Význam klasifikácie narástol aj z dôvodu väčšieho využívania virtuálnych privátnych sietí (VPN - *Virtual Private Networks*) a poskytovateľ služieb musí zabezpečiť bezproblémové fungovanie každej vytvorenej VPN siete pre každého zákazníka, pre tieto činnosti je potrebné pakety klasifikovať a rozhodnúť o ďalšej akcii [7].

Klasifikátor paketov môže byť implementovaný programovo alebo obvodovo. Rýchlosť klasifikátorov hrá významnú úlohu predovšetkým vo vzťahu k dnes dostupným prenosovým rýchlostiam a je zrejmé, že klasifikátor vyvinutý ako program bežiaci na bežne dostupnom počítači nedokáže nároky na rýchlosť uspokojiť. Preto sa hľadajú alternatívy vo forme klasifikátorov vytvorených obvodovo vo forme ASIC (*Application Specific Integrated Circuit*) alebo v FPGA (*Field Programmable Gate Array*) čipov. Hardvérová implementácia nesie so sebou výhodu výrazne vyššieho výkonu (i za cenu zložitejšieho návrhu a následného vývoja) z dôvodu možnosti využitia paralelizmu a návrhu obvodu pre konkrétny účel. Táto

práca sa zameriava na postupy klasifikácie vhodné pre obvodovú implementáciu.

Bolo vyvinutých a popísaných množstvo algoritmov, ktoré využívajú rôzne prístupy: bitový paralelizmus [7], delenie n-rozmerného priestoru do menších podpriestorov [19, 13], kartézsky súčin [3], [16], rozhodovacie stromy [14] či iné.

V druhej kapitole je priblížená problematika klasifikácie paketov spolu s požiadavkami kladenými na klasifikačný algoritmus. Tretia kapitola obsahuje popis najčastejšie používaných prístupov pri klasifikácii paketov spolu s príkladmi vyvinutých algoritmov a ich vlastnosťami. Štvrtá kapitola prináša prehľad metód pre redukciu pamäťovej náročnosti založených na prehľadávaní stavového priestoru a možnosti obvodovej implementácie dvoch druhov asociatívnych pamätí použiteľných pre klasifikáciu paketov.

Kapitola 2

Teoretický rozbor

2.1 Klasifikácia

Vstupom klasifikácie paketov sú jednotlivé pravidlá (filtre) a pakety. Samotný problém klasifikácie paketov je možno definovať ako určenie vyhovujúceho pravidla (obvykle s definovanou prioritou). Políčka hlavičky každého paketu potom slúžia ako informácia pre porovnanie s jednotlivými položkami pravidla.

Klasifikátor, t.j. sada pravidiel je definovaná ako konečná množina pravidiel $\{R_1, R_2 \dots R_N\}$. Každé takéto pravidlo je reprezentované K hodnotami, jednotlivé hodnoty potom zodpovedajú príslušným políčkam v hlavičke paketu. Tieto hodnoty sú vlastne definované podmienky. Každá podmienka je reprezentovaná obvykle 3 rôznymi spôsobmi: presne definovaná hodnota, prefix alebo rozsah [13].

- presne definovaná hodnota: políčko paketu sa musí zhodovať s danou hodnotou (užitočné pre políčko typ protokolu a flag)
- prefix: hodnota zodpovedá prefixu nejakého políčka v hlavičke paketu, definuje sa presne niekoľko vyšších bitov daného slova, ostatné nižšie bity môžu takto nadobúdať rôzne hodnoty (užitočné pre definovanie konkrétnej podsiete)
- rozsah: hodnoty políčka paketu musia ležať v definovanom rozsahu (užitočné pre definíciu rozsahu hodnôt portov)

Každú z týchto možností je možné reprezentovať vo forme prefixov. A teda aj ľubovoľný rozsah je možné previesť do prefixového tvaru, pričom počet prefixov, ktoré môže pri tomto prevode vzniknúť je nanajvýš $2N - 2$ [4].

Ako príklad pre 4-bitové pole možno uviesť prefix 10*, ktorý reprezentuje rozsah $[1000_2, 1011_2] = [8, 11]$. Podobne, pre 16-bitovú hodnotu čísla portu rozsah ≤ 1023 môžeme vyjadriť v podobe 000000* a rozsah > 1023 zase vyjadríme pomocou šiestich prefixov: 000001*, 00001*, 0001*, 001*, 01*, 1* [15].

Samotné pravidlo je možné definovať ako $(n+1)$-ticu $R : (p, a_1, a_2, \dots, a_n)$, kde $p \in N$ je priorita pravidla a a_x sú prefixy [10]. Ku každému pravidlu je priradená nejaká akcia act_i, ktorá určuje, ako naložiť s paketom, ktorý danému pravidlu vyhovel. Takáto akcia môže zahŕňať napr. preposlanie paketu na definované rozhranie, či zahodenie paketu [15].

2.2 Požiadavky na klasifikačný algoritmus

Efektívny klasifikačný algoritmus musí spĺňať niekoľko kritérií [7, 10]:

- **Rýchlosť:** klasifikačný algoritmus musí byť dostatočne rýchly, aby dokázal spracovať pakety na sieťach o rýchlosti rádovo niekoľko Gb/s či desiatok Gb/s (wire-speed). Väčšina prenášaných paketov má menšiu veľkosť ako typické TCP pakety veľkosti 552 bajtov, takmer polovica paketov má veľkosť v rozmedzí od 40 do 44 bajtov [7]. Tieto pozostávajú z potvrdzovania TCP spojení (angl. *acknowledgments*) a TCP kontrolných segmentov. Najdlhšia doba potrebná pre klasifikáciu musí byť taká, aby sa na prenosovej rýchlosti spracovali všetky prichádzajúce pakety bez ohľadu na ich typ a veľkosť. Táto doba závisí predovšetkým na dobe prístupu do pamäti, zložitosť väčšiny klasifikačných algoritmov sa hodnotí podľa najväčšieho počtu prístupov do pamäti (samozrejme je možné použiť i iné vlastnosti). Tabuľka 2.1 udáva maximálnu dobu potrebnú pre spracovanie 1 paketu dĺžky 64 B, čo je minimálna dĺžka v sieti typu Ethernet.

- **Pravidlá:** Klasifikačné pravidlá musia byť založené na niekoľkých políčkach hlavičky paketu. Medzi takéto políčka môžu patriť zdrojová a cieľová IP adresa, zdrojové a cieľové porty, typ protokolu, TCP flagy a iné. Pomocou pravidiel musí byť možné vyjadriť aj rozsahy a nie len presné hodnoty či jednoduché prefixy. Musí existovať možnosť priradiť jednotlivým pravidlám prioritu, pretože paket môže vyhovieť viacerým pravidlám v databázi. Pomocou priority je potom možné definitívne určiť pravidlo a príslušnú akciu viažucu sa k tomuto pravidlu, ktorá sa nad daným paketom vykoná. Počet pravidiel, ktoré môžeme do systému uložiť, by mal byť čo najvyšší.

- **Pamäťová náročnosť:** Veľkosť pamäti ovplyvňuje rýchlosť klasifikácie, veľkosť spotrebovanej pamäti vplýva na cenu konečného riešenia. Najčastejšie sa udáva v bajtoch na pravidlo.

- **Obvodová implementácia:** Pre spracovanie paketov na prenosových rýchlostiach musí byť možné klasifikačný algoritmus realizovať v hardvéri (s využitím FPGA či ASIC čipov). V tomto prípade je nutné zohľadňovať aj zabratú plochu na čipe, viac zdrojov vyžaduje drahšie čipy, čím sa celkové riešenie opäť predražuje.

Prenosová rýchlosť [Gb/s]	**Paketová rýchlosť [paketov/s]**	**Doba spracovania 1 paketu [ns/paket]**
1	1 488 095	672
4	5 952 380	168
10	14 880 952	67,2
40	59 523 809	16,8
100	148 809 523	6,72

Tabuľka 2.1: Doba potrebná pre spracovanie paketov dĺžky 64 bajtov (+12 bajtov medzera medzi rámcami a +8 bajtov preambula) pre rôzne prenosové rýchlosti

Kapitola 3

Klasifikačné algoritmy

V tejto kapitole si priblížime niekoľko klasifikačných algoritmov, ktoré k problému klasifikácie pristupujú rôznymi spôsobmi.

3.1 Naivný klasifikačný algoritmus

Tento algoritmus je charakteristický svojím intuitívnym prístupom ku klasifikácii. Je založený na použití lineárneho zoznamu všetkých pravidiel, kde hodnoty jednotlivých polí sú prevedené do prefixového tvaru. Pri vyhľadávaní sa postupne prechádzajú položky tohto zoznamu a vracia sa najdlhší zhodný prefix. Položky je možné ľubovoľne vkladať a odoberať, čo je jednou z vlastností tejto dátovej štruktúry. Pamäťová náročnosť tohto spôsobu je iba $O(N)$, kde N predstavuje počet prefixov. Časová zložitosť je lineárna, takže s rastúcim počtom prefixov rastie aj doba potrebná k vyhľadaniu najdlhšieho prefixu, čo však bráni praktickému použitiu tejto metódy. Priemerný čas vyhľadávania je možné znížiť, ak sa prefixy usporiadajú podľa poradia s klesajúcou dĺžkou.

Ďalším možným vylepšením je použitie cache pamätí. Do cache pamäti sa ukladajú celé hlavičky paketov pre urýchlenie vyhľadávania pre ďalšie prichádzajúce pakety. Ukladanie celých hlavičiek však nemusí viesť k významnému zvýšeniu výkonu — môže dochádzať k častým výpadkom, čím sa čas vyhľadávania ešte viac predlžuje. Použitie cache pamätí však prináša aj istú nutnú réžiu, čo spolu s nízkym počtom úspechov vyhľadania v cache pamäti (angl. *hit-rate*) počas prehľadávania môže viesť k degradácii celkového výkonu. Navyše, veľké rozdiely v dobe vyhľadania v cache pamäti pre rôzne typy paketov bránia využitiu tohto spôsobu pre obvodovú implementáciu. V dnes dostupných zariadeniach sa už tento prístup v podstate vôbec nevyužíva [4].

Ďalším príkladom naivného prístupu ku klasifikácii je uloženie čísla zodpovedajúceho pravidla do pamäte pre každý možný paket. Takto sa ako adresa do pamäti použijú vyextrahované položky z hlavičky paketu spojené do jedného slova a na takto vytvorenej adrese je potom možné nájsť číslo daného pravidla. Tento prístup je síce možné považovať za najrýchlejší možný (časová zložitosť $O(1)$), ale s obrovským nárokmi na kapacitu pamäti. Potrebná kapacita nie je a ani v blízkej budúcnosti dostupná nebude, preto je tento spôsob pre praktickú implementáciu úplne nevhodný [10].

Lineárne prehľadávanie je úzkym miestom pri klasifikácii, hoci sa takéto prehľadávanie môže týkať iba malej časti paketov. V ďalších podkapitolách si priblížime iné metódy, ktoré pre klasifikáciu používajú sofistikovanejšie spôsoby.

3.2 TCAM pamäte

Plne asociatívna pamäť (angl. Content Addressable Memory) môže byť využitá pre vyhľadávanie presne definovanej hodnoty a to počas jedného hodinového taktu. Vstupom do pamäti tohto typu je hodnota kľúča, podľa ktorej sa bude v pamäti vyhľadávať. Táto hodnota sa porovná so všetkými hodnotami uloženými v pamäti (porovnanie prebieha paralelne) a na výstupe sa objaví adresa, na ktorej došlo k zhode.

Variantou CAM pamätí sú pamäte TCAM (*Ternary Content Adressable Memory*). Ternárne CAM pamäte pracujú okrem dvoch typických stavov ešte s tretím — *„don't care"*. Položky sa do TCAM pamäte ukladajú ako dvojice *(hodnota, maska)*. Týmto spôsobom TCAM pamäť priamo podporuje vyhľadávanie prefixov. Do pamäte sa uložia položky pravidla vo forme prefixov a vyhľadanie potom prebieha paralelne pre celý obsah pamäti.

TCAM pamäte sú charakteristické svojím vysokým výkonom, nezávisle na uložených hodnotách (medzi dve najdôležitejšie výhody možno teda zaradiť *rýchlosť* a *determinizmus* — vysoký výkon je dostupný pre všetky typy pravidiel, nie len pre „typické sady"). Najnovšie TCAM pamäte dokážu realizovať vyhľadávanie pre viac ako 100 miliónov paketov za sekundu.

Nevýhodou pamätí tohto typu je malá kapacita (CAM pamäte majú nižšiu hustotu, množstva prvkov na jednotku plochy čipu, zároveň sa určitá kapacita spotrebuje aj na rozklad pravidiel používajúcich rozsahy hodnôt do prefixovej podoby, čím sa jedno pravidlo v definovanej sade rozgeneruje na viac pravidiel), vysoký príkon (z dôvodu paralelného porovnávania pre celý obsah) a cena — cena za bit v TCAM pamäti je až 15-krát vyššia ako cena porovnateľných statických pamätí (SRAM) a na jeden prístup spotrebujú až 50-krát viac príkonu [3, 13]. Tabuľka 3.1 porovnáva vlastnosti pamätí TCAM a SRAM.

	TCAM (18 Mb čip)	**SRAM (18 Mb čip)**
Maximálny kmitočet	266 MHz	400 MHz
Veľkosť bunky (počet tranzistorov na bit)	16	6
Príkon	$12 \sim 15$ W	≈ 0.1 W

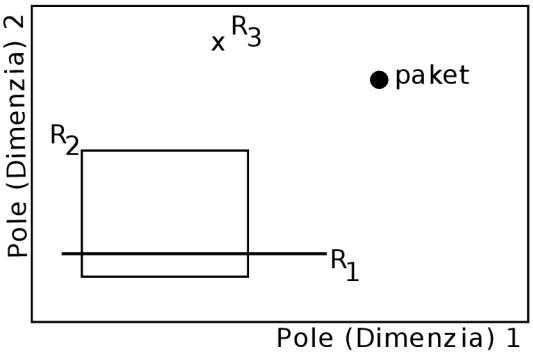
Tabuľka 3.1: Porovnanie technológií TCAM a SRAM [5]

Napriek uvedeným nevýhodám sa tieto pamäte uplatňujú aj v komerčnej sfére, každopádne však pre tieto nevýhody má zmysel pokračovať vo vývoji nových algoritmov pre klasifikáciu, ktoré namiesto TCAM pamätí používajú SRAM či DRAM pamäte.

3.3 Klasifikácia ako geometrický problém

Problém klasifikácie je možné vnímať aj ako geometrický problém. Ak máme umiestnený bod v N-rozmernom priestore, v ktorom sú zároveň umiestnené útvary, tak cieľom je nájsť všetky také útvary, ktoré tento bod obsahujú. Ako príklad môžeme uviesť 32-bitový prefix 00*, ktorý na číselnej osi reprezentuje rozsah adries od $000\ldots00_2$ do $001\ldots11_2$. Keď uvážime, že jednotlivé prefixy môžeme považovať za segmenty číselnej osi, ďalšou úvahou dospejeme k myšlienke, že pravidlá s dvoma podmienkami budú reprezentované ako obdĺžniky, pravidlá s tromi podmienkami ako trojrozmerné teleso. Z geometrického hľadiska je teda cieľom obmedziť sa na nájdenie najmenšieho útvaru, ktorý obsahuje bod umiestnený v priestore, pričom tieto útvary sa môžu ľubovoľne prekrývať. Bod v priestore

je teda paket, jednotlivé útvary zodpovedajú pravidlám [13]. Hľadáme pravidlo s najvyššou prioritou, ktoré danému paketu vyhovuje. Takáto geometrická reprezentácia je názorne zobrazená na obrázku 3.1, kde uvažujeme 2D priestor (klasifikáciu podľa dvoch polí). Označenie R_i reprezentuje pravidlá.



