Tato práce vznikla za podpory projektu TeamIt, jenž je spolufinancován Evropským sociálním fondem a státním rozpočtem České republiky prostřednictvím grantu ESF CZ.1.07/2.3.00/09.0067.

Budování konkurenceschopných výzkumných týmů pro IT

Výzkumná skupina ANT at FIT:

Hardwarová akcelerace klasifikace paketů

Technická zpráva

10.května 2010

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ VUT V BRNĚ
Božetěchova 2, 612 66 Brno
tel.: +420 541 141 144, +420 541 212 219, fax: +420 541 141 170, 270
http://www.fit.vutbr.cz, http://teamit.fit.vutbr.cz

1 Úvod

Klasifikace paketů je úloha, na kterou lze pohlížet jako na geometrický, anebo kombinatorický matematický problém. Zároveň však praktické řešení této úlohy má velký dopad v mnoha aplikacích počítačových sítí.

V současnosti existuje mnoho přístupů ke klasifikaci paketů, avšak každý z nich má některé nevýhody. Některá řešení jsou vhodná pouze pro softwarovou implementaci, novější práce však již předpokládají využití specializovaných technických prostředků pro dosažení vysokého výkonu. Zatímco použití software nebo hardware je spíše implementační otázkou, mnohem důležitejší je analýza algoritmů z teoretického pohledu. Při práci s klasifikačními algoritmy je důležitá především časová a prostorová složitost. Z důvodů popsaných dále dělíme časovou složitost popsaných algoritmů na složitost vyhledání, a složitost vytvoření datových struktur.

2 Teoretický rozbor

2.1 Model klasifikace

Klasifikační algoritmus má k dispozici množinu pravidel uspořádanou podle priority. Vstupem algoritmu jsou hodnoty z hlavičky paketu. Klasifikační algoritmus musí provést hledání, jehož výsledkem je pravidlo, které odpovídá danému paketu. Pokud paketu odpovídá více pravidel, potom je z nich vybráno pravidlo s nejvyšší prioritou. Výstupem klasifikace je číslo výsledného pravidla.

2.2 Klasifikační pravidla

Pravidlo pohlíží na několik polí z hlavičky paketu a pro každé pole určuje podmínku. Podmínka může být:

- **Rozsah**: Je určen souvislý rozsah hodnot, které vyhovuje pravidlu. Takové podmínky se používají například pro čísla TCP/UDP portů.
- **Prefix**: Je zadáno pouze několik vyšších bitů datového slova, nižší bity mohou mít libovolnou hodnotu. Takové podmínky se často využívají při definicích skupiny IP adres. Potom prefix odpovídá adrese sítě, zatímco adresa počítače je libovolná a podmínka tak vyhovuje IP adrese libovolného počítače dané sítě. Jde o speciální případ rozsahu.
- **Hodnota**: Pole musí mít přesně definovanou hodnotu. Jde o speciální případ prefixu nebo rozsahu.
- **Libovolná**: Pole může mít libovolnou hodnotu. Jde o speciální případ prefixu nebo rozsahu.

Ačkoliv rozsah je nejobecnější podmínka, často se podmínky reprezentují pomocí prefixů. Každý rozsah lze převést na několik prefixů, ilustrace je na obrázku 1.

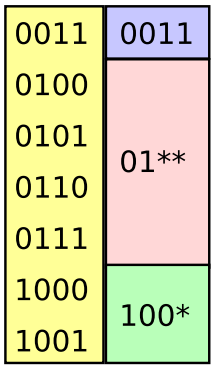
Počet prefixů, které mohou při této operaci vzniknout, je v nejhorším případě $2N-2$, kde N je počet bitů daného pole[1] [6]. Přesnou hodnotu v podmínce pravidla lze chápat jako prefix maximální délky (žádné bity nejsou volitelné). Podobně podmínku povolující libovolnou hodnotu lze chápat jako prefix nulové délky (všechny bity jsou volitelné).

Pokud hodnoty polí konkrétního paketu splňují všechny definované podmínky nějakého pravidla, vyhovuje paket danému pravidlu a tedy patří do dané třídy. Takto by paket mohl zároveň patřit do několika tříd, proto pravidla definují navíc prioritu. Potom je jako výsledek klasifikace vybráno platné pravidlo s nejvyšší prioritou. Nyní lze pravidlo definovat formálně:

Pravidlo je $(n+1)$-tice $R : (p, a_1, a_2, \ldots, a_N)$, kde $p \in \mathbb{N}$ je priorita a a_x jsou prefixy. N je počet dimenzí.

Značení: Je-li třeba pravidla číslovat, budeme používat dolní index $(R_1, R_2, \ldots)$. K označení priority a jednotlivých prefixů daného pravidla použijeme tečkovou notaci, takže prefix a_1 pravidla R_2 lze napsat jako $R_2.a_1$.

[1] Nejhorším případem je rozsah $\langle 1, 2^N - 2\rangle$



Obrázek 1: Převod rozsahu $\langle 3, 9\rangle$ na tři prefixy: 0011, 01**, 100*

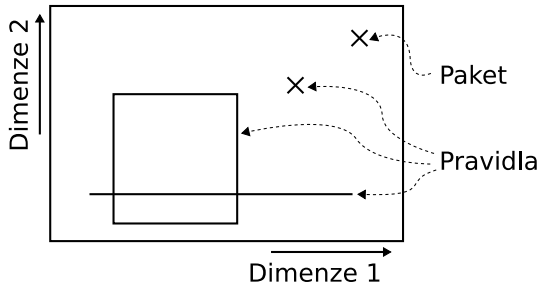
Klasifikátor je množina pravidel taková, že každá dvě pravidla mají rozdílnou prioritu.

$$K = \{R; R_i.p = R_j.p \Rightarrow i = j\} \tag{1}$$

Protože priorita je přirozené číslo a žádná dvě pravidla v klasifikátoru nemají stejnou prioritu, existuje úplné uspořádání pravidel podle priority.