Obrázok 3.1: Geometrická reprezentácia klasifikácie

Samozrejme, pre praktické potreby je nutné uvažovať o priestore väčších rozmerov, pretože sa klasifikuje podľa viacerých polí hlavičky paketu.

3.3.1 Algoritmy založené na delení priestoru

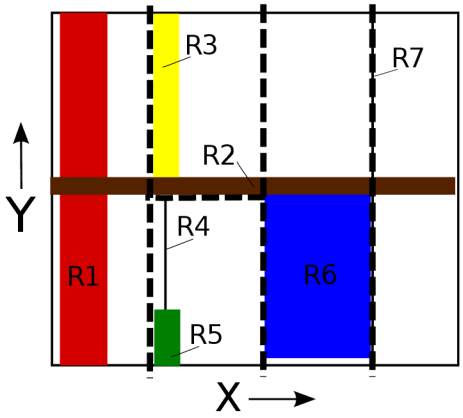
Priblížime si algoritmy založené na myšlienke delenia geometrického priestoru. Medzi tieto patrí algoritmus HiCuts a HyperCuts.

HiCuts algoritmus

HiCuts (*Hierarchical Intelligent Cuttings*) [4] využíva ako dátovú štruktúru nad sadou pravidiel rozhodovací strom. Táto štruktúra sa vytvorí a prispôsobí vlastnostiam pravidiel v sade počas procesu predspracovania. Vnútorné uzly stromu obsahujú informáciu pre ďalšie smerovanie pri priechode stromom a v listových uzloch stromu sa nachádza malé množstvo pravidiel. Výsledný počet pravidiel nachádzajúcich sa v listových uzloch je ohraničený zvolenou konštantou, ktorá sa označuje pojmom *binth* (z angl. *„bin-threshold“*). Výsledná podoba rozhodovacieho stromu — hĺbka stromu, stupeň každého uzlu a rozhodnutie o ďalšom smere prehľadávania sa určí počas fázy predspracovania vstupných pravidiel.

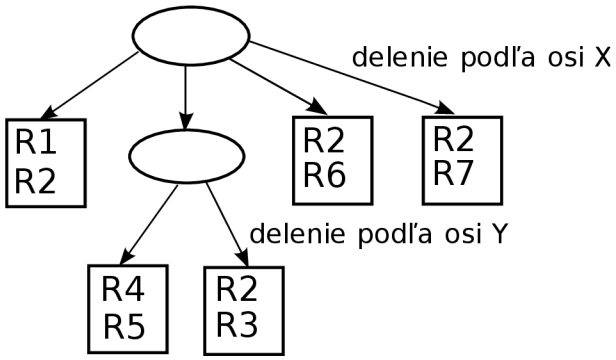
Vyhľadanie pravidla zodpovedajúceho danému paketu prebieha pomocou priechodu rozhodovacím stromom až do niektorého z listových uzlov. Ďalšie hľadanie je potom obmedzené na lineárne prehľadanie zoznamu pravidiel v danom listovom uzle a výber takého, ktorý predstavuje najlepšiu zhodu.

Použitie takejto dátovej štruktúry je možné zovšeobecniť do d-rozmerného priestoru. Každý interný uzol v rozhodovacieho stromu reprezentuje časť celého geometrického priestoru, kde bude realizované prehľadávanie. Samotný koreňový uzol takto reprezentuje celý d-rozmerný priestor. Priestor v uzle v je rozdelený rezom v jednom z d rozmerov. Jednotlivé podpriestory sú potom reprezentované ako potomkovia uzlu v, pričom každý z nich reprezentuje jednu hyperrovinu. Delenie podpriestorov je rekurzívne až dovtedy, pokiaľ daný podpriestor neobsahuje viac pravidiel, ako je stanovený prah hodnotou *binth*. V takom prípade to bude listový uzol a uložia doňho príslušné pravidlá.



Obrázok 3.2: Príklad delenia 2-rozmerného priestoru

Obrázok 3.2 zobrazuje delenie priestoru na niekoľko častí. Podľa osi X sa priestor rozdelí na 4 časti, ďalšie delenie je podľa osi Y. Symbolom R_i sú označené pravidlá a tie, ktoré zasahujú do viacerých častí sa objavia vo viacerých listových uzloch. Príslušný rozhodovací strom je možné vidieť na obrázku 3.3.



Obrázok 3.3: Príklad delenia 2-rozmerného priestoru

HyperCuts algoritmus

Algoritmus HyperCuts [13] nadväzuje na predchádzajúci algoritmus, takisto využíva rozhodovacie stromy. Na rozdiel od HiCuts algoritmu však každý uzol rozhodovacieho stromu reprezentuje k-rozmernú hyperkocku. Použitie ďalšieho stupňa voľnosti spolu s využitím novej heuristiky pre tvorbu stromu viedlo k zlepšeniu vlastností tohto typu klasifikácie oproti pôvodnému HiCuts algoritmu.

Každý interný uzol stromu v tomto algoritme predstavuje rozhodnutie o delení priestoru nad najviac reprezentatívnymi dimenziami (oproti HiCuts, kde sa používa iba 1 rozmer). Aktuálne delená sada pravidiel sa takto rozdelí podľa hodnôt jedného alebo viacerých polí pravidla. Pre každý zvolený rozmer sa vypočíta počet rezov, ktorý závisí stanovenom množstve pamäti pre rozhodovací strom.

Klasifikácia prebieha rovnako — pre každý prichádzajúci paket je potrebný priechod stromom na základe hodnôt jednotlivých položiek hlavičky paketu až do listového uzlu.

V listovom uzle sa postupne prejde zoznam uložených pravidiel a vyberie sa také, čo najviac zodpovedá danému paketu.

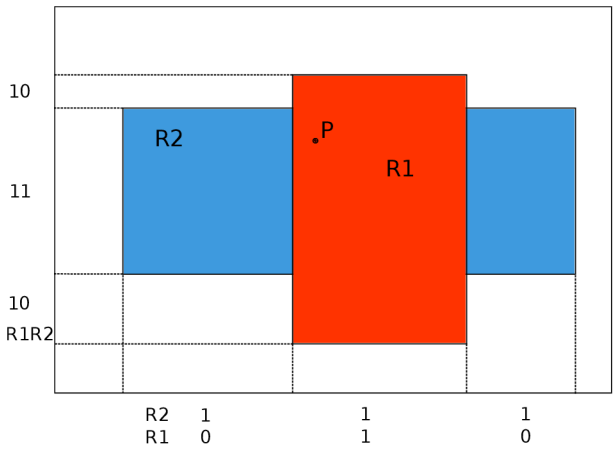
Má nižšiu pamäťovú náročnosť a dosahuje vyšších rýchlostí pri vyhľadávaní. Pri priechode stromom, vyžaduje iba jeden prístup do pamäti pre každý uzol. Navyše, celý algoritmus je možné plne reťaziť.

Bit-Vector algoritmus

Tento algoritmus má dve fázy. Prvou je spracovanie sady pravidiel a druhou následné prehľadávanie. Jednotlivé pravidlá v sade definujú rozsahy možných hodnôt v jednotlivých položkách hlavičky paketu. Všetky rozsahy pre každé pravidlo sa pre každý rozmer k premietnu na číselnú os. Tým vznikne nanajvýš $2n + 1$ neprekrývajúcich sa intervalov na každej ose. Pre každý takýto interval sa vytvorí množina pravidiel taká, že do tejto množiny budú patriť všetky pravidlá, ktoré sa s daným intervalom prekrývajú v danej položke pravidla (dimenzii). Tieto množiny sú reprezentované ako bitové vektory dĺžky n (n je počet pravidiel), kde každý nastavený bit určuje prekryv pravidla s daným intervalom.

Vyhľadávanie potom prebieha tak, že pre každú dimenziu sa určí interval, do ktorého daná hodnota položky hlavičky pakety patrí. Je možné použiť binárne hľadanie, ktoré vyžaduje najviac $\lceil log_2(2n + 1) \rceil + 1$ porovnaní. Prehľadávanie môže prebehnúť nezávisle pre každý rozmer. Nad všetkými množinami získanými týmto hľadaním sa vykoná prienik. Obvodovo túto operáciu môžeme realizovať ako logický súčin medzi všetkými bitmapami vrátenými ako výsledok hľadania v každej dimenzii. Predpokladom je, že všetky pravidlá boli zoradené podľa priority, aby bolo možné nakoniec vrátiť pravidlo s najvyššou prioritou a určiť akciu.

Činnosť algoritmu je možné názorne zobraziť 3.4.



Obrázok 3.4: Klasifikácia algoritmom bit-vector

V tomto prípade použijeme klasifikáciu v dvoch dimenziách (podľa dvoch položiek). Jednotlivé pravidlá sa takto zobrazia ako obdĺžniky. Tie sa môžu ľubovoľne prekrývať. Hrany všetkých obdĺžnikov sa premietnu do oboch číselných os. V tomto prípade 2 obdĺžniky vytvorili 5 intervalov na každej ose. Každému intervalu bol priradený bitový vektor, v ktorom sú jednotlivé bity nastavené iba vtedy, ak sa príslušný obdĺžnik prekrýva s daným intervalom.

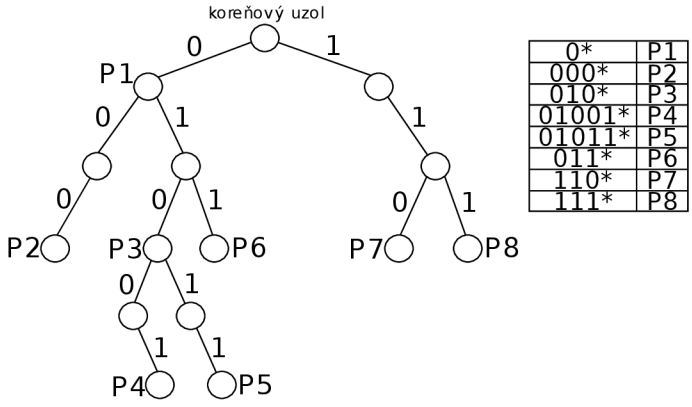
Predpokladajme klasifikáciu paketu P, ktorého umiestnenie v priestore je znázornené na obrázku. Určia sa intervaly, do ktorých paket patrí a v ďalšom kroku sa pomocou logického súčinu nad danými bitmapami určí pravidlo s najvyššou prioritou — bude to práve pravidlo 1 (podľa prvého nastaveného bitu vo výslednej bitmape) [7].

3.4 Algoritmy založené na dekompozícii problému

Klasifikácia paketov založená na dekompozícii problému prebieha v dvoch základných krokoch. V prvom kroku sa pre každé pole hlavičky paketu vyhľadá najdlhší zhodný prefix zo sady pravidiel. Výsledok tohoto kroku však nepostačuje na určenie príslušného pravidla, podľa ktorého sa paket ďalej spracuje. Ďalší krok preto pozostáva z metódy na presné určenie pravidla. V tejto časti si priblížime dátové štruktúry používané v niektorých algoritmoch a popíšeme niektoré z publikovaných.

3.4.1 Dátová štruktúra trie

Táto štruktúra je najbežnejšie používanou pre potreby vyhľadávania najdlhšieho zhodného prefixu. Je to stromová dátová štruktúra realizovaná vo forme binárneho stromu. Jednotlivé uzly reprezentujú jednotlivé prefixy. Hodnota prefixu zodpovedá ceste od koreňového uzlu k uzlu reprezentujúcemu daný prefix. Počas prehľadávania (priechod stromom) sa rozhodnutia o vetvení realizujú na základe hodnôt jednotlivých bitov prefixu. Jednobitový trie používa pre rozhodovanie o ďalšom postupe hodnotu jedného bitu, je však možné príslušné rozhodnutia prijímať aj na základe viacerých bitov. Každý uzol obsahuje hodnotu reprezentovaného prefixu a ukazovateľ na ďalší uzol. Každý z nich zároveň obsahuje informáciu, či daný uzol reprezentuje platný prefix a počas vyhľadávania sa v každom kroku platný prefix zapamätá a po skončení sa vráti ako výsledok vyhľadávania. Časová zložitosť vyhľadávania je lineárna s počtom bitov, nezávislá na počte uložených prefixov [5] [10]. Na obrázku 3.5 je zobrazená sada prefixov a k nej zodpovedajúci trie.



Obrázok 3.5: Príklad štruktúry trie pre sadu prefixov

3.4.2 Bloomov filter

Bloomov filter je jednoduchá dátová štruktúra na reprezentáciu množiny, pomocou ktorej je možné realizovať dotazy o existencii určitého prvku v danej množine. Bloomov filter

dovoľuje možnosť vzniku chyby, pri ktorom môže vrátiť pozitívnu odpoveď, napriek tomu, že sa daný prvok v množine nenachádza. Pravdepodobnosť takejto chyby v prípade, že je dostatočne malá, však vyváži skutočnosť, že táto štruktúra je pamäťovo veľmi úsporná [1].

Formálne: Množina $S = \{x_1, x_2, \ldots, x_n\}$ obsahujúca n prvkov je reprezentovaná ako pole v dĺžky m bitov. Na začiatku sú všetky bity tohto poľa nastavené na hodnotu 0. Ďalej, Bloomov filter používa sadu k hashovacích funkcií $h_1, h_2, \ldots, h_k$, pričom každá z nich má obor hodnôt $1, 2, \ldots, m$. Výstupy hashovacích funkcií $h_1(x), h_2(x), \ldots, h_k(x)$ slúžia ako ukazovatele na prvky uvedeného poľa pre nastavenie ich hodnôt. Potom pre ľubovoľný prvok $x \in S$ sú všetky bity na pozíciách $h_k(x)$ nastavené na 1 pre $1 \leq i \leq k$.

Na zistenie, či nejaký prvok y leží v množine S, postačí zistiť, či všetky pozície poľa dané výsledkom hashovacích funkcií $h_i(y)$ sú nastavené na 1. Treba však uvážiť, že po dostatočne veľkom množstve prvkov vložených do množiny S môže nastať situácia, kedy budú všetky prvky poľa nastavené na hodnotu 1, čo už povedie pozitívnej odpovedi na členstvo ľubovoľného prvku v množine.

Mnohé algoritmy vyžívajú Bloomove filtre pri klasifikácii. Jeden z nich, pre ktorý je táto štruktúra základnou a najdôležitejšou pri klasifikácii paketov, je popísaný v [1]. Používa variantu tzv. čítačových Bloomových filtrov, pri ktorej sa jednotlivé bity filtru nahradia čítačom. Pri každom vložení prvku, ktoré spôsobí, že by sa mala nastaviť hodnota už nastaveného bitu, tak sa inkrementuje hodnota čítača. Každá položka potom ukazuje sa lineárny zoznam priradených prvkov (je vhodné si všimnúť redundancie pri ukladaní prvkov). Tento algoritmus využíva optimalizovanú verziu čítačových Bloomových filtrov, pri ktorej sa prvok ukladá do pamäte iba jedenkrát.

3.4.3 Naivný algoritmus kartézskeho súčinu

Kartézsky súčin jednotlivých položiek je vhodný pre ľubovoľný typ hodnôt definovaných v pravidlách (presné hodnoty, prefixy, rozsahy). Základom je rozdelenie sady pravidiel do stĺpcov, pričom každý z nich obsahuje všetky odlišné prefixy v danom políčku. Potom pre každý paket určíme najdlhší zhodný prefix v každej položke hlavičky paketu a skombinujeme výsledky takéhoto vyhľadávania.

Vyhľadávanie najdlhšieho zhodného prefixu je možné realizovať pomocou LPM (angl. *Longest Prefix Match*) štruktúry nad každou položkou samostatne. Označme v_i najdlhší zhodný prefix v políčku f_i. Následne vytvoríme vyhľadávací kľúč v tvare $\langle v_1, v_2, \ldots, v_k \rangle$ ktorý bude vstupom do tabuľky kartézskych súčinov. Táto je obvykle implementovaná ako hash tabuľka. Pokiaľ je to kľúč, pre ktorý hash tabuľka obsahuje nejaké položky, vrátia sa identifikátory zodpovedajúcich pravidiel. Pre správnu činnosť tohto algoritmu je nutné hash tabuľku upraviť. S tým súvisí aj zavedenie pojmu *pseudopravidiel*.

Pre jednoduchosť si uveďme sadu 3 pravidiel, klasifikovať sa bude v tomto prípade iba nad dvoma položkami. Celý problém je zobrazený v tabuľkách 3.2, 3.3 a obrázku 3.6.

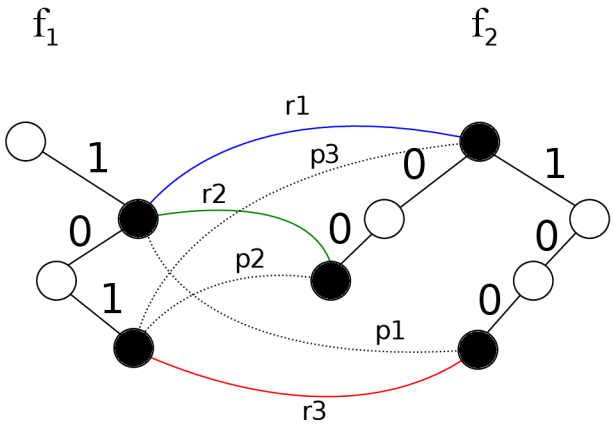
pravidlo	**pole** f_1	**pole** f_2
r1	1*	*
r2	1*	00*
r3	101*	100*

Tabuľka 3.2: Pôvodná sada pravidiel

pseudopravidlo	**pole** f_1	**pole** f_2	**pravidlo**
p1	1*	100*	r1
p2	101*	00*	r2,r1
p3	101*	*	r1

Tabuľka 3.3: Sada s pseudopravidlami

Uzly, ktoré sú platnými prefixami, sú vyznačené čiernou farbou. Prepojenie medzi dvoma uzlami dvoch trie štruktúr reprezentuje definované pravidlo. Teraz predpokladaj-



Obrázok 3.6: Pravidlá a pseudopravidlá

me, že budeme klasifikovať paket, ktorého prvé políčko vyhovuje najdlhšiemu prefixu 101* a druhé políčko prefixu 00*. V sade pravidiel však nie je obsiahnuté pravidlo $\langle 101*, 00* \rangle$. Stačí si však všimnúť, že prefix 1* je zároveň aj prefixom pre 101*, takže pri pokuse o vyhľadanie kľúča $\langle 101*, 00* \rangle$ by sme mali dostať pravidlo r_2. Pre zachovanie správnosti vyhľadávania pravidiel je potrebné do pôvodnej sady pridať nové pravidlá — *pseudopravidlá* a pripojiť k nim informáciu o čísle pravidla, ktorému zodpovedajú. V tomto prípade pridáme pseudopravidlo $p_1 : \langle 101*, 00* \rangle$ a pripojíme k nemu identifikátor pravidla r_2. Takto postupne pridáme ďalšie pseudopravidlá a vyhľadávanie bude správne. Výsledok obsahuje tabuľka 3.3.

Algoritmus kartézskeho súčinu je veľmi úsporný v počte nutných prístupov do pamäte. Tieto prístupy sa týkajú určenia najdlhšieho prefixu pre každé pole hlavičky a následného vyhľadania v hash tabuľke. Vyhľadanie pomocou LPM je možné vylepšiť použitím Bloomových filtrov. V tom prípade je možné dosiahnuť zníženie počtu prístupov až na približne jeden pre každú LPM. Klasifikáciu paketu je potom možné uskutočniť s veľmi vysokou pravdepodobnosťou iba v počte prístupov do pamäti daným počtom položiek, podľa ktorých sa vyhľadáva. Pre niektoré políčka, ako napr. typ protokolu však nie je nutné realizovať vyhľadávanie pomocou LPM, postačí priame vyhľadanie v malej pamäti na čipe.

Najväčším problémom pri tomto prístupe je však obrovské zvýšenie pôvodnej sady pravidiel o nové pseudopravidlá (teoreticky až exponenciálny nárast), pôvodná sada sa môže na základe experimentálnych výsledkov zväčšiť až 200-krát [3]. Určitými postupmi je však možné tento nežiaduci jav obmedziť.

3.4.4 MSCA algoritmus

Predchádzajúci algoritmus vedie k veľkým nárokom na množstvo spotrebovanej pamäte. Určitými technikami je však možné dosiahnuť výrazné zlepšenie oproti naivnému prístupu. Tento algoritmus sa nazýva *Multi-subset crossproduct*, voľne toto označenie možno preložiť ako „algoritmus kartézskeho súčinu založený na delení do podmnožín“. Pôvodnú sadu pravidiel je možné rozdeliť na menšie časti a uvažovať kartézsky súčin už iba v rámci jednotlivých častí. Rozdelením možno dosiahnuť veľmi výrazné zníženie počtu vzniknutých pseudopravidiel. Použitím samostatnej hash tabuľky pre každú podmnožinu pravidiel je možné vyhľadávať nezávisle výsledné pravidlo pre každú z nich. Mohlo by sa zdať, že tak-

to budú potrebné ďalšie prístupy do pamäte (jednak z dôvodu vyhľadávania najdlhšieho zhodného prefixu pre každú podmnožinu a jednak použitím samostatných hash tabuliek pre každú z nich), čo by mohlo znižovať výkon algoritmu. Tento algoritmus rieši ďalším trikom aj uvedené problémy.

Pôvodná LPM štruktúra sa upraví tak, že pre každú podmnožinu sa vytvorí tabuľka globálnych prefixov. Takáto tabuľka potom obsahuje iba unikátne prefixy z daného políčka pre každú podmnožinu. Pre každý takýto prefix z každej podmnožiny stačí určiť iba príslušnú dĺžku voči prefixu zaznamenanému v tabuľke. V niektorých podmnožinách sa však daný prefix z tabuľky nemusí nachádzať (ani vo forme jeho podprefixu), tabuľka v takýchto prípadoch obsahuje hodnotu NULL. Vďaka týmto úpravám k stačí použiť iba jednu LPM štruktúru pre každé políčko, podľa ktorého sa vyhľadáva.

Je možné obmedziť aj nutnosť použitia hash tabuľky pre jednotlivé výsledky z LPM vyhľadávania pre určenie správneho pravidla pomocou Bloomových filtrov. Použije sa sada Bloomových filtrov pre každú podmnožinu. Na vstupe majú hodnotu kľúča sformovaného z LPM vyhľadávacej fázy a výsledkom je výber iba tých hash tabuliek, v ktorých prebehne určenie všetkých zodpovedajúcich pravidiel a vyberie sa pravidlo s najvyššou prioritou.

Významnou časťou tohto algoritmu je rozdelenie pôvodnej množiny pravidiel do menších, v ktorých by vznikal iba veľmi malý zlomok pseudopravidiel voči pravidlám obsiahnutým v týchto množinách. Ďalej je potrebné uvážiť množstvo podmnožín, pretože viac podmnožín vedie zase na väčšie množstvo použitých Bloomových filtrov, čo zase vedie k vyššej spotrebe dostupných zdrojov na čipe. Cieľom je teda vytvoriť pre definovaný počet podmnožín tieto podmnožiny tak, aby vzniklo čo najmenšie množstvo pseudopravidiel. Algoritmus, ktorý celé delenie vykonáva, sa označuje skratkou NLTSS (*Nested Level Tuple Space Search*). Je založený na technike označovanej pojmom *Nested Level Tuple* (voľný preklad — *n-tica vnorenej úrovne*), pomocou ktorej dochádza k zníženiu počtu vytváraných podmnožín na stanovenú úroveň. Dosahuje to spájaním podmnožín a vytváraním kartézskeho súčinu medzi nimi.

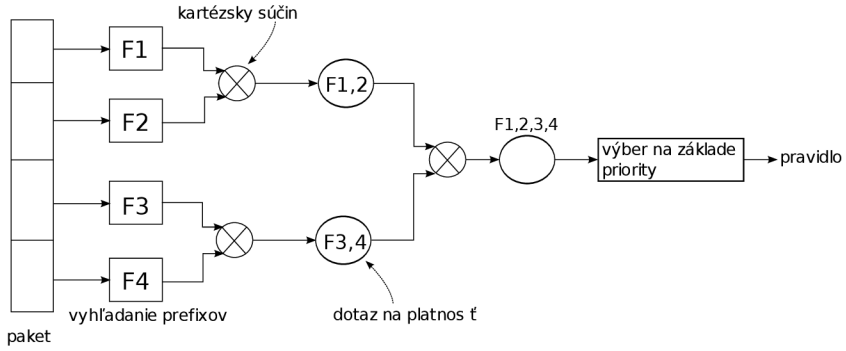
Pre každé políčko sa vytvára štruktúra podobná trie, nazývaná *Nested Level Tree*. Tá sa vytvorí úpravou pôvodného trie takým spôsobom, že uzly nezodpovedajúce platným prefixom sa odstránia a každý uzol reprezentujúci platný prefix sa spojí s najbližším predkom. Definuje sa pojem vnorenej úrovne (angl. *Nested Level*), ktorý zodpovedá počtu predchodcov uzlu reprezentujúceho platný prefix a ktoré sú zároveň tiež platnými prefixami v štruktúre Nested Level Tree. Po vytvorení takejto štruktúry pre každé políčko je možné určiť tzv. n-tice vnorených úrovní, ktoré sú asociované ku každému prefixu políčka príslušného pravidla. Všetky pravidlá obsiahnuté v jednej n-tici vnorených úrovní však medzi sebou nevytvárajú žiadne pseudopravidlá. Výsledný počet takýchto n-tíc potom udáva počet všetkých podmnožín obsahujúcich iba pravidlá nevytvárajúce žiadne pseudopravidlá. Tento počet však môže prevyšovať požadovanú hranicu na celkové množstvo podmnožín. Zníženie počtu sa dosiahne spájaním niektorých vnorených n-tíc a vytváraním kartézskeho súčinu medzi nimi. K takémuto spájaniu sa používa algoritmus NLTMC (*Nested Level Tuple Merging and Crossproduct*), ktorý vyhľadáva také vnorené n-tice, ktoré je možné medzi sebou spojiť. Je založený na pozorovaní, že väčšina pravidiel je rozložená iba do malého počtu takýchto vnorených n-tíc, čím je možné veľkú časť pravidiel pokryť iba malým počtom podmnožín. Spájanie vnorených n-tíc síce vedie k zníženiu počtu podmnožín, ale samo o sebe môže následne viesť k veľkému počtu vytvorených pseudopravidiel. Pri spájaní vnorených n-tíc sa do podmnožiny vloží pravidlo a určí sa počet pseudopravidiel, ktoré vygeneruje. Ak počet pseudopravidiel prekročí stanovený prah, takéto pravidlo sa vynechá. Označované je pojmom *„spoiler“*. Experimentálne výsledky viedli k zisteniu, že takýchto

pravidiel je približne $1\% - 2\%$ z celkového počtu. Nízky počet takýchto pravidiel umožňuje ich umiestnenie napr. do TCAM pamäte.

Pomocou algoritmu MSCA je možné obmedziť veľkosť celkovej sady pravidiel spolu s vytvorenými pseudopravidlami až na hodnoty v rozmedzí priemerne 1,2-1,4 násobku veľkosti pôvodnej sady pravidiel. Dôležitým krokom je rozdelenie množiny pravidiel do jednotlivých podmnožín, detailný popis algoritmu spolu so všetkými nutnými krokmi k správnej činnosti je obsiahnutý v [3].

3.4.5 DCFL algoritmus

Algoritmus *Distributed Crossproducting of Field Labels* [16] bol navrhnutý na základe skúmania vlastností reálne používaných sád pravidiel: počet unikátnych hodnôt v jednotlivých políčkach je relatívne malý oproti celkovému počtu pravidiel a súčasne počet unikátnych hodnôt, pri ktorých dochádza pre pakety k zhode je veľmi malý oproti celkovému počtu pravidiel. Využíva paralelné prehľadávanie pre každé políčko hlavičky paketu a určité techniky pre agregáciu jednotlivých výsledkov a ich spracovanie. DCFL algoritmus pozostáva z troch základných komponent: komponent zabezpečujúcich vyhľadávanie pre jednotlivé políčka, agregačnej siete pre spracovanie výsledkov z predchádzajúcej úrovne a komponenty pre určenie pravidla s najvyššou prioritou. Schéma je naznačená na obrázku 3.7.



Obrázok 3.7: Schéma činnosti algoritmu DCFL

V prvej úrovni dochádza k nezávislému porovnaniu hodnôt políčok hlavičky paketu voči daným políčkam medzi všetkými pravidlami. Takéto vyhľadanie sa môže realizovať LPM štruktúrou pre políčka obsahujúce IP adresy, pre protokol či TCP flagy možno použiť malú hash tabuľku a pod. Výsledky sú potom spracovávané v agregačnej sieti, ktorá určí všetky pravidlá, voči ktorým by mala nastať zhoda. Agregačná sieť skúma, či sa v niektorých pravidlách vyskytla kombinácia daných dvoch prefixov (kartézsky súčin sa vykonáva na dvoch množinách). Na tieto účely sa používajú polia Bloomových filtrov. Použitím viacerých takýchto filtrov je možné znížiť počet prístupov do pamäti. Nakoniec sa v poslednom kroku vyberie pravidlo s najvyššou prioritou.

Úzkym miestom tohto algoritmu je postupné vyhodnocovanie kartézskeho súčinu, z dôvodu množstva takto vzniknutých položiek, pre ktoré je nutné vyhľadávanie realizovať.

3.4.6 PHCA algoritmus

Ďalším z algoritmov kartézskeho súčinu založený na dekompozícii problému je algoritmus PHCA *(Perfect Hashing Crossproduct Algorithm)*. Je vytvorený pre účely obvodovej imple-

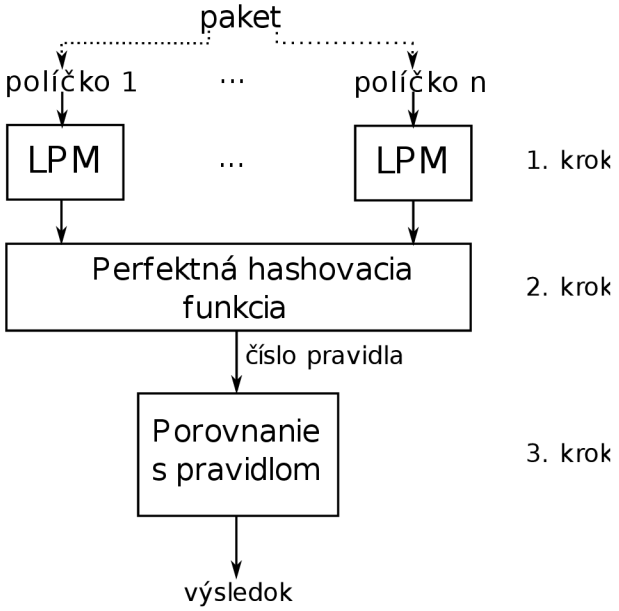
mentácie pre dosiahnutie čo najvyššej rýchlosti klasifikácie paketov. Cieľom algoritmu je dosiahnuť konštantnú dobu spracovania paketov spolu s dobrou škálovateľnosťou s dĺžkou filtrovacích pravidiel. Algoritmus vyžíva perfektné hashovacie funkcie pre vyhľadávanie v tabuľkách pravidiel v konštantnom čase [9].