2.3 Geometrický pohled na klasifikaci

Na klasifikaci lze pohlížet jako na geometrický problém. Každé pole, které klasifikujeme, definuje jednu dimenzi. Každý paket je potom jeden bod ve vícedimenzionálním diskrétním prostoru. Pravidla jsou body, nebo (protože mohou definovat rozsahy) vícerozměrné pravoúhlé objekty v tomto prostoru a mohou se libovolně překrývat. Úkolem klasifikačního algoritmu je potom nalézt všechny objekty, které obsahují zadaný bod, a vybrat z nich ten s nejvyšší prioritou. Obrázek 2 ilustruje geometrický pohled na klasifikaci podle dvou polí. V praxi se však používá více dimenzí, často pět (zdrojová IP adresa, cílová IP adresa, zdrojový port, cílový port, protokol).

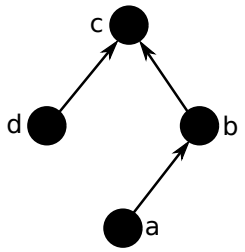


Obrázek 2: Ilustrace geometrického pohledu na klasifikaci podle dvou polí. Zatímco daný paket jednoznačně určuje jeden bod v prostoru, pravidla mohou mít podobu obdélníku (obecně vícerozměrného kvádru), úsečky nebo bodu.

2.4 Kombinatorický problém na klasifikaci

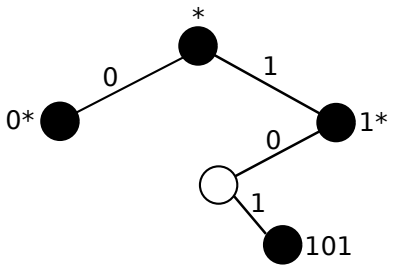
Jiný pohled nám poskytuje úvaha nad systémem prefixů v jednotlivých dimenzích. Pokud z klasifikátoru vybereme všechny prefixy v jedné dimenzi, dostaneme množinu prefixů, obecně různé délky. Takových množin lze vytvořit tolik, kolik je klasifikačních polí (dimenzí). Je zřejmé, že některé prefixy mohou být celé obsaženy v jiných, kratších prefixech. Jinými slovy, prefix a může být speciálním případem několika prefixů $A = \{b, c, ...\}$, kde každý prefix z A je kratší než prefix a, a žádné dva prefixy z A nemají

stejnou délku. Naopak prefix c může být obecnějším případem několika prefixů $C = \{a, b, ...\}$, kde každý prefix z B je delší než prefix c. Prefixy z B mohou mít stejnou délku. Na základě tohoto vztahu vzniká relace částečného uspořádání, kterou lze přehledně vyjádřit stromovým diagramem (obr. 3). Doplněním



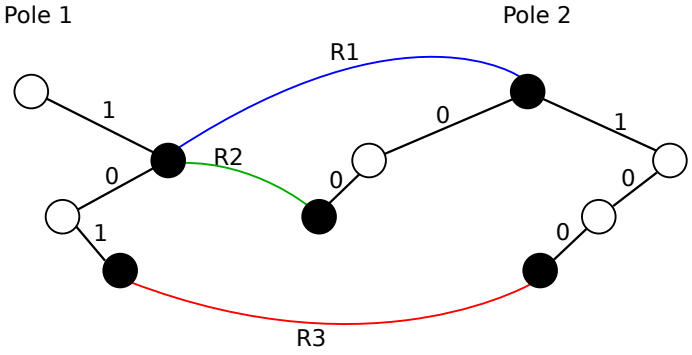
Obrázek 3: Čtyři prefixy v relaci částečného uspořádání. Prefix a je speciálním případem prefixů b a c. Prefix c je obecnějším případem všech ostatních prefixů.

myšlených prefixů lze získat binární strom (obr. 4). Pro zobrazení pravidla je nutné doplnit do obrázku



Obrázek 4: Konkrétní příklad prefixů z předchozího obrázku, doplněný do podoby binárního stromu. Používáme přímý zápis binárních čísel, hvězdička označuje libovolné zbylé bity.

binární stromy pro všechny dimenze (obr. 5). Pravidlo je potom spojnice jdoucí právě jedním uzlem v každém binárním stromě. Pro přehlednost jsou v tab. 1 pravidla vypsána.



Obrázek 5: Dva binární stromy a tři pravidla $R1, R2, R3$.

V této interpretaci problému je úkolem klasifikačního algoritmu nejprve vyhledat odpovídající prefixy ve všech dimenzích. V každé dimenzi může paketu odpovídat hned několik prefixů, výsledkem prvního kroku tedy není N prefixů, ale N množin prefixů. V dalším kroku je třeba vytvořit kartézský součin těchto množin a zjistit, které prvky kartézského součinu odpovídají některému pravidlu. Prvky tohoto kartézského součinu mají totiž tvar pravidla (až na chybějící prioritu). Výsledkem algoritmu bude pravidlo s nejvyšší prioritou.

Pravidlo	Pole 1	Pole 2
R1	1*	*
R2	1*	00*
R3	101*	100*

Tabulka 1: Pravidla z obr. 5

2.5 Požadavky na klasifikační algoritmus

Klasifikační algoritmus musí splňovat jisté vlastnosti, aby byl vhodný pro praktické použití v síťových zařízeních:

- **Rychlost**: Klasifikační algoritmus musí pracovat v reálném čase, tak, aby nesnižoval přenosovou rychlost sítě. Nejhorším případem je přenos nejkratších paketů, protože v takové situaci se přenáší nejvíc paketů za jednotku času. Tabulka 2 ukazuje počet přenesených paketů za sekundu pro typické přenosové rychlosti současných nebo budoucích sítí. Při výpočtu je uvažována síť typu Ethernet, kde má nejkratší rámec (odpovídá paketu na síťové vrstvě) velikost 64 B, ale ke každému rámci je nutno připočíst 8 B preambuli a 12 B mezirámcovou mezeru.

  Většina literatury uvádí složitost klasifikačního algoritmu jako nejhorší počet přístupů do paměti. Taková metrika je nezávislá na použité technologii. Pochopitelně existují i další kritéria rychlosti, například reálná propustnost v paketech za sekundu, nebo latence výpočtu.

- **Paměťové požadavky**: Velikost spotřebované paměti je důležitá, protože ovlivňuje cenu celého řešení. Často se udává v bajtech na pravidlo. Velikost paměti má také vliv na rychlost řešení. Obecně totiž platí, že menší paměti jsou rychlejší než paměti s větší kapacitou. Například použití blokových pamětí uvnitř čipu je rychlejší než využití externí statické paměti.

- **Plocha na čipu**: V případě hardwarové implementace algoritmu je důležitým ukazatelem zabraná plocha FPGA nebo ASIC čipu.

- **Počet a vlastnosti pravidel**: Důležitým parametrem řešení je také maximální počet pravidel, která lze nahrát do systému. Kromě množství pravidel je velice žádoucí zkoumat jejich vlastnosti. Především je třeba se zaměřit na počet unikátních hodnot pro jednotlivá pole v pravidlech. Ukazuje se, že množství unikátních prefixů v jednotlivých polích je většinou mnohem nižší než počet pravidel [16, 15]. Této skutečnosti s výhodou využívají pokročilé algoritmy klasifikace paketů.

Rychlost sítě [Gbit/s]	Paketová rychlost [paketů/s]	Doba zpracování paketu [ns/paket]
1	1 488 095	672
10	14 880 952	67,2
40	59 523 809	16,8
100	148 809 523	6,72

Tabulka 2: Počet přenesených paketů délky 64 B pro různé síťové rychlosti.

3 Současné přístupy ke klasifikaci paketů

Předchozí kapitola uvedla problém klasifikace paketů a vymezila podmínky, které musí algoritmus řešící popsaný problém splňovat. V této části práce je uvedeno několik základních metod pro klasifikaci paketů.

První dva uvedené algoritmy jsou pouze teoretickou možností, po nich následují již propracovanější metody. Jsou vybrány algoritmy určující směry, kterými se nejčastěji inspirují další navazující práce.

3.1 Naivní přístupy ke klasifikaci

Snad nejjednodušší přístup je *lineární prohledání množiny pravidel*. Stačí uložit všechna pravidla do jedné tabulky a každý paket postupně porovnat se všemi pravidly. Je zřejmé, že takový algoritmus má lineární časovou složitost s počtem pravidel a je tedy nevhodný z pohledu rychlosti. Pokud by množina pravidel obsahovala tisíc prvků, bylo by nutné udělat tisíc přístupů do paměti pro klasifikaci každého paketu. Tento algoritmus má však nejmenší nároky na paměť.

Opačným extrémem je algoritmus, který všechna rozhodující pole z hlavičky paketu spojí za sebe do jednoho datového slova a vzniklé slovo pak použije jako adresu do paměti. Na dané adrese v paměti je pak uloženo přímo číslo výsledného pravidla. Musíme tak vlastně mít *paměť s číslem pravidla pro každý možný paket*. Pro klasifikaci paketu pak stačí jediný přístup do paměti. Z toho důvodu je takový algoritmus teoreticky nejrychlejší možný. Jeho použití je však nereálné z důvodu paměťové náročnosti. Například: Spojíme-li dvě IPv4 adresy, dvě čísla portů a jedno číslo protokolu do jednoho datového slova, dostaneme adresu širokou $2 \cdot 32 + 2 \cdot 16 + 8 = 104$ bitů. Paměť by tedy musela obsahovat 2^{104} položek, což je za současného stavu počítačové techniky prakticky nemožné.

3.2 TCAM

Asociativní paměť je zvláštním typem paměti, která podporuje vyhledávání podle obsahu. Vstupem je hledané slovo a výstupem je adresa, na které se slovo nachází. Paměť má u každého slova komparátory, které vyhodnocují shodu hledaného slova s uloženým. Ternární varianta asociativních pamětí (TCAM) pracuje kromě jedniček a nul ještě s třetím stavem, nazývaným *don't care* a označovaným X. Je-li v paměti na nějaké bitové pozici tato hodnota, není při vyhledávání brán na daný bit zřetel, takže může být na vstupu nula nebo jednička a v obou případech se bit považuje za shodný.

Tím je přímo podporováno hledání prefixů. Do horní části datového slova se uloží požadované bity, zatímco spodní část slova se nastaví na don't care. Převedeme-li rozsahy v definici pravidel na prefixy, můžeme do TCAM ukládat přímo pravidla, a poté je vyhledávat jediným přístupem.