Klasifikácia pomocou tohto algoritmu je rozdelená do 3 základných krokov:

- vyhľadanie najdlhšieho zhodného prefixu pre každé políčku paketu (LPM)
- mapovanie výsledkov predchádzajúceho kroku na číslo pravidla pomocou perfektnej hashovacej funkcie
- kontrola políčok paketu voči nájdenému pravidlu

Posledný krok je dôležitý z toho hľadiska, že pri mapovaní výsledkov LPM fáze hashovacia funkcia vždy namapuje daný prichádzajúci paket na nejaké pravidlo hoci tento paket žiadnemu pravidlu nevyhovuje. Preto je potrebné mať uložené aj všetky pôvodné pravidlá v nejakej tabuľke.

Uvedené kroky sú znázornené na obrázku 3.8



Obrázok 3.8: Jednotlivé kroky algoritmu PHCA

Základom tohto algoritmu je konštrukcia perfektnej hashovacej funkcie, ktorá realizuje mapovanie pseudopravidiel na príslušné pravidlá. Pre každé políčko sa vyhľadá najdlhší zhodný prefix, jednotlivé prefixy dohromady reprezentujú jedno možné pseudopravidlo. Paket nevyhovujúci žiadnemu pravidlu môže vytvoriť neplatné pseudopravidlo, ktoré sa namapuje na niektoré z pravidiel. V poslednej fázi však dochádza k porovnaniu výsledného pravidla s položkami daného prichádzajúceho pravidla a tento problém sa odstráni.

Použitá hashovacia funkcia sa konštruuje ako bezkolízna pomocou vytvárania neorientovaného acyklického grafu (množina kľúčov je dopredu známa). Hrana grafu zodpovedá jednému vstupnému slovu a uzol grafu výsledkom dvoch rôznych hashovacích funkcií.

Takáto hashovacia funkcia rieši problém zobrazenia určitého pravidla a všetkých jeho asociovaných pseudopravidiel na dané pravidlo. V podstate je tento spôsob založený na vytváraní zámerných kolízií. Ohodnotenie jednotlivých uzlov (vrcholov) grafu je uložené v tabuľke vrcholov.

Výsledná hashovacia funkcia pozostáva z 2 hashovacích funkcií, tabuľky vrcholov a sčítačky.

Spôsob klasifikácie použitím perfektných hashovacích funkcií podrobnejšie popisujú nasledovné body:

- z hlavičky paketu sa extrahujú hodnoty z príslušných políčok
- pre každé pole prebehne nezávisle (paralelne) vyhľadanie najdlhšieho zhodného prefixu
- výsledky sa zreťazia do jedného dátového slova
- vzniknuté dátové slovo je vstupom dvoch rôznych hashovacích funkcií
- podľa výstupných hodnôt sa z tabuľky vrcholov získajú ohodnotenia vrcholov acyklického grafu
- získané hodnoty sa sčítajú a výsledok reprezentuje číslo pravidla
- z tabuľky pravidiel sa získa pravidlo priradené k danému číslu a porovná sa s hodnotami v hlavičke paketu. Ak toto porovnanie dopadne úspešne, výsledkom klasifikácie je dané pravidlo, inak sa použije univerzálne pravidlo, ak takéto pravidlo bolo definované v sade pravidiel. Univerzálne pravidlo vyhovuje každému paketu.

Tento algoritmus má však nevýhodu v množstve potrebných pamäťových zdrojov spôsobených veľkým počtom vzniknutých pseudopravidiel. Skúmaním charakteristík vytvorených pseudopravidiel je možné identifikovať príčiny ich vzniku, ktoré môžeme rozdeliť do dvoch skupín:

- priama príčina
- nepriama príčina

Priamou príčinou je použitie všeobecných pravidiel (obvykle s nízkou prioritou), čo vedie k vzniku veľkého množstva ich špeciálnych prípadov — a tými sú práve pseudopravidlá. Nepriama príčina sa týka počtu unikátnych prefixov v jednotlivých dimenziách. Problémy spôsobuje prevod rozsahov na prefixy (rozsahy v pravidlách najčastejšie použité pre porty), kedy všetky pravidlá, ktoré pokrývajú celý rozsah musia vygenerovať ďalšie pravidlá pokrývajúce jednotlivé prefixy. Navyše, toto sa týka každej dimenzie, takže počet takto vytvorených pseudopravidiel môže byť enormný.

Cieľom je identifikovať pravidlá, ktoré vedú k vzniku veľkého množstva pseudopravidiel. Takéto pravidlá by potom bolo možné z pôvodnej množiny vyňať a umiestniť do malej asociatívnej pamäte priamo na čipe. Najjednoduchšou metódou je odstránenie tých pravidiel, ku ktorým je asociované najväčšie množstvo pseudopravidiel. Týmto postupom však nie je možné identifikovať pravidlá, ktoré spôsobujú expanziu pseudopravidiel tým, že zvyšujú počet unikátnych prefixov v jednotlivých dimenziách. Mohlo by sa zdať, že by postačilo otestovať všetky možnosti postupným odoberaním pravidiel a vyhodnocovaním

počtu pseudopravidiel. Je to však časovo najnáročnejší postup, preto by bolo výhodné vyvinúť nejakú heuristiku, ktorou by bolo možné identifikovať pravidlá, ktoré by sa umiestnili do asociatívnej pamäti a zároveň by postihovala obe uvedené pričiny vzniku pseudopravidiel. Spôsob takejto identifikácie problematických pravidiel však predstavuje netriviálny problém, ktorý nebol doteraz riešený.

3.5 Projekt NIFIC

V rámci projektu Liberouter pod záštitou združenia CESNET prebieha vývoj sieťovej platformy pre filtrovanie a preposielanie paketov na plných rýchlostiach na nasadených linkách bez straty paketov. Vyvíjané zariadenie je schopné pakety filtrovať, preposielať do operačného systému počítača, v ktorom je nasadené či preposielať pakety na jedno alebo viac výstupných sieťových rozhraní.

Klasifikácia paketov je založená na políčkach hlavičky paketov úrovní L2, L3 a L4 ISO/OSI sieťového modelu (Pôvodné políčka podľa ktorých sa klasifikuje boli rozšírené o zdrojovú a cieľovú MAC adresu, TCP flagy a číslo sieťového rozhrania. Takéto rozšírenie vyžaduje iba zväčšenie tabuľky pravidiel a neovplyvňuje rýchlosť klasifikácie).

Jadrom poslednej verzie je klasifikačný algoritmus PHCA [9] ako ukážka jeho praktickej obvodovej implementácie.

Algoritmus sa implementuje do FPGA čipu (Xilinx Virtex 5), dátové štruktúry pre LPM operácie a tabuľka pravidiel sú implementované priamo na čipe (buď s využitím distribuovaných alebo pomocou BlockRAM pamätí). Pre uloženie tabuľky vrcholov acyklického grafu sa používa externá statická pamäť (Cypress QDR-II SRAM)

Cieľom pri ďalšom vývoji je ďalšia optimalizácia za účelom zníženia pamäťovej náročnosti, čo by odstránilo nutnosť používať externé pamäte a využívať prostriedky dostupné výlučne na čipe.

Kapitola 4

Optimalizačné metódy

Nasledujúce optimalizačné metódy sú založené na redukcii pôvodnej sady pravidiel výberom určitého počtu pravidiel a ich umiestnením do asociatívnej pamäte. Cieľom takéhoto výberu je identifikácia takých pravidiel, ktoré spôsobujú najväčší nárast počtu pseudopravidiel.

Klasifikácia paketov je potom založená na využití pôvodného klasifikačného algoritmu spolu s klasifikáciou pomocou asociatívnej pamäti. Asociatívna pamäť pracuje paralelne s klasifikačným algoritmom, pričom na konci fáze klasifikácie postačí vybrať pravidlo s väčšou prioritou.

Ternárne asociatívne pamäte majú však nevýhody súvisiace s veľkou energetickou náročnosťou a vysokou cenou za jednotku plochy čipu. Je však možné tento prístup použiť pre výber určitého malého počtu pravidiel, čím sa uvedené nevýhody prevážia veľmi efektívnou technikou klasifikácie.

Problémom však zostáva správny výber takýchto pravidiel. Takýto prístup nebol ešte dostatočne preskúmaný a metóda, ktorá by presne odhalila pravidlá, ktoré spôsobujú najväčšiu expanziu totiž ešte nebola vyvinutá a publikovaná. V nasledujúcom texte budú priblížené metódy, ktoré výber pravidiel uskutočňujú prehľadávaním stavového priestoru vzniknutých pseudopravidiel. Tieto metódy možno považovať za jedny zo základných, ako k takejto optimalizácii pristupovať. Táto technika nemusí viesť k uspokojivým výsledkom, predstavuje však cestu, ktorou je možné sa uberať pri identifikácii problémových pravidiel a tým aj vývine dokonalejších metód.

4.1 Naivná optimalizácia hrubou silou

Touto metódou je možné pokryť celý stavový priestor týkajúci sa pseudopravidiel priradených jednotlivým pravidlám. Postupným skúšaním všetkých možných kombinácií výberov daného počtu pravidiel sa určí výber, ktorý dosiahne najvýznamnejšiu redukciu počtu pseudopravidiel.

Prakticky sa jedná o metódu hrubej sily, z matematického pohľadu je to klasická kombinatorická úloha —kombinácia bez opakovania: výber k prvkov z n prvkovej množiny:

$$C_k(n) = \binom{n}{k}$$

Výsledkom je počet všetkých takýchto kombinácií daných vzťahom:

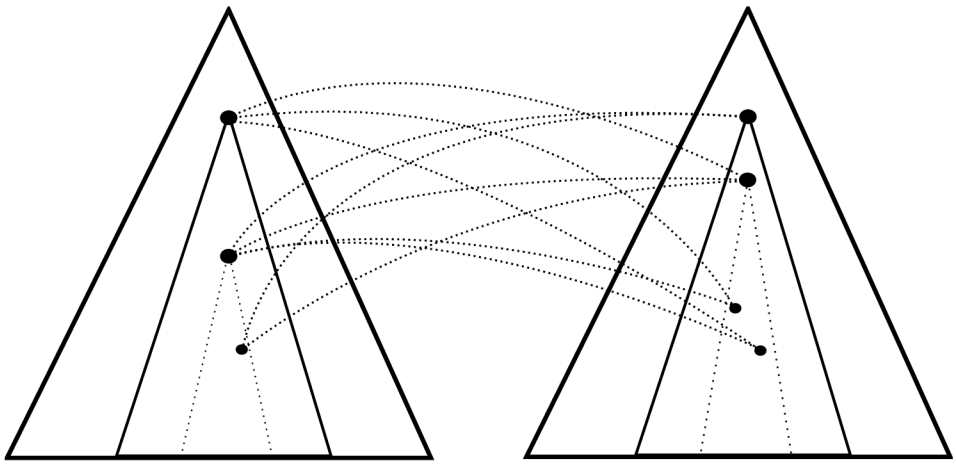
$$\frac{n!}{k!(n-k)!}$$

Tento prístup je možné použiť iba pre veľmi malé sady pravidiel, výpočtová náročnosť je totiž obrovská. Ako príklad možno uviesť výpočet pre sadu 500 pravidiel, z ktorej je potrebné vybrať vhodne 10 pravidiel a umiestniť ich do asociatívnej pamäte. Za predpokladu, že spracovanie jednej kombinácie by trvalo 1 takt a je realizované na frekvencii 3 GHz, tak samotný výpočet by trval takmer 2600 rokov!

Metóda hrubej sily je najpresnejšia, ale vzhľadom na povahu výpočtu aj najmenej vhodná pre praktické potreby. Preto je nutné hľadať iné prístupy.

4.2 Optimalizácia založená na odhade počtu pseudopravidiel

Ďalšou možnosťou je odhad počtu pseudopravidiel pre každé pravidlo v definovanej sade. Táto metóda nevykonáva žiadne prehľadávanie stavového priestoru. Je založená na vyhľadávaní špecifickejších prefixov pre každé políčko, podľa ktorého sa klasifikuje. Situáciu je názorne zobrazená na obr. 4.1 vo forme stromovej štruktúry.



Obrázok 4.1: Vyhľadávanie špecifickejších prefixov pre jednotlivé políčka

Obrázok znázorňuje odhad pseudopravidiel pre dve políčka (možno si predstaviť napr. zdrojovú a cieľovú IP adresu). V stromovej štruktúre je na najvyššej úrovni hodnota daného políčka, takto je možné určiť všetky prefixy, ktoré hodnota daného políčka pokrýva vzhľadom na ostatné pravidlá. Takto získané hodnoty pre každé políčko daného pravidla sa navzájom vynásobia a výsledný súčin potom reprezentuje odhad počtu priradených pseudopravidiel. Pre udržanie prehľadnosti nie sú na obrázku znázornené všetky väzby medzi hodnotami príslušných políčok.

Pseudokód tejto metódy je možné popísať nasledovne:

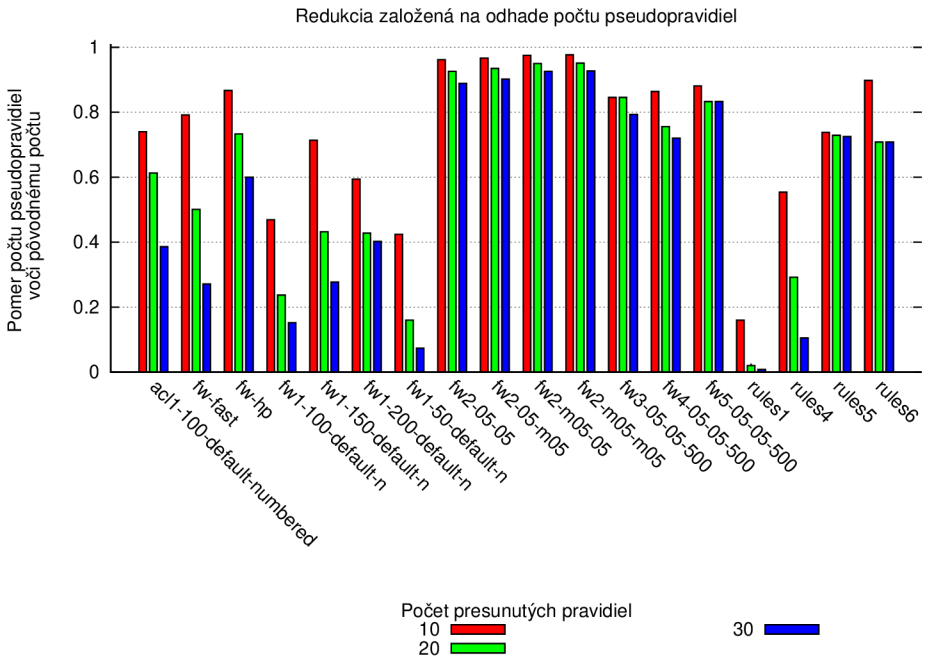
1: Vstup: množina pravidiel R, počet odstraňovaných pravidiel d
2: vytvor množinu pravidiel s najväčším počtom pseudopravidiel $R_{max} := \emptyset$
3: **for all** pravidlá r_i z množiny pravidiel R, $i \in \{1 \dots n\}$, $n = |R|$ **do**
4: **for all** políčka a_i pravidla r_i **do**
5: n_i := počet špecifickejších prefixov políčka a_i pravidla r_i zo všetkých ostatných pravidiel v políčku a_i
6: priraď pravidlu r_i súčin $n_1 \times n_2 \times \dots \times n_m$, m je počet políčok
7: **end for**
8: **end for**
9: vyber d pravidiel s najväčším odhadnutým počtom asociovaných pseudopravidiel a ulož ich do množiny R_{max}
10: Výstup: množina R_{max} obsahujúca d pravidiel s najväčším počtom priradených pseudopravidiel

Experimenty a zhodnotenie výsledkov

Metóda bola použitá na niekoľko sád filtrovacích pravidiel. Sady pravidiel *fw-fast.rul, fw-hp.rul a rules{1,4,5,6}.rul* sú reálne sady používané v univerzitnej sieti, ostatné sady sú syntetické a sú vytvorené nástrojom Classbench [17]. Výsledky dosiahnuté touto metódou sú v tabuľke 4.1 a obrázku 4.2.