Z uvedeného by se mohlo zdát, že TCAM podporuje klasifikaci paketů v konstantní časové a lineární prostorové složitosti. Jednak něco takového není z definice složitosti možné (prostorová složitost nemůže nikdy přesáhnout časovou), navíc je nutné považovat paralelní vyhledávání v TCAM za vícepáskový Turingův stroj (každá páska pro jedno pravidlo). Ten po převedení na jednopáskový získá lineární časovou i prostorovou složitost s počtem pravidel.

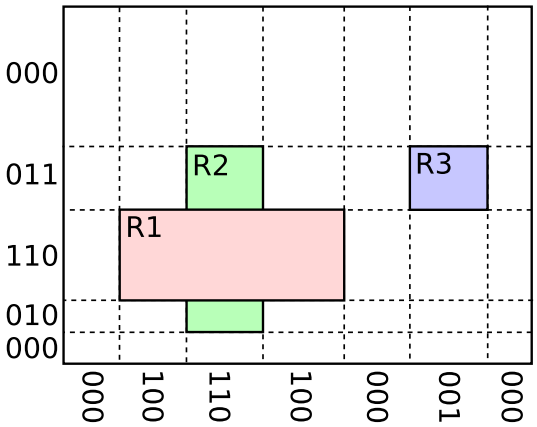
TCAM mají omezenou kapacitu, poměrně velký příkon a také vysokou pořizovací cenu. I přes známé nevýhody jsou paměti TCAM často používány v komerčních zařízeních [12, 5]. Uvedené nevýhody způsobují, že stále existuje snaha nacházet a zkoumat algoritmy, které používají paměti s náhodným přístupem.

3.3 BitVector algoritmus

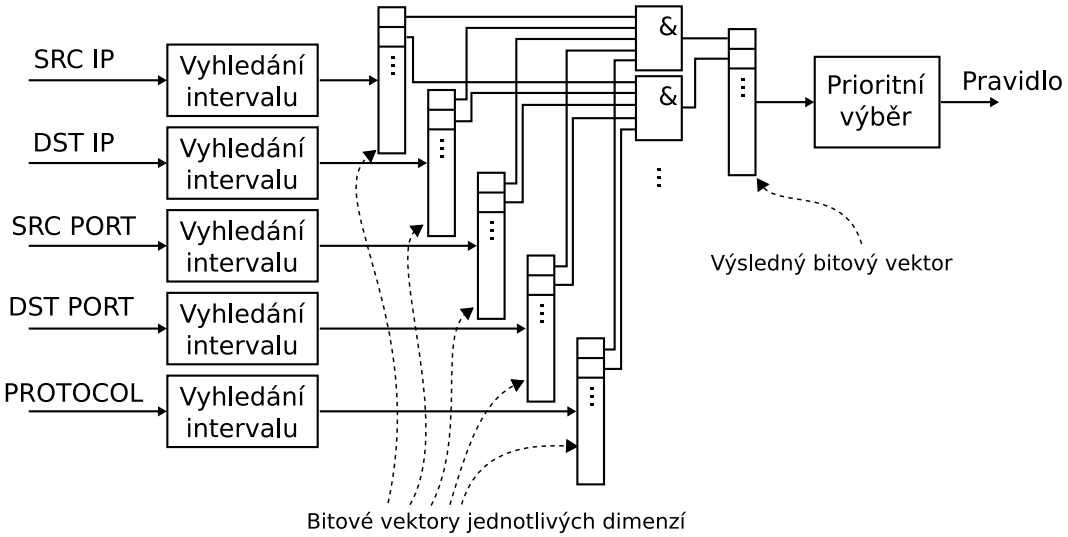
Algoritmus BitVector [8] pohlíží na klasifikaci jako na geometrický problém. Algoritmus nejprve promítne v každé dimenzi všechny rozsahy na číselnou osu. Tím na ose vznikne maximálně $2n+1$ nepřekrývajících se intervalů, kde n je počet různých rozsahů v této dimenzi. Každému intervalu je přiřazen vektor, který obsahuje jeden bit pro každé pravidlo klasifikátoru. Bity vektoru jsou nastaveny na jedničku právě když se v této dimenzi dané pravidlo překrývá s intervalem (obrázek 6).

Klasifikace paketů potom spočívá ve vyhledání intervalu pro danou hodnotu pole paketu (lze provést binárním hledáním – logaritmická časová složitost) ve všech polích nezávisle. Pro každé pole tak získáme jeden vektor. Pokud nad těmito vektory provedeme bitový součin, zůstanou ve výsledku jedničky pouze pro ta pravidla, která byla splněna ve všech dimenzích. Struktura algoritmu je naznačena na obrázku 7.

Byly navrženy některé úpravy tohoto algoritmu (například *Aggregated Bit Vector*[1], nebo *BV-TCAM* [13]), které dosahují lepších výsledků jak paměťové, tak časové náročnosti.



Obrázek 6: Výpočet vektorů pro dvě dimenze. Tři pravidla vytvořila 7 intervalů na vodorovné ose a 5 intervalů na svislé ose. Ke každému intervalu je přiřazen vektor tří bitů, protože klasifikátor obsahuje tři pravidla.



Obrázek 7: Schéma klasifikace algoritmem BitVector. Po vyhledání intervalu v každé dimenzi je proveden bitový součin všech vektorů a ve výsledném vektoru je nalezeno pravidlo s nejvyšší prioritou.

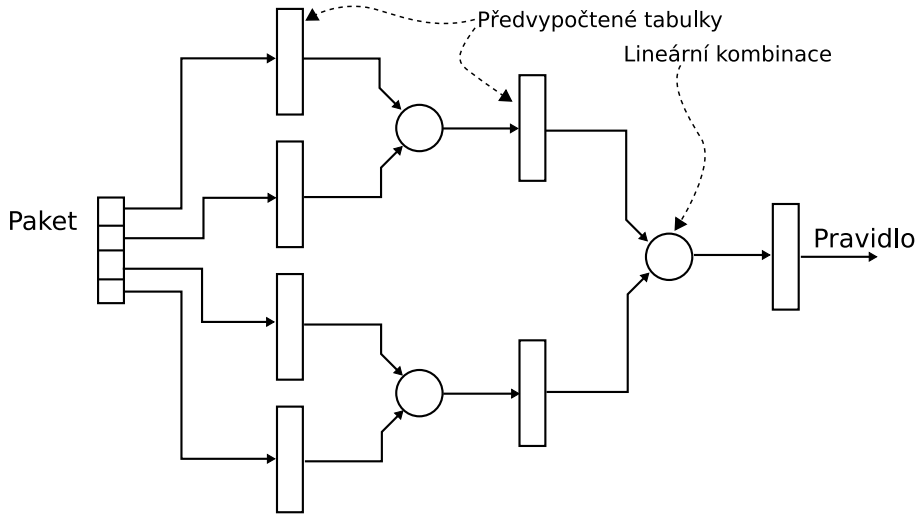
Nicméně právě počet přístupů do paměti je slabinou algoritmů založených na Bit Vector. Pro větší počet pravidel roste délka vektoru tak, že je nutné přistoupit několikrát do paměti pro načtení celého vektoru. Tento fakt je příčinou nižší rychlosti algoritmu.

3.4 RFC algoritmus

Recursive Flow Classification (popsaný v [6]) vychází z naivního přístupu s velikou tabulkou pro všechny možné pakety. Rozdíl je v tom, že RFC postupuje rekurzivně. Pole z hlavičky paketu jsou rozdělena na slova se vhodnou datovou šířkou (např. 16 bitů), a vzniklá slova jsou použita jako adresy do tabulek. Tabulky jsou předvyplněny hodnotami tak, aby pakety patřící do různých pravidel dávaly odlišné hodnoty (rozklad na třídy ekvivalence). Několik výstupů těchto tabulek je spojeno lineární kombinací a použito jako adresa do další tabulky.

Lineární kombinace je násobení konstantou a součet, přičemž konstanty jsou voleny tak, aby pro různé vstupy byl vždy různý výsledek. Například pro lineární kombinaci dvou hodnot je jedna hodnota nejprve vynásobena počtem prvků druhé množiny a potom jsou obě hodnoty sečteny.

Takto je vybudována hierarchická struktura, která postupně snižuje počet bitů, kterými adresujeme, až na konci je tabulka obsahující samotná pravidla. Schéma výpočtu je na obrázku 8. Tento algoritmus



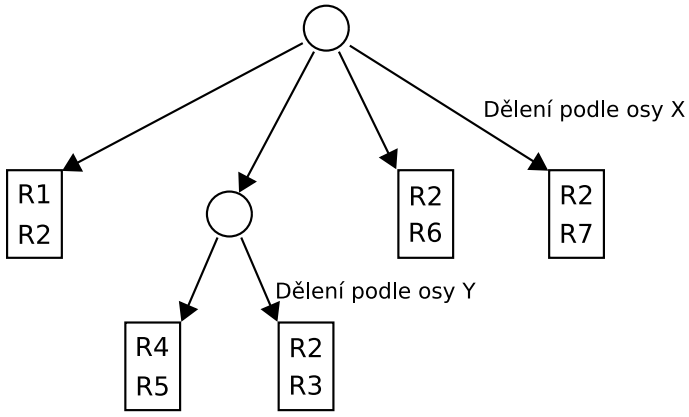
Obrázek 8: Schéma klasifikace algoritmem RFC. Na první úrovni jsou adresy do tabulek přímo hodnoty polí z hlavičky paketu, dále jsou výsledky kombinovány a použity jako adresy dalších tabulek. Poslední tabulka obsahuje pravidla.

poskytuje velice dobré výsledky z pohledu výkonnosti, ale jeho slabinou je paměťová náročnost, která je mnohokrát větší než u ostatních metod [12].

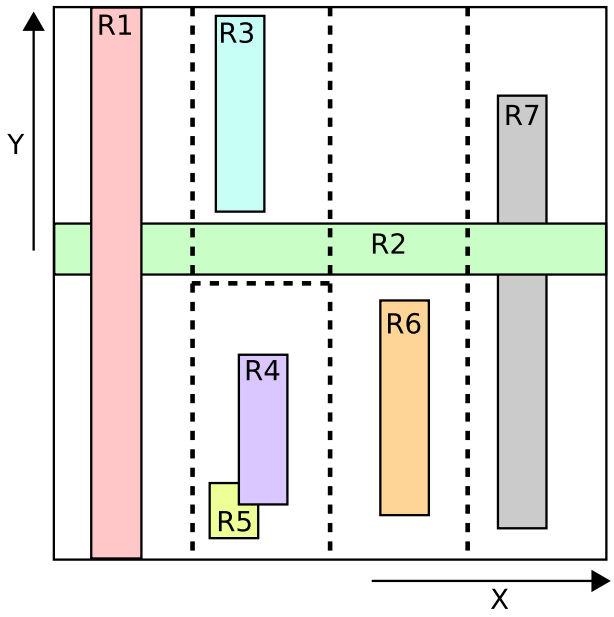
3.5 Rozhodovací stromy

Hierarchical Intelligent Cuttings (Hi-Cuts) [6] je algoritmus vycházející z geometrické repre- zentace klasifikace. Spočívá v konstrukci rozhodovacího stromu. Při průchodu stromem je v každém uzlu prostor paketů dělen rovnoběžnými plochami na více oblastí. Další postup stromem je zvolen takový, aby klasifikovaný paket ležel v dané oblasti. Takto je nakonec prostor omezen tak, že dostáváme pouze oblast obsahující správná pravidla.