Sada	Počet pravidiel	Počet pseudopravidiel	Po redukcii (10)	Po redukcii (20)	Po redukcii (30)
acl1_100_default_numbered.rul	99	191 700	146 461	117 504	73 964
fw-fast.rul	104	1 322 496	1 046 976	662 592	358 400
fw-hp.rul	173	270 000	234 000	198 000	162 000
fw1_100_default_n.rul	100	432 640	203 112	102 600	65 747
fw1_150_default_n.rul	154	704 480	503 200	304 640	195 181
fw1_200_default_n.rul	188	887 040	526 680	380 086	356 594
fw1_50_default_n.rul	47	42 500	18 000	6 789	3 124
fw2_05_05.rul	956	2 819 460	2 712 864	2 610 104	2 507 344
fw2_05_m05.rul	962	3 206 308	3 101 868	2 997 428	2 892 988
fw2_m05_05.rul	984	4 625 936	4 508 028	4 395 888	4 283 748
fw2_m05_m05.rul	992	4 914 700	4 799 060	4 672 080	4 556 720
fw3_05_05_500.rul	472	4 986 436	4 219 292	4 219 292	3 951 780
fw4_05_05_500.rul	482	13 177 728	11 384 064	9 961 056	9 486 720
fw5_05_05_500.rul	481	8 705 340	7 668 990	7 254 450	7 254 450
rules1.rul	68	168 000	26 928	3 373	1 316
rules4.rul	335	44 153	24 460	12 900	4 649
rules5.rul	1 194	114 826	84 743	83 733	83 252
rules6.rul	1 529	2 584 281	2 321 258	1 830 820	1 830 407

Tabuľka 4.1: Redukcia odhadom počtu pseudopravidiel



Obrázok 4.2: Redukcia odhadom počtu pseudopravidiel — pomer po redukciách

V grafe 4.2 je zobrazená účinnosť redukcie metódy odhadom pre rôzne počty presúvaných pravidiel do asociatívnej pamäti. Pre niektoré sady filtrovacích pravidiel sa dosiahla významná redukcia, pre ostatné však ani po stále väčšom počte odobratých pravidiel nedošlo k výraznejšiemu zníženiu pamäťovej náročnosti (ako je vidno z obrázku, v niektorých prípadoch sa dosiahnutý pomer pohybuje takmer na úrovni pôvodnej veľkosti).

Pri výpočte odhadu asociovaných pseudopravidiel dochádza k tomu, že veľké množstvo pravidiel obsahuje rovnaký odhad. Pred samotným výberom sa pravidlá zoradia podľa hodnoty odhadu a následne sa odoberie požadovaný počet pravidiel. Nezohľadňuje sa však fakt, že určité pravidlá spôsobia významnejší nárast počtu pseudopravidiel ako ostatné. K dispozícii je iba odhad, ale jeho hodnota ešte nemusí znamenať, že konkrétne pravidlo sa na výslednom počte pseudopravidiel podieľa výraznejšou mierou. Navyše, použitie inej radiacej metódy môže viesť k inému poradiu pravidiel (hoci majú rovnaký odhad) a následný výber môže znamenať výrazne odlišné výsledky. Pre dosiahnutie lepších výsledkov by bolo možné po samotnom určení odhadu aplikovať na pravidlá s najväčšou hodnotou odhadu ešte niektorú z nasledujúcich metód, čím by sa dosiahol presnejší výber.

Veľkou výhodou tejto metódy je jej rýchlosť. Táto metóda nepracuje so pseudopravidlami, čo v prípade ostatných metód predstavuje úzke hrdlo v dobe výpočtu. Generovanie pseudopravidiel pre danú sadu pravidiel je výpočtovo náročná úloha. V tomto prípade určenie odhadu trvá iba zanedbateľnú dobu a je otázkou okamihu. Na druhú stranu, aplikácia tejto metódy nemusí vždy viesť k uspokojivej redukcii.

4.3 Optimalizácia založená na priraďovaní počtu pseudopravidiel

Táto metóda je založená na jednorazovom vygenerovaní všetkých pseudopravidiel z danej vstupnej sady pravidiel. Následne sa každému pravidlu priradí počet pseudopravidiel k nemu asociovaných. Po tejto operácii stačí už len vybrať požadovaný počet pravidiel, ku ktorým je asociované najväčšie množstvo pseudopravidiel.

Táto metóda je presnejšia, ale nie je úplne presná. Nemusí eliminovať problémové pravidlá spôsobujúce nárast počtu pseudopravidiel z dôvodu množstva unikátnych prefixov v jednotlivých políčkach, podľa ktorých sa klasifikuje. Zároveň však už generuje pseudopravidlá, čím je náročnejšia na čas, ale toto generovanie je iba jednorázové, takže čas výpočtu nepredstavuje závažný problém.

Pseudokód tejto metódy je nasledovný:

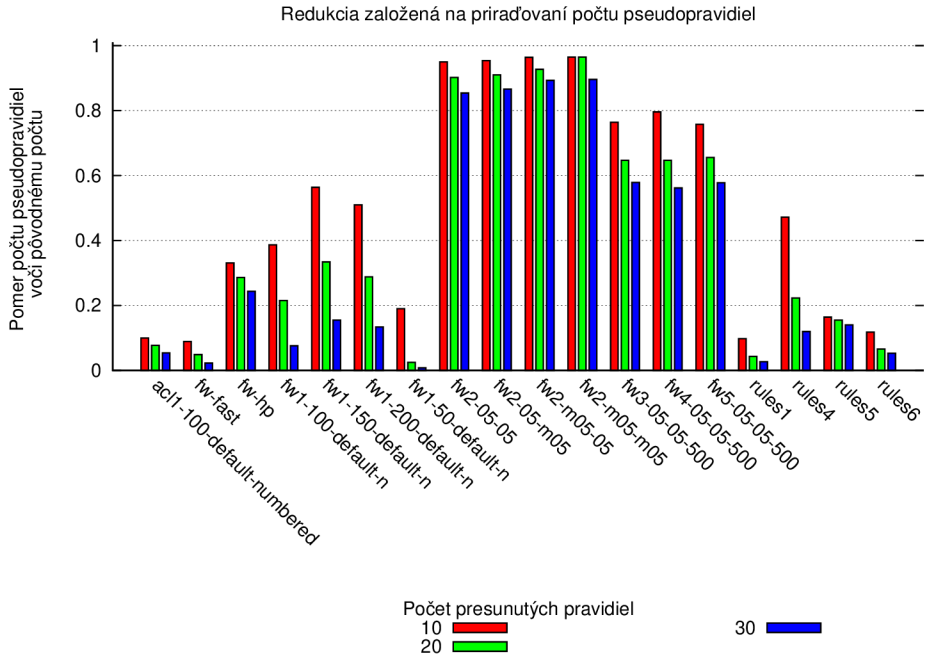
1: Vstup: množina pravidiel R, počet odstraňovaných pravidiel d
2: vytvor množinu pseudopravidiel P z množiny pravidiel R
3: vytvor množinu pravidiel s najväčším počtom pseudopravidiel $R_{max} := \emptyset$
4: **for all** pravidlá r_i z množiny pravidiel R, $i \in \{1 \dots n\}$, $n = |R|$ **do**
5: pravidlu r_i priraď počet asociovaných pseudopravidiel z množiny P
6: **end for**
7: vyber d pravidiel s najväčším počtom asociovaných pseudopravidiel a ulož ich do množiny R_{max}
8: Výstup: množina R_{max} obsahujúca d pravidiel s najväčším počtom priradených pseudopravidiel

Experimenty a zhodnotenie výsledkov

Sada	Počet pravidiel	Počet pseudopravidiel	Po redukcii (10)	Po redukcii (20)	Po redukcii (30)
acl1_100_default_numbered.rul	99	191 700	19 193	14 681	10 349
fw-fast.rul	104	1 322 496	117 437	65 303	30 136
fw-hp.rul	173	270 000	89 424	77 220	65 880
fw1_100_default_n.rul	100	432 640	167 486	92 873	33 193
fw1_150_default_n.rul	154	704 480	397 031	235 509	109 692
fw1_200_default_n.rul	188	887 040	452 732	255 892	119 168
fw1_50_default_n.rul	47	42 500	8 073	1 071	324
fw2_05_05.rul	956	2 819 460	2 679 600	2 542 540	2 408 280
fw2_05_m05.rul	962	3 206 308	3 060 288	2 917 068	2 776 648
fw2_m05_05.rul	984	4 625 936	4 457 376	4 291 616	4 128 656
fw2_m05_m05.rul	992	4 914 700	4 740 960	4 740 960	4 401 880
fw3_05_05_500.rul	472	4 986 436	3 808 800	3 225 060	2 885 580
fw4_05_05_500.rul	482	13 177 728	10 492 000	8 524 320	7 411 824
fw5_05_05_500.rul	481	8 705 340	6 602 310	5 710 716	5 029 376

rules1.rul	68	168 000	16 384	7230	4 486
rules4.rul	335	44 153	20 822	9 830	5 316
rules5.rul	1 194	114 826	18 808	17 836	16 117
rules6.rul	1 529	2 584 281	304 921	171 371	136 053

Tabuľka 4.2: Redukcia priraďovaním počtu pseudopravidiel



Obrázok 4.3: Redukcia priraďovaním počtu pseudopravidiel — pomer po redukciách

Ako znázorňuje graf 4.3 touto metódou sa pre niektoré sady podarilo dosiahnuť lepšie výsledky než v prípade predchádzajúcej metódy založenej na odhade. Naopak, aplikácia tejto metódy na niektoré sady viedla k horším výsledkom. Tento nežiaduci jav však môže byť spôsobený, podobne ako v prípade metódy odhadu, na poradí jednotlivých pravidiel. Z pozorovania výsledkov je možné zistiť, že k mnohým pravidlám sa asociuje rovnaké množstvo pseudopravidiel. Počet takýchto pravidiel však môže prevyšovať počet pravidiel požadovaných k odobratiu, pričom pri výbere sa zase nemusia zasiahnuť pravidlá, ktorých odobratie by viedlo k väčšej redukcii.

Túto metódu je možné v určitých smeroch považovať za presnejšiu, v mnohých prípadoch dosiahla veľmi zaujímavé výsledky. Nevýhodou metódy priraďovania počtu pseudopravidiel je však generovanie pseudopravidiel. Táto operácia je časovo náročná a závisí nielen na počte filtrovacích pravidiel v sade, ale aj ich podobe (sada, v ktorej prevažujú pravidlá s presnými hodnotami v jednotlivých políčkach sa spracuje rýchlejšie ako sada, kde sa v políčkach pre jednotlivé pravidlá vyskytujú prefixové hodnoty pokrývajúce veľký počet

špecifickejších prefixov vzhľadom k ostatným pravidlám).

4.4 Optimalizácia založená na postupnej redukcii

Ďalšou metódou, ktorú je možné použiť, je opakovaná postupná redukcia. Začína vygenerovaním množiny pseudopravidiel. Následne každému pravidlu priradí príslušný počet asociovaných pseudopravidiel. Na rozdiel od predchádzajúcej metódy však vyberie iba jedno pravidlo, ku ktorému je priradené najväčšie množstvo pseudopravidiel. Následne sa celý proces zopakuje, ale s redukovanou sadou vstupných pravidiel. Množina pseudopravidiel sa teda generuje opakovane, ale odoberá sa postupne iba jedno pravidlo. Celý postupný výber končí pri dosiahnutí odobratia požadovaného počtu pravidiel z pôvodne definovanej sady.

Tento prístup však možno považovať v istom zmysle za prístup založený na hrubej sile. Ku generovaniu pseudopravidiel totiž dochádza opakovane. Túto metódu možno považovať za presnejšiu než metódy predchádzajúce. Stavový priestor pseudopravidiel sa prehľadáva dôkladnejšie, nejedná sa však o jeho úplné prehľadanie.