Aby algoritmus šetřil paměť, je několik posledních stupňů stromu nahrazeno lineárním prohledáním. Pro konstrukci rozhodovacího stromu je navržena heuristika. Obrázek 9 ukazuje příklad jednoduchého rozhodovacího stromu.



Obrázek 9: Rozhodovací strom algoritmu Hi-Cuts. Uzly obsahují informaci o větvení stromu (ve které dimenzi, na kolik částí) a ukazatele na následující uzly nebo listy. Listy obsahují několik pravidel.



Obrázek 10: Plocha rozdělená podle rozhodovacího stromu z obrázku 9. První dělení je ve vodorovné ose na čtyři části, dále je jedna část rozdělena svisle na dvě části. Některá pravidla mohou zasahovat do více částí, potom se objeví ve více listech stromu.

Hypercuts [12] je modifikací předchozího algoritmu tak, aby bylo možné v jednom uzlu stromu dělit

prostor podle více než jedné dimenze. Tím je dosaženo menší hloubky stromu a tedy urychlení algoritmu. Experimentální výsledky ukazují také na menší paměťovou náročnost.

3.6 Algoritmy založené na dekompozici problému

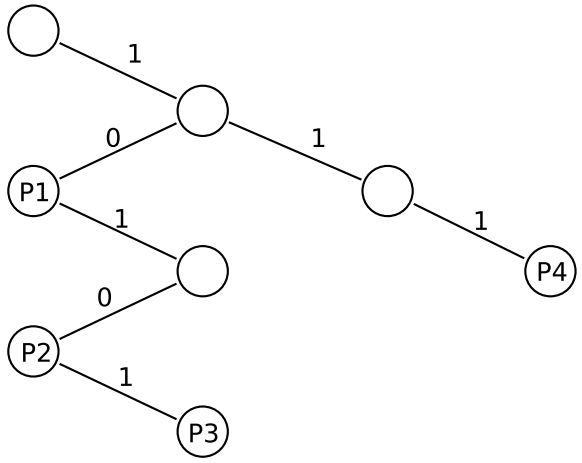
Dekompoziční algoritmy řeší klasifikaci ve dvou základních krocích, které odpovídají kombinatorickému pohledu na klasifikaci, uvedenému v části 2.4. V prvním kroku je vyhledán nejdelší shodný prefix pro každé pole. To znamená, že ze všech prefixů pro jednotlivé pole daného klasifikátoru je nalezen ten nejspecifičtější, který odpovídá hodnotě v hlavičce konkrétního paketu. Protože výsledky prvního kroku neidentifikují jednoznačně pravidlo, následuje proces hledání odpovídajícího pravidla. Výsledkem druhého kroku je již číslo nalezeného pravidla. Ve druhém kroku je nutné vyřešit problém tzv. *pseudopravidel*.

V této části je nejprve vysvětleno několik souvisejících pojmů, a pak jsou uvedeny některé důležité algoritmy ze této skupiny.

Vyhledání nejdelšího shodného prefixu

Algoritmus vyhledání nejdelšího prefixu (Longest Prefix Match - LPM) má na vstupu množinu prefixů různé délky a jednu konkrétní hodnotu. Výstupem algoritmu je ze všech prefixů, které odpovídají dané hodnotě ten nejdelší, tedy nejspecifičtější.

Klasický algoritmus vyhledání nejdelšího prefixu se nazývá *trie* z anglického *retrieval*, někdy také označovaný *prefixový strom*. Je zde využita stromová datová struktura, která obsahuje uložená slova přímo ve své konstrukci. Každý uzel stromu obsahuje dva (v případě binární trie) ukazatele na další uzly. V průběhu výpočtu se zpracovávají postupně bity hledaného slova od nejvýznamnějšího k nejméně významnému. V každém kroku výpočtu se podle tohoto bitu rozhoduje, zda se bude postupovat levým nebo pravým podstromem. V každém uzlu je označeno, zda zatím zpracované bity tvoří platný prefix. Pokud ano, je zde také uloženo číslo tohoto prefixu pro identifikaci. Při průchodu stromem je zapamatován poslední platný prefix a na konci je předán jako výsledek. Výpočet je ukončen, když s danou hodnotou bitu již není možné postupovat dále stromem (potřebný podstrom neexistuje), nebo jsou již zpracovány všechny vstupní bity. Časová složitost vyhledání je lineární s počtem bitů datového slova. Jednoduchý příklad struktury trie je na obrázku 11.



Obrázek 11: Trie pro pětibitové pole. Jsou zde uloženy 4 prefixy: 10*, 1010*, 10101, 111*.

Existuje řada modifikací trie, základní z nich je zvýšení arity uzlů. V každém kroku se potom rozhoduje podle více než jednoho bitu a uzel se může tedy větvit do více než dvou podstromů.