Celý postup je možné názorne popísať nasledovným pseudokódom:

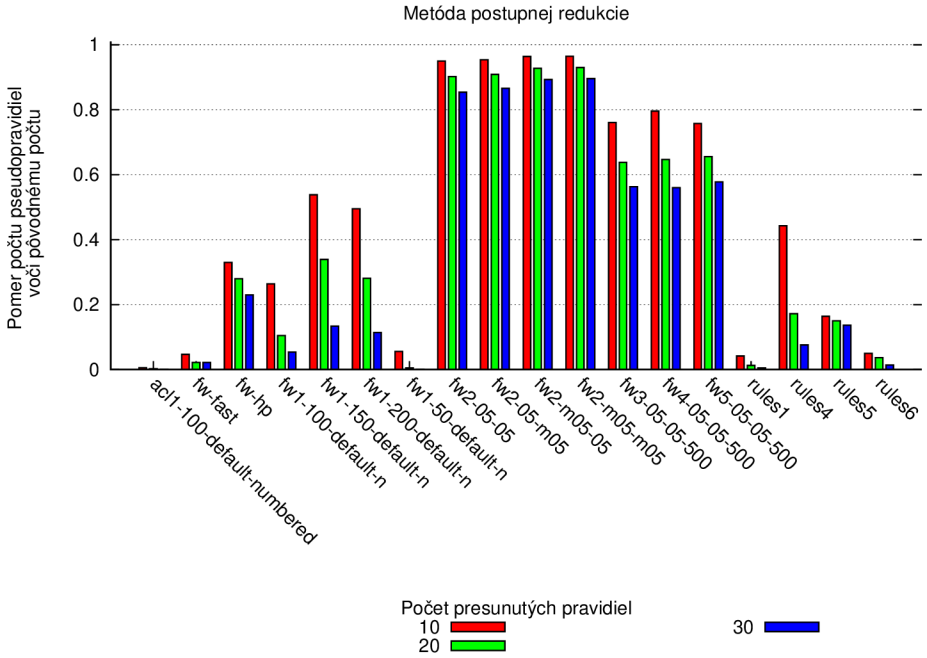
1: Vstup: množina pravidiel R, počet odstraňovaných pravidiel d
2: vytvor množinu pravidiel $R_a := R$
3: vytvor množinu s najväčším počtom pseudopravidiel $R_{max} := \emptyset$
4: **for** $i := 1$ to d **do**
5: **for all** $r_i \in R_a$ **do**
6: metódou priraďovania počtu pseudopravidiel urči počet pseudopravidiel asociovaných k pravidlu r_i
7: **end for**
8: urči pravidlo r_x s najväčším počtom asociovaných pseudopravidiel z R_a
9: $R_{max} := R_{max} \cup \{r_x\}$
10: $R_a := R_a \setminus \{r_x\}$
11: **end for**
12: Výstup: množina R_{max} obsahujúca d pravidiel s najväčším počtom asociovaných pseudopravidiel

Experimenty a zhodnotenie výsledkov

Sada	Počet pravidiel	Počet pseudopravidiel	Po redukcii (10)	Po redukcii (20)	Po redukcii (30)
acl1_100_default_numbered.rul	99	191 700	1 067	556	246
fw-fast.rul	104	1 322 496	61 542	29 301	29 301
fw-hp.rul	173	270 000	89 100	75 600	62 100
fw1_100_default_n.rul	100	432 640	114 261	45 415	23 319
fw1_150_default_n.rul	154	704 480	379 080	238 680	94 074
fw1_200_default_n.rul	188	887 040	438 672	248 976	100 776

fw1_50_default_n.rul	47	42 500	2 394	210	35
fw2_05_05.rul	956	2 819 460	2 679 600	2 542 540	2 408 280
fw2_05_m05.rul	962	3 206 308	3 060 288	2 917 068	2 776 648
fw2_m05_05.rul	984	4 625 936	4 457 376	4 291 616	4 128 656
fw2_m05_m05.rul	992	4 914 700	4 740 960	4 570 020	4 401 880
fw3_05_05_500.rul	472	4 986 436	3 792 600	3 179 520	2 809 404
fw4_05_05_500.rul	482	13 177 728	10 492 000	8 524 320	7 382 928
fw5_05_05_500.rul	481	8 705 340	6 602 310	5 710 716	5 029 376
rules1.rul	68	168 000	7 109	2 108	811
rules4.rul	335	44 153	19 575	7 607	3 323
rules5.rul	1 194	114 826	18 808	17 231	15 676
rules6.rul	1 529	2 584 281	128 508	96 714	35 195

Tabuľka 4.3: Postupná redukcia



Obrázok 4.4: Metóda postupnej redukcie — pomer po redukciách

Oproti predchádzajúcej metóde táto dosahuje lepšie výsledky. Potvrdilo sa, že proces redukcie je presnejší. Dosahovanie lepších výsledkov je však daňou za náročnejší výpočtový proces, nakoľko ku generovaniu pseudopravidiel dochádza opakovane. Za nevýhodu možno považovať, že v niektorých prípadoch nedošlo k výraznejšiemu zníženiu počtu pseudopravidiel oproti metóde priraďovania počtu pseudopravidiel a málo významnejšia redukcia nevyváži dobu potrebnú na ďalšiu optimalizáciu.

Táto metóda patrí k časovo najnáročnejším, vyhodnocovanie pseudopravidiel je výpočtovo náročné predovšetkým pre veľké sady pravidiel (závisí však aj na charaktere pravidiel). V tomto prípade výpočty pre niektoré sady trvajú rádovo až niekoľko dní, čo je pre praktické použitie neakceptovateľné. Lepšie výsledky je však možné použiť pre ďalšie skúmanie vlastností pravidiel a vývoji rovnako kvalitných, ale výpočtovo menej náročných metód.

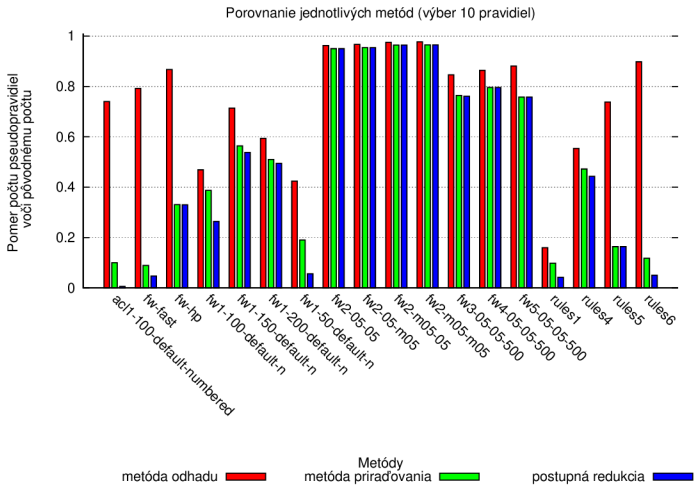
Zhodnotenie predchádzajúcich metód

Predchádzajúce tri metódy možno považovať za základné prístupy k redukcii počtu pseudopravidiel. Prezentované metódy boli zoradené podľa výpočtovej náročnosti potrebnej k získaniu výsledkov od najmenej časovo náročnej. Každá metóda sa odlišuje výsledkami, ktorých kvalita závisí nielen od spôsobu, akým daná metóda pracuje, ale aj poradia výberu pravidiel z pôvodne definovanej sady (v prípade, že obsahuje pravidlá ohodnotené rovnakým počtom asociovaných pseudopravidiel).

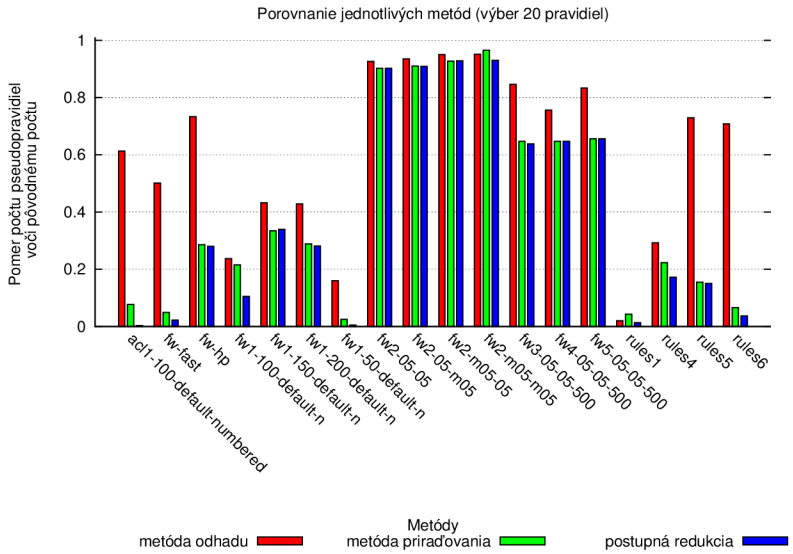
V niektorých prípadoch by bolo možné použiť ďalšiu optimalizáciu založenú na skúmaní, ktoré z rovnako ohodnotených pravidiel spôsobí významnejšiu redukciu v porovnaní s ostatnými, čím sa odstráni závislosť výberu pravidiel na ich poradí. Tento prístup však vyžaduje ďalšie výpočtové kroky, preto je potrebné zvážiť, akým spôsobom sa prípadne vykoná.

Uvedené metódy je však možné kombinovať pre dosiahnutie čo najlepších výsledkov. Keďže požiadavkami na optimalizáciu sú rýchlosť a dosahovaná úroveň, stačí použiť metódy v poradí od najnižšej výpočtovej náročnosti postupne k vyššej. Postačí stanoviť určitý prah a následne aplikovať jednotlivé metódy. V prípade, že použitá metóda nedosiahne požadovanú kvalitu výsledkov, siahne sa v poradí po ďalšej. Navyše, aj v prípade dosiahnutia požadovanej miery redukcie, je možné použiť nasledujúce metódy pre výpočet „na pozadí“. To umožní nasadiť filtrovacie pravidlá okamžite a zároveň priebežne počítať prípadné kvalitnejšie výsledky.

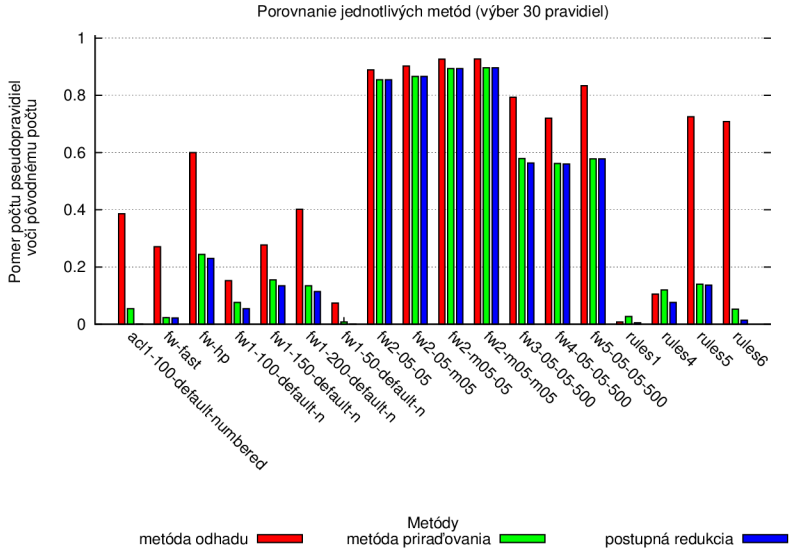
V nasledujúcich grafoch je zobrazená dosahovaná miera redukcie pre jednotlivé metódy.



Obrázok 4.5: Porovnanie metód pre výber 10 pravidiel



Obrázok 4.6: Porovnanie metód pre výber 20 pravidiel



Obrázok 4.7: Porovnanie metód pre výber 30 pravidiel

Dosiahnuté výsledky zodpovedajú predpokladu, že metóda založená na odhade je síce rýchla, ale produkuje najmenej kvalitné výsledky. Kvalita výsledkov (nižší počet pseudopravidiel) rastie s výpočtovou zložitosťou metód a s počtom odoberaných pravidiel.

4.5 Optimalizácia genetickým algoritmom

4.5.1 Úvod do genetických algoritmov

V poslednej dobe sa v počítačovej vede čoraz výraznejšie začína prejavovať záujem o evolučné optimalizačné techniky. Genetické algoritmy patria medzi základné stochastické optimalizačné techniky využívajúce evolučné črty. Tieto sa inšpirujú procesmi v živej prírode pre vývoj nových techník pri riešení veľmi zložitých problémov, kde tradičné postupy zlyhávajú alebo nie sú použiteľné (napr. prehľadávanie celého stavového priestoru možných riešení) a využívajú princípov Darwinovskej evolučnej teórie.

Základnou myšlienkou je, že na jednotlivé prvky množiny prípustných riešení sa pri využití týchto princípov dívame ako na živé organizmy v umelom životnom prostredí. Vývoj týchto „organizmov“ — schopnosť prežitia a reprodukcie závisí na kvalite týchto jedincov — lepší a silnejší jedinci prežívajú a prenášajú svoje vlastnosti do ďalšej generácie.

Spôsob riešenia daného problému potom začína vytvorením počiatočnej populácie organizmov nasledovanom simulovaním ich vývoja. Vývoj je kontrolovaný evolučnými mechanizmami — prirodzenom výbere, reprodukcii, mutácii...

Populácie sa vyvíjajú postupom jednotlivých nových generácií. Pri tomto vývoji sú „menej vhodné“ riešenia nahradzované riešeniami lepšími, pričom „lepšie“ riešenia sa medzi sebou krížia (dochádza ku skombinovaniu genetického materiálu obvykle dvoch kvalitných jedincov) s cieľom získať ešte kvalitnejšie riešenie. Samotné kríženie (reprodukcia jedincov) však ešte nepôsobí dostatočne efektívne pre vznik jedincov, ktorí sú dobre prispôsobení a obsahujú vlastnosti, ktoré by im významne uľahčili prežitie. Preto je nutné používať tzv. mutáciu, ktorá prináša novú informáciu do genetickej informácie jedincov (vo výsledku sa však táto zmena môže prejaviť nielen pozitívne, ale aj negatívne) [8] [6].

Tieto neobvyklé techniky predstavujú pomerne mladú disciplínu v počítačovej vede — vznikli v roku 1975 na Michiganskej univerzite, kde sa John Holland venoval štúdiu bunečných automatov. Napriek tomu však predstavujú pozoruhodný spôsob prístupu k riešeniu mnohých komplikovaných problémov a sú predmetom intenzívneho štúdia, aby boli použiteľné v čo najširšom množstve praktických problémov, nie len v informatike.

4.5.2 Princíp činnosti genetického algoritmu

Genetické algoritmy patria medzi najčastejšie používané evolučné optimalizačné algoritmy využívané pri optimalizácii vysoko modálnych funkcií, kombinatorických a grafovoteoretických problémov až po aplikácie typu „umelý život“.

Biologická evolúcia predstavuje progresívnu zmenu genetickej informácie populácie v priebehu niekoľkých generácií. Veľmi dôležitým pojmom pri využívaní genetických algoritmov je pojem „sila“ jedinca (vhodnosť, angl. „*fitness*“). Tento pojem je prenesený z biologického významu a určuje schopnosť prežitia jedinca v danom prostredí. Presne rovnakým spôsobom sa používa aj v algoritmických úlohách, a to vo forme nejakého kladného reálneho čísla, ktoré je priradené danej genetickej informácii reprezentujúcej organizmus.

Pojem biologický jedinec sa v tejto oblasti nahrádza pojmom chromozóm. Ten predstavuje lineárne usporiadaný informačný obsah jedinca (tzv. genotyp). Jednotlivé chromozómy potom tvoria populáciu, pričom medzi nimi s istou pravdepodobnosťou zodpovedajúcou ich vhodnosti dochádza ku kríženiu (reprodukcii). Chromozómy podliehajú mutácii, pomocou ktorej je možné do chromozómov vložiť novú informáciu. Kvalitnejšie chromozómy vytesnia slabšie z danej populácie (s menšou vhodnosťou)[6]. Celý proces sa neustále opakuje, pričom môže byť nekonečný. Pre praktické účely sa však volí určitý

mechanizmus ukončenia činnosti genetického algoritmu.

Medzi základné pojmy genetických algoritmov patria [11][12]:

- gén: základná stavebná jednotka chromozómu, nositeľ informácie
- alela: konečná množinu hodnôt nad definovanou abecedou (množina prirodzených čísel, binárne symboly, ...), ktorú môžu gény nadobúdať
- chromozóm: zložený z génov, predstavuje jedinca a obsahuje zakódované parametre a jedno riešenie problému v danom stavovom priestore, má pevný počet génov
- populácia: je tvorená konečným počtom chromozómov, počiatočná populácia sa generuje (pseudo)náhodne
- genotyp: je zakódovaný tvar riešenia
- fenotyp: je dekódovaný tvar riešenia
- účelová funkcia: reprezentuje stavový priestor, v ktorom prebieha hľadanie minima, bod v tomto priestore sa reprezentuje pomocou chromozómu
- fitness funkcia: je invertovaná účelová funkcia (prevedie hľadanie minima na hľadanie maxima)
- schéma σ: je šablóna reprezentujúca množinu chromozómov, je zložená z pevných a voľných symbolov
- stavebné bloky: sú dôležité gény chromozómu, snahou je ich minimálne rozbíjanie

Pri voľbe použitia genetického algoritmu pre riešenie určitého problému je nutné sa zaoberať postupom jeho návrhu.

Reprezentácia problému

Zahŕňa spôsob, akým bude zakódovaná informácia pre jedno riešenie v chromozóme. Používa sa niekoľko spôsobov zakódovania: binárny, permutačný (celočíselný), stromová štruktúra.