Pseudopravidla

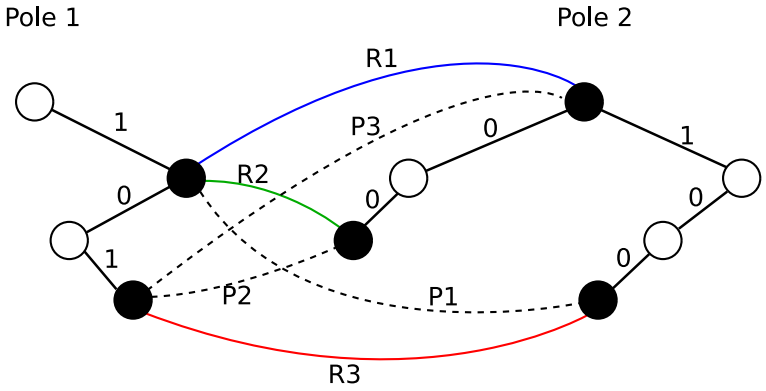
Pojem *pseudopravidlo*, zavedený v [5], označuje pravidla, která je nutné přidat do množiny pravidel tak, aby každá platná kombinace výsledků LPM byla pokryta. Vznik pseudopravidel demonstrujme na příkladě klasifikace podle dvou polí. Tabulka 3 obsahuje tři jednoduchá pravidla.

Pravidlo	**Pole 1**	**Pole 2**
R1	1*	*
R2	1*	00*
R3	101*	100*

Tabulka 3: Původní pravidla

Pseudopravidlo	**Pole 1**	**Pole 2**	**Pravidlo**
P1	1*	100*	R1
P2	101*	00*	R2
P3	101*	*	R1

Tabulka 4: Přidaná pseudopravidla



Obrázek 12: Reprezentace pravidel a pseudopravidel (čárkovaně) pomocí dvou trie. Pseudopravidla jsou specifičtějšími případy pravidel.

Na obrázku 12 jsou potom naznačeny trie pro LPM v obou dimenzích. Vyplněné uzly značí platné prefixy. Barevnými čarami jsou označeny kombinace vstupů, které byly definovány v původní množině pravidel. Čárkovaně jsou vyznačena pseudopravidla. Jde o doplnění množiny pravidel tak, aby i specifičtější výsledky LPM měly smysl a bylo z nich možné snadno určit správné pravidlo. Tabulka 4 obsahuje pseudopravidla pro tento příklad.

Na pseudopravidla lze pohlížet také jako na způsob, jak se vyhnout vyhodnocování kartézského součinu popsanému v části 2.4. Díky operaci LPM nemá vyhledání v jednotlivých dimenzích za výsledky množiny prefixů, ale jen ty nejspecifičtější (tedy nejpřesnější). Žádný kartézský součin tedy nevzniká. A díky přidání pseudopravidel je každá taková kombinace výsledků platná a odkazuje na správné číslo pravidla.

Příklad: Pokud by operace LPM vrátily výsledek (1*, 100*), potom správným výsledkem klasifikace je pravidlo R1, protože 100* je speciální případ obecnější hodnoty *. Nicméně v původní množině pravidel se kombinace (1*, 100*) vůbec nevyskytuje. Proto je nutné přidat pseudopravidlo P1, které uvedenou kombinaci ošetřuje. Pseudopravidlo v sobě obsahuje informaci, že správným výsledkem klasifikace je pravidlo R1.

Generovaná pseudopravidla připomínají kartézský součin, jelikož je nutné pro každé pravidlo pokrýt všechny specifičtější prefixy ve všech polích. Vzhledem k charakteru kartézského součinu může růst počet pseudopravidel exponenciálně s počtem pravidel. V reálných klasifikátorech sice nemusí docházet k tak výraznému nárůstu, ale přesto experimentální výsledky ukazují na řádové zvětšení potřebné paměti [5].

Naivní algoritmus kartézského součinu

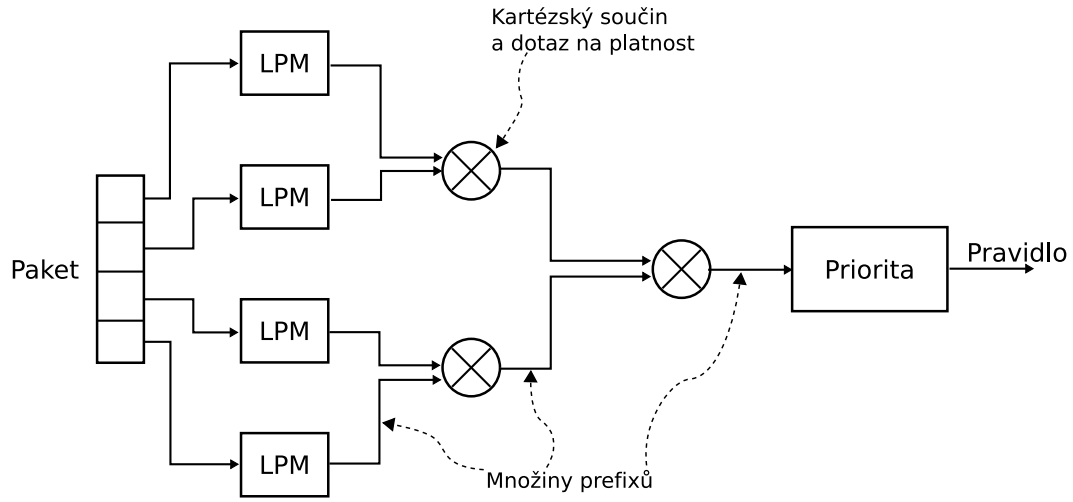
Pakety lze klasifikovat tak, že se nejprve nalezne LPM pro každé pole a výsledky se spojí do jednoho datového slova [14]. Toto slovo je vstupem hashovací funkce. Výstupem je potom adresa do tabulky, ve které jsou uložena jak pravidla, tak pseudopravidla. Jsou-li vhodně ošetřeny kolize hashovací funkce, je časová složitost takového dohledání konstantní.