Pri binárnom spôsobe zakódovania je možné použiť klasické binárne zakódovanie alebo kódovanie pomocou Grayovho kódu. Binárny spôsob je najčastejší spôsob kódovania a nevyžaduje špeciálne genetické operátory.

Permutačné zakódovanie však už vyžaduje špeciálne genetické operátory, aby nedochádzalo k vzniku neplatných riešení.

Pre stromové štruktúry, podobne ako v predchádzajúcom prípade, je nutné použiť špeciálne genetické operátory.

Vytvorenie počiatočnej populácie

Na začiatku sa náhodne vytvorí počiatočná populácia, ktorá bude obsahovať určený počet jedincov. V prípade binárneho zakódovania sa jedincom náhodne priradia hodnoty 1 a 0 do jednotlivých génov, v prípade permutačného zakódovania sa priradia hodnoty z povolenej množiny hodnôt tak, aby jedinec (chromozóm) predstavoval nejaké platné riešenie a pod.

Spôsob ohodnotenia jedincov v populácii

Každého jedinca v populácii je nutné ohodnotiť a určiť tak jeho vhodnosť. Predstavuje časovo najnáročnejšiu časť celého algoritmu (z dôvodu obyčajne vysokej výpočtovej náročnosti). Lepšia hodnota *fitness* znamená lepšie vlastnosti jedinca. Obyčajne zakódovanie informácie jednoduchým spôsobom spôsobuje zložitejší výpočet ohodnotenia. Tento princíp však obvykle platí i naopak.

Výber jedincov rodičovskej populácie

Výber jedincov z rodičovskej populácie je dôležitý pre operácie kríženia a mutácie. Obvykle sa vyberá pár jedincov, ale môže sa týkať aj väčšieho počtu. V procese výberu sa uprednostňujú kvalitnejšie jedince podľa ich ohodnotenia.

Používa sa niekoľko druhov výberu:

- ruletový výber: kvalitnejší jedinci majú väčšiu pravdepodobnosť výberu
- turnajový výber: náhodne sa vyberie skupina (zvyčajne pár) jedincov a následne najkvalitnejší z nich
- pravdepodobnostný výber: môže byť realizovaný najprv ruletovým výberom a následne turnajom
- iné možnosti...

Operátor kríženia

Operátory kríženia (rekombinácie) sú najdôležitejšou časťou genetických algoritmov. Do operácie kríženia je nutné nejakým spôsobom vybrať vhodných jedincov. Takýto výber môže byť náhodný, alebo založený na klonovaní jedincov, či na podobnosti jedincov (genotypickej či fenotypickej).

Operátory kríženia sú odlišné pre každý spôsob reprezentácie chromozómu. Operátory pre binárne zakódovanie sú jednoduchšie (kríženie je priamočiare). Kríženie binárne zakódovaných chromozómov môže byť:

- jednobodové: bod kríženia v chromozóme sa vyberá náhodne, vznikajú dva nové chromozómy
- dvojbodové: body kríženia sa vyberajú náhodne a tiež vznikajú dva nové chromozómy
- viacbodové: zriedkavé použitie
- uniformné: používa náhodne vygenerovanú masku určujúcu, z ktorého rodiča sa vyberie príslušná hodnota génu na danej pozícii, vzniká jeden nový chromozóm

Podobne pre permutačné zakódovanie:

- Order 1 Crossover (označovaný OX): vzniká jeden nový chromozóm z dvoch rodičov
- Partially Mapped Crossover (označovaný PMX): opäť jeden nový chromozóm z dvoch rodičov

- Cycle Crossover (označovaný CX): z dvoch rodičov vzniknú dva nové jedince

Pre stromové kódovanie riešenia môže byť operátor kríženia realizovaný napr. výmenou podstromov medzi jedincami vo vybraných uzloch.

Operátor mutácie

Operátor mutácie sa používa obvykle v omnoho menšej miere v porovnaní s operátorom kríženia (typicky s pravdepodobnosťou 0.01 až 0.001). Existujú však prípady, kedy je vyššia pravdepodobnosť mutácie žiadaná. Opäť, pre rôzne spôsoby reprezentácie informácie v chromozóme sa používajú rozdielne operátory mutácie. Pre binárnu reprezentáciu je to jednoduchý mechanizmus inverzie hodnoty náhodne vybraného bitu v chromozóme. Pre permutačné zakódovanie to môže byť výmena hodnôt dvoch náhodne vybraných pozícií v chromozóme.

Spôsob obnovy populácie

Existujú dva základné spôsoby obnovy populácie: generatívna obnova, ktorá využíva úplnú obnovu rodičovskej populácie a potom čiastočná obnova (*steady-state* genetické algoritmy), pri ktorej sa rodičovská populácia nahradzuje iba v určitej miere. Čiastočná obnova populácie môže byť vo forme tzv. elitizmu, pri ktorom sa vyberie určitý počet najlepších jedincov a tí postúpia do novej generácie priamo. Alebo vo forme turnajového výberu, podľa kvality jedincov či faktoru premnoženia. Pri použití faktoru premnoženia sa náhodne vyberie podmnožina rodičov, pričom nový jedinec nahradzuje rodičov s podobným genotypom.

Spôsob ukončenia algoritmu

Genetický algoritmus je možné ukončiť špecifikovaním počtu generácií, detekciou malej diverzity populácie (medzi jedincami sú iba malé rozdiely) alebo nevyhovujúcim rastom fitness hodnoty. Genetický algoritmus je nutné spustiť opakovane, jedno spustenie nemá o dosiahnutých výsledkoch veľkú výpovednú hodnotu. Jednak sa nemusí získať vyhovujúce riešenie a jednak je nutné riešiť resp. objaviť vhodné nastavenie parametrov genetického algoritmu pre riešenie daného problému. Navyše, genetický algoritmus je stochastický.

Výhody a nevýhody genetických algoritmov

Genetické algoritmy predstavujú nástroj pre riešenie zložitých problémov, kde tradičné postupy zlyhávajú. Veľkou výhodou je, že nevyžadujú znalosť prehľadávaného priestoru či cieľovej funkcii, ktorá podlieha optimalizácii (často však doplnenie určitej znalosti môže viesť k nájdeniu kvalitnejšieho riešenia). Vďaka genetickým operátorom sú odolné voči skĺznutiu do lokálneho optima.

Nevýhodou je, že majú problém s nájdením globálneho optima — vždy však vedú aspoň k suboptimálnemu riešeniu. Vyhodnocovanie vhodnosti jedincov, ktoré sa deje opakovane a vo veľkom počte môže predstavovať úzke miesto algoritmu, kde výpočtová náročnosť môže brániť použitiu genetických algoritmov. Navyše, spôsob voľby či návrh jednotlivých častí genetického algoritmu nemusí byť vždy jednoduchý a vyžaduje určitú zručnosť a skúsenosti.

4.5.3 Redukcia počtu pseudopravidiel použitím genetického algoritmu

Redukcia pseudopravidiel pomocou genetického algoritmu bola vyskúšaná ako alternatívna metóda hľadania vhodných kombinácií pravidiel pre výber do asociatívnej pamäti. Implementácia bola založená na využití voľne dostupnej knižnice GALib [18], čím sa proces návrhu programu podstatne zjednodušil.

Cieľom bolo navrhnúť čo najjednoduchšiu aplikáciu využívajúcu optimalizáciu genetickým algoritmom. Úlohou genetického algoritmu v tomto prípade je nájsť takú kombináciu výberu pravidiel, ktorá by čo najvýraznejšie minimalizovala výsledný počet pseudopravidiel z redukovanej sady.

Reprezentácia problému je realizovaná celočíselne vo forme indexov do zoznamu pravidiel. Je síce možná aj binárna reprezentácia, pri ktorej by chromozóm obsahoval taký počet génov, koľko je v sade pravidiel. V tomto prípade pre každé pravidlo, ktoré by bolo súčasťou výberu, by sa nastavila hodnota bitu na 1. Počet jednotkových bitov je daný počtom pravidiel k výberu. Takáto reprezentácia je však pre väčšie sady pamäťovo zbytočne náročná.

Preto sa použila reprezentácia celočíselná s počtom génov daných počtom pravidiel k výberu. S ohľadom na reprezentáciu problému je potrebné zvoliť špeciálny operátor kríženia, aby nedochádzalo k vzniku neplatných riešení. Použil sa operátor Order 1 Crossover, ale je možné použiť prakticky ľubovoľný operátor pre celočíselné reprezentácie problémov. Navyše, reprezentácia nezávisí na poradí hodnôt v chromozóme, dôležitá je iba prítomnosť konkrétnych hodnôt.

Operátor mutácie je potrebné riešiť takisto špeciálne. Ten pracuje tak, že náhodne zvolí určitý gén v chrozóme a nahradí ho náhodným indexom z množiny pravidiel. Tu však opäť môže nastať problém, že sa takto do chromozómu môžu dostať dva zhodné indexy, takže dochádza ku kontrole nahradzovaného indexu a v prípade potreby sa vyberie index znova.

Vyhodnotenie vhodnosti chromozómu je realizované jednoduchým výpočtom počtu výsledných pseudopravidiel zo sady, z ktorej sa odoberú pravidlá dané indexami uloženými v chromozóme. Kvalitnejší chromozóm je taký, ktorý redukuje väčší počet pseudopravidiel.

Bol použitý *steady-state* genetický algoritmus. Pri spustení programu je možné voliť pravdepodobnosti kríženia a mutácie, veľkosť chromozómu (odpovedá počtu odoberaných pseudopravidiel), počet jedincov v jednej populácii a počet generácií.

Experimentálne výsledky

Optimalizácia genetickým algoritmom vyžaduje opakované spustenie procesu vývoja jedincov a vykonanie množstva pokusov s rôzne zvolenými parametrami. Navyše, najväčším problémom je doba výpočtu vhodnosti jedincov, ktorá môže pre veľké sady dosiahnuť neakceptovateľné hodnoty. Z toho dôvodu bol vykonaný iba obmedzený počet experimentov.

Výsledky však naznačujú, že táto metóda dokáže pri vhodne zvolených parametroch dostatočne dobre konkurovať metóde postupnej redukcie. Navyše, z opakovaných pokusov vyplynulo, že pre dosiahnutie dobrých výsledkov je potrebné výrazne zvýšiť pravdepodobnosť mutácie jedinca, čím sa táto metóda odlišuje od tradičných metód, kde sa operátor mutácie používa zriedka.

Tabuľka 4.4 obsahuje ukážky výsledkov redukcie pre niektoré vybrané sady pravidiel. Tabuľka obsahuje výsledky z genetického algoritmu a najlepšie porovnateľné výsledky, ktoré boli získané z metódy postupnej redukcie.

Pre všetky prípady sa ako vhodné nastavenie parametrov javila pravdepodobnosť krí-

	Redukcia gen. alg.		Metóda postupnej redukcie	
Sada	Po redukcii (10)	Po redukcii (20)	Po redukcii (10)	Po redukcii (20)
acl1_100_default-_numbered.rul	1 067	554	1 067	556
fw1_50_default_-n.rul	2 340	92	2 394	210
rules1.rul	7 109	2 108	7 109	2 108
rules4.rul	19 316	6 774	19 575	7 607
rules5.rul	22 117	17 290	18 808	17 231
rules6.rul	198 497	96 990	128 508	96 714

Tabuľka 4.4: Porovnanie výsledkov: počet pseudopravidiel pre genetický algoritmus a metódu postupnej redukcie

ženia na úrovni 0,7 a pravdepodobnosť mutácie 0,45. [1] Je to z toho dôvodu, aby sa zaistil výber aj tých pravidiel, ktoré nemuseli byť súčasťou pôvodných chromozómov.

Z dosiahnutých výsledkov možno konštatovať, že genetický algoritmus dokáže vyprodukovať veľmi kvalitné výsledky. Pre experimenty boli uprednostňované sady s menším počtom pravidiel, aby bolo možné výsledky získať v prijateľnom čase. Genetický algoritmus však bol aplikovaný aj na sady s väčším počtom pravidiel (*rules5.rul, rules6.rul*) obsahujúcim po 1 194 a 1 529 pravidiel. Charakter pravidiel však umožňoval rýchle vyhodnocovanie vhodnosti jedincov. Nakoniec aj pre takto veľké sady je možné genetickým algoritmom dosiahnuť kvalitné výsledky.

Pri použití genetického algoritmu je potrebné navyše skúmať vplyv parametrov na dosahované výsledky a hľadať ich čo najvýhodnejšie hodnoty. Pre redukciu pseudopravidiel však neexistuje jedno univerzálne nastavenie parametrov. Každá sada je totiž odlišná a predstavuje samostatný problém. Hľadanie optimálnych parametrov takto môže dokonca viesť na optimalizáciu ich samotných iným genetickým algoritmom. Problémom však ostáva veľká časová náročnosť nielen takéhoto riešenia, ale aj optimalizácia sád, ktoré obsahujú niekoľko stoviek pravidiel.

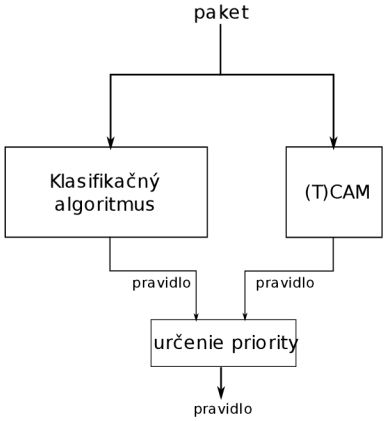
4.6 Použitie asociatívnych pamätí pre klasifikáciu

Asociatívne pamäte (*Content Addressable Memories — CAM*) slúžia k rýchlemu vyhľadaniu umiestnenia určitých dát. Vstupná hodnota sa porovnáva s obsahom pamäte na jednotlivých riadkoch. Porovnávanie prebieha paralelne voči všetkým uloženým hodnotám, takže vyhľadanie môže prebehnúť počas 1 taktu. Výstupom asociatívnej pamäti je adresa nájdených dát a signál zhody vstupných dát voči obsahu pamäti. Pri použití CAM pamätí pre určitú aplikáciu je nutné vziať do úvahy niekoľko faktorov: šírku dátového slova, ich počet, čas porovnania, vplyv rýchlosti zápisu dát pre konkrétnu aplikáciu, hodinový kmitočet, masku pre dáta a výstupy [2]. Je to nutné preto, že neexistuje jednotný predpis pre vytváranie a používanie CAM pamätí univerzálny pre všetky aplikácie.

[1] Okrem sady *rules4*, kde sa použila populácia o 250 jedincoch, sa pre ostatné sady vytvorila populácia o 400 jedincoch. Pre sada *rules1* sa použilo 60 generácií, pre sady *rules5* a *rules6* 200 generácií a pre všetky ostatné 150 generácií. Uvedené parametre sa týkajú výberu 10 a 20 pravidiel.

V predchádzajúcom texte boli navrhnuté možnosti redukcie pamäťovej náročnosti spôsobenej množstvom pseudopravidiel výberom určitých pravidiel do asociatívnej pamäte. Určitý malý počet problémových pravidiel sa takto spracuje veľmi efektívnym spôsobom. Použitie asociatívnej pamäti je flexibilné — je totiž nezávislé na zvolenom klasifikačnom algoritme.

Klasifikácia paketov prijímaných zo sieťových rozhraní sa potom uskutočňuje jednak klasifikačným algoritmom a zároveň použitím asociatívnej pamäti tak, ako to naznačuje obrázok 4.8



Obrázok 4.8: Klasifikácia realizovaná ľub. klasifikačným algoritmom a asociatívnou pamäťou

Na konci postačí zvoliť správne pravidlo na základe jeho priority. Je možné zvoliť buď ternárnu asociatívnu pamäť alebo pamäť s podporou rozsahov. Asociatívne pamäte možno použiť aj vo forme samostatného čipu, to však samotné riešenie môže predražiť a zároveň znížiť požadovaný výkon. Cieľom je teda integrovať na jeden čip (FPGA) celý systém klasifikácie vrátane pamätí. Ďalší text upriamuje pozornosť na možnosť implementácie asociatívnej pamäti v čipe FPGA. Takúto pamäť je potom možné popísať napr. v jazyku VHDL.

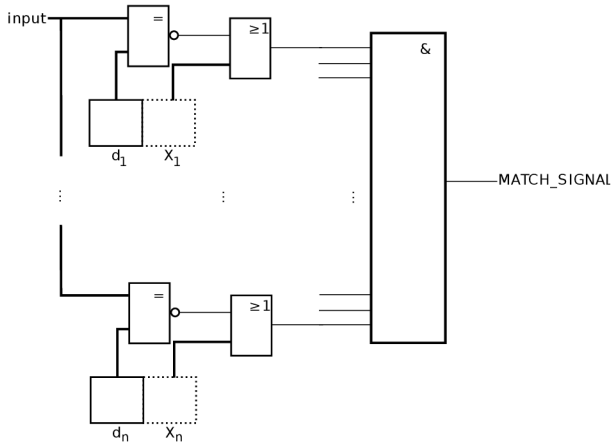
4.6.1 Ternárna asociatívna pamäť

Ternárna asociatívna pamäť (TCAM) priamo podporuje vyhľadávanie prefixov. Dáta sa do tohto typu pamäti ukladajú vo forme dvojice *(dáta, maska)*. Pomocou masky sa definujú bitové pozície, ktoré budú následne voči vstupným dátam vyhodnocované ako *don't care* (zodpovedá ďalšiemu stavu oproti pôvodným dvom — log. 0 a 1).

Pamäť TCAM môže mať podľa obr. 4.9.

Na obrázku políčko d reprezentuje dátové slovo určitej šírky, ktorým je táto pozícia inicializovaná a políčko x hodnotu masky.

Požadovaný počet takýchto buniek potom tvorí jeden riadok TCAM pamäte. Jeden riadok takto môže pokrývať všetky políčka paketu, podľa ktorých sa klasifikuje. Z každej takejto bunky je potom výstup logického členu OR privedený na jedno n-vstupové hradlo AND, ktoré produkuje hodnotu výstupného signálu zhody voči vstupným dátam. Počet takto zostavených riadkov potom reprezentuje počet položiek, ktoré je možné do pamäte uložiť. Platný signál zhody na výstupe hradla AND potom znamená, že vstupné dáta vyhoveli uloženým dátam spolu s definovanou maskou.

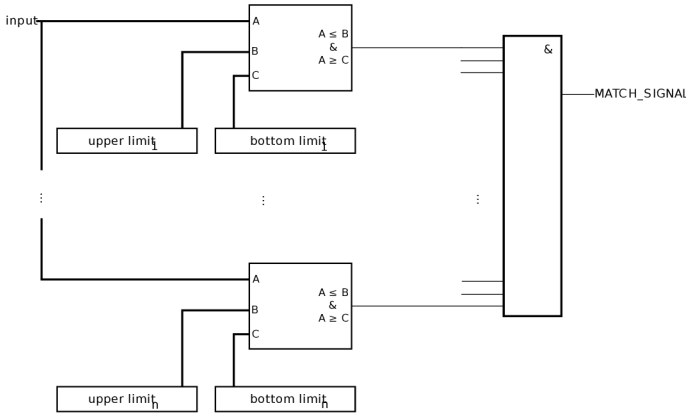


Obrázok 4.9: Vyhodnotenie zhody pre jeden riadok v TCAM pamäti

4.6.2 Asociatívna pamäť s podporou rozsahov

Pre klasifikáciu paketov je možné použiť aj asociatívnu pamäť, ktorá podporuje hľadanie hodnôt v rámci uložených intervalov. V tomto prípade sa namiesto vyhľadávania prefixov používa vyhľadávanie rozsahov. Oba spôsoby reprezentácie hodnôt je možné medzi sebou prevádzať.

Pri tomto spôsobe vyhľadávania sa ukladá horná a dolná hranica požadovaného intervalu a namiesto detekcie zhody voči vstupným dátam klasickým komparátorom sa použije komparátor aritmetický. Schematicky je takéto porovnanie naznačené na obr. 4.10.



Obrázok 4.10: Vyhľadávanie pomocou CAM použitím aritmetických komparátorov

Pre každé požadované políčko sa použije zobrazená štruktúra a výstupy z jednotlivých komparátorov sú privedené na vstup n-vstupového hradla AND, ktorého výstupný signál rozhoduje o tom, či sa hodnoty políčok daného paketu zhodujú voči uloženým hodnotám intervalov.

4.6.3 Implementácia CAM pamätí v Xilinx FPGA čipoch

V FPGA čipoch firmy Xilinx je možné využiť dostupné nástroje pre automatické vygenerovanie zdrojového kódu CAM pamäte podľa požadovaných vlastností. Podporované sú počty položiek CAM od 16 do 4096, šírky dát od 1 do 512 bitov, spôsob implementácie využitím SRL16 primitív či BlockRAM pamätí na čipe FPGA, podpora pre ternárne operácie a iné [20].

CAM pamäte vytvorené pomocou BlockRAM pamätí vyžadujú dva takty pre zápis dát a jeden takt pre vyhľadanie.

Ternárny režim je podporovaný iba v prípade využitia SRL16 primíitív (predstavujú 16-bitový posuvný register). Takto vytvorené pamäte však vyžadujú 16 taktov pre zápis dát do pamäte. Vyhľadanie ale prebieha počas 1 taktu. Doba zápisu v prípade klasifikácie paketov nepredstavuje problém. Táto operácia sa uskutočňuje iba v prípade počiatočnej konfigurácie či následnej zmene sady pravidiel. Podporované sú dva ternárne režimy:

- štandardný, ktorý používa tri stavy (0, 1, X), kde X je vo význame *don't care bit*
- rozšírený, ktorý používa štyri stavy (0, 1, X, U), kde U je vo význame *unmatchable bit*

Je nutné si uvedomiť, že väčšia šírka dátového slova a väčší počet slov uložiteľných do pamäte má pomerne výrazný vplyv na spotrebu zdrojov na čipe a výslednú maximálnu pracovnú frekvenciu. Tieto pamäte je možné si vygenerovať pomocou generátoru, ktorý umožňuje nastaviť množstvo požadovaných parametrov s ohľadom na cieľovú aplikáciu. Po vygenerovaní postačí vygenerovanú komponentu doplniť do výsledného riešenia. Takto sa použitie asociatívnych pamätí v FPGA zjednodušuje a odpadá potreba kompletného návrhu a testovania samostatne vytvorenej jednotky.

Kapitola 5

Záver

Táto práca predstavuje problém klasifikácie paketov a boli v nej priblížené rôzne prístupy spolu s príkladmi algoritmov publikovanými v poslednej dobe v odbornej literatúre. Popisuje zároveň aj ich vlastnosti s ohľadom na možnú obvodovú implementáciu. Podrobnejšie sa zameriava na moderné algoritmy klasifikácie paketov kartézskeho súčinu polí, ktoré patria do skupiny algoritmov založených na dekompozícii problému.

Tieto algoritmy sa vyznačujú svojou rýchlosťou, ale ich nevýhoda spočíva v pomerne veľkej pamäťovej náročnosti. Tá vyplýva z množstva vznikajúcich pseudopravidiel voči pôvodne definovaným pravidlám vo vstupnej sade. Určité techniky na obmedzenie počtu pseudopravidiel boli predstavené v sekcii 3.4.4 popisujúcej algoritmus MSCA. Algoritmus PHCA popísaný v sekcii 3.4.6 je pamäťovo úspornejší, keďže neukladá všetky možné kombinácie vyhľadania najdlhšieho zhodného prefixu. Problém s pamäťovou náročnosťou však neodstraňuje úplne.

Jednou z metód na redukciu počtu pseudopravidiel je umiestnenie problémových pravidiel do asociatívnej pamäti. Identifikácia a výber pravidiel však nebol ešte dostatočne preskúmaný. Táto práca navrhuje 3 metódy výberu založené prehľadávaní stavového priestoru. Jednou z možností je prostý odhad počtu pseudopravidiel priradený jednotlivým pravidlám, ďalšou určenie počtu priradených pseudopravidiel rozgenerovaním pôvodnej sady a poslednou metódou je postupná a opakovaná redukcia. Pre každú z týchto metód prezentuje experimentálne výsledky a ich vzájomné porovnanie. Experimentálne bol implementovaný genetický algoritmus a aplikovaný na sady s menším počtom pravidiel, pričom sa ním podarilo dosiahnuť pomerne kvalitné výsledky.

Dosiahnuté výsledky sa zhodujú s očakávaním, že jednoduchšie a rýchlejšie metódy dosahujú horšie výsledky a naopak. Nevýhodou metód založených na prehľadávaní stavového priestoru je ich časová zložitosť. Vyhodnotenie vstupnej sady totiž totiž trvá dlhšie nielen s rastúcim počtom pravidiel, ale aj spôsobe, akým sú definované (pravidlá obsahujúce v niektorých políčkach hodnoty, ktoré pokrývajú množstvo iných hodnôt totiž spôsobujú nárast počtu pseudopravidiel a tento výpočet sa musí zohľadniť aj voči ostatným pravidlám). Získavanie výsledkov pre zložitejšie metódy a pre niektoré sady je časovo veľmi náročné — ako príklad možno uviesť metódu postupnej redukcie pre sadu syntetických pravidiel s 984 pravidlami, kde optimalizácia výberom 20 pravidiel zabrala približne 106 hodín (CPU Intel Xeon E5420 2.50GHz, 4GiB RAM). Program na vyhodnotenie výsledkov optimalizácie je implementovaný z dôvodu rýchlosti v jazyku C, rýchlosť výpočtu však aj napriek tomu predstavuje problém.

Pokračovanie práce môže byť založené na alternatívnych metódach prehľadávania stavového priestoru, napr. s využitím iných evolučných techník, či hľadania ďalších optimali-

začných metód, ktoré sa netýkajú iba výberu pravidiel do asociatívnej pamäti. Je potrebné naďalej skúmať charakter problémových pravidiel a hľadať ďalšie spôsoby umožňujúce redukciu pamäťovej náročnosti. Vývoj takýchto metód je potrebný predovšetkým pre rýchle spracovanie vstupnej sady, pretože môže byť významná z hľadiska bezpečnosti, kedy je nutné nasadiť novú sadu pravidiel okamžite (napr. automatickým systémom v prípade detekcie útoku).

Nové metódy by nakoniec umožnili lepšie využitie tejto skupiny algoritmov aj v praktickom použití. Redukcia pamäťovej náročnosti umožní odstrániť nutnosť použitia externých pamätí a integrovať celé riešenie do jedného čipu, čo umožní znížiť jeho cenu či energetickú náročnosť.

Literatúra

[1] Ahmadi, M.; Wong, S.: Modified collision packet classification using counting Bloom filter in tuple space. In *PDCN'07: Proceedings of the 25th conference on Proceedings of the 25th IASTED International Multi-Conference*, Anaheim, CA, USA: ACTA Press, 2007, s. 315–320.

[2] Brelet, J.-L.: *An Overview of Multiple CAM Designs in Virtex Family Devices.* Xilinx Inc., 1999, Application Note 201, version 1.1.

[3] Dharmapurikar, S.; Song, H.; Turner, J.; aj.: Fast packet classification using bloom filters. In *ANCS '06: Proceedings of the 2006 ACM/IEEE symposium on Architecture for networking and communications systems*, New York, NY, USA: ACM, 2006, ISBN 1-59593-580-0, s. 61–70.

[4] Gupta, P.: *Algorithms for routing lookups and packet classification.* Stanford University, 2001, Ph.D. thesis.

[5] Jiang, W.; Prasanna, V. K.: Parallel IP lookup using multiple SRAM-based pipelines. *Parallel and Distributed Processing, 2008. IPDPS 2008*, 2008: s. 1–14, ISSN 1530-2075.

[6] Kvasnička, V.; Pospíchal, J.; Tiňo, P.: *Evolučné algoritmy.* Vydavateľstvo STU Bratislava, 2000, ISBN 80-227-1377-5.

[7] Lakshman, T. V.; Stiliadis, D.: High-speed policy-based packet forwarding using efficient multi-dimensional range matching. *SIGCOMM Comput. Commun. Rev.*, ročník 28, č. 4, 1998: s. 203–214, ISSN 0146-4833.

[8] Luner, P.: Jemný úvod do genetických algoritmů. `http://cgg.mff.cuni.cz/~pepca/prg022/luner.html`.

[9] Puš, V.; Kořenek, J.: Fast and scalable packet classification using perfect hash functions. In *FPGA '09: Proceeding of the ACM/SIGDA international symposium on Field programmable gate arrays*, New York, NY, USA: ACM, 2009, ISBN 978-1-60558-410-2, s. 229–236.

[10] Puš, V.: *Klasifikace paketů s využitím technologie FPGA.* FIT VUT v Brně, 2008, diplomová práce.

[11] Satola, R.: *Genetické algoritmy.* FIT VUT v Brně, 2004, přednášky k předmětu Aplikované evoluční algoritmy.

[12] Schwarz, J.; Sekanina, L.: *Aplikované evoluční algoritmy, studijní opora.* FIT VUT v Brně, 2006.

[13] Singh, S.; Baboescu, F.; Varghese, G.; aj.: Packet classification using multidimensional cutting. In *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, New York, NY, USA: ACM, 2003, ISBN 1-58113-735-4, s. 213–224.

[14] Song, H.; Turner, J.; Lockwood, J.: Shape Shifting Tries for Faster IP Route Lookup. In *ICNP '05: Proceedings of the 13TH IEEE International Conference on Network Protocols*, Washington, DC, USA: IEEE Computer Society, 2005, ISBN 0-7695-2437-0, s. 358–367.

[15] Srinivasan, V.; Varghese, G.; Suri, S.; aj.: Fast and scalable layer four switching. *SIGCOMM Comput. Commun. Rev.*, ročník 28, č. 4, 1998: s. 191–202, ISSN 0146-4833.

[16] Taylor, D. E.; Turner, J. S.: Scalable packet classification using distributed crossproducing of field labels. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, IEEE, 2005, ISBN 0-7803-8968-9, s. 269–280.

[17] Taylor, D. E.; Turner, J. S.: ClassBench: a packet classification benchmark. *IEEE/ACM Trans. Netw.*, ročník 15, č. 3, 2007: s. 499–511, ISSN 1063-6692.

[18] Wall, M.: GAlib: A C++ Library of Genetic Algorithm Components. `http://lancet.mit.edu/ga/`.

[19] Wang, P.-C.; Lee, C.-L.; Chan, C.-T.; aj.: Hardware-based Packet Classification Made Fast and Efficient. In *ICPADS '05: Proceedings of the 11th International Conference on Parallel and Distributed Systems - Workshops (ICPADS'05)*, Washington, DC, USA: IEEE Computer Society, 2005, ISBN 0-7695-2281-5, s. 47–51.

[20] Xilinx Inc.: *Content-Addressable Memory v6.1*. Xilinx Inc., 2008, Product Specification, DataSheet DS253.