Nevýhodou popsaného algoritmu je paměťová náročnost, protože nárůst počtu pseudopravidel může být exponenciální.

DCFL

V *Distributed Crossproducting of Field Labels* [15] je operace LPM modifikována tak, aby výsledkem nebyl pouze jediný, nejdelší prefix. Namísto toho je výsledkem množina všech prefixů, na které algoritmus po cestě stromem narazil (tedy obecnější prefixy).

V dalším kroku je nutné zkusit všechny kombinace výsledků a zjistit, zda některá z nich odpovídá některému pravidlu. Jak už bylo uvedeno, taková operace je kartézský součin několika množin a má exponenciální časovou složitost. DCFL popsaný problém řeší tak, že kombinace prefixů vyhodnocuje distribuovaně. Kartézský součin se provádí vždy pouze na dvou množinách - zkoumá se, zda se v některém pravidle vyskytla kombinace daných dvou prefixů. Pro dotazy jsou použita pole Bloomových filtrů [2]. Z kartézského součinu mohou být některé kombinace zamítnuty, protože neodpovídají žádnému pravidlu. Ostatní, nezamítnuté kombinace procházejí do dalších stupňů, které pracují na stejném principu. Takto je vytvořena stromová struktura, naznačená na obrázku 13. Výsledkem posledního stupně je množina pravidel odpovídajících danému paketu.



Obrázek 13: Schéma algoritmu DCFL pro klasifikaci podle 4 polí. Po vyhledání všech prefixů pro každé pole jsou zkoumány všechny kombinace prefixů.

Ze schématu výpočtu je vidět, že algoritmus je vhodný pro paralelní provádění, protože jednotlivé operace LPM mohou běžet paralelně, podobně jako vyhodnocování kartézských součinů.

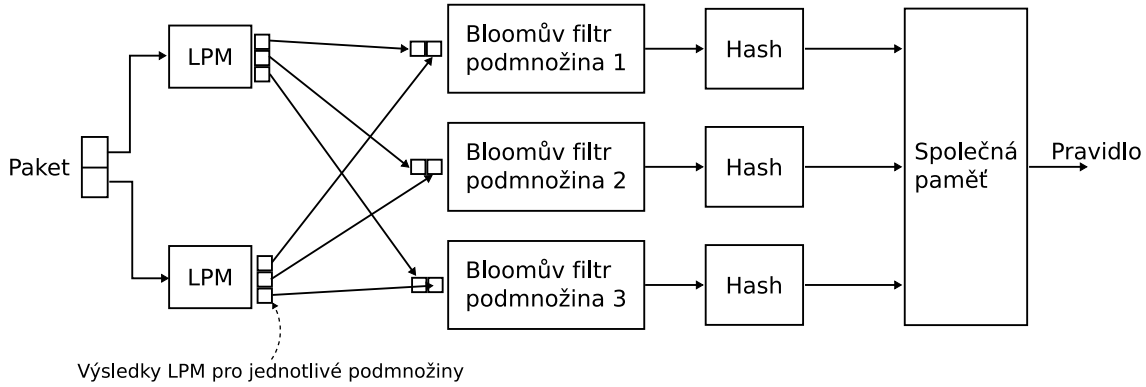
Slabinou algoritmu je postupné vyhodnocování kartézského součinu. Jsou-li výsledky LPM např. dvě množiny čtyřech prefixů, potom kartézský součin obsahuje 16 položek, přičemž pro každou z nich musíme ověřit její přítomnost v množině. I když takové ověřování lze provádět s konstantní časovou složitostí, je nutné značně paralelní vykonávání několika těchto operací naráz. Navíc použití Bloomových filtrů přináší možnost chyby, kterou je nutné v pozdějších fázích výpočtu eliminovat.

Případná analýza časové složitosti musí počítat s nejhorším případem ve všech částech výpočtu, přičemž nejhorší případ se může značně odlišovat od typického.

MSCA

Velmi důležitý Multi-Subset Crossproduct Algorithm, popsaný v [5], vychází z pozorování, že množinu pravidel lze rozdělit do několika podmnožin tak, aby vznikalo pouze velmi málo pseudopravidel. Experimentální výsledky ukazují, že počet pravidel se díky tomu zvětší pouze 1,2 až 2 krát.

Je-li množina pravidel rozdělena do více podmnožin, je třeba pro každou podmnožinu provádět oddělené vyhledání nejdelšího shodného prefixu. Tomu se algoritmus vyhne tak, že jako výsledek operace LPM uloží výsledek pro každou podmnožinu. Také je nutné pro každou podmnožinu odděleně provádět hashování a přístup do paměti, abychom zjistili, zda bylo nalezeno platné pravidlo. Tomu se však lze vyhnout použitím Bloomových filtrů. Celé schéma výpočtu je na obrázku 14.



Obrázek 14: Schéma algoritmu založeného na rozdělení množiny pravidel do podmnožin. (Zobrazena klasifikace podle dvou polí, tři podmnožiny pravidel)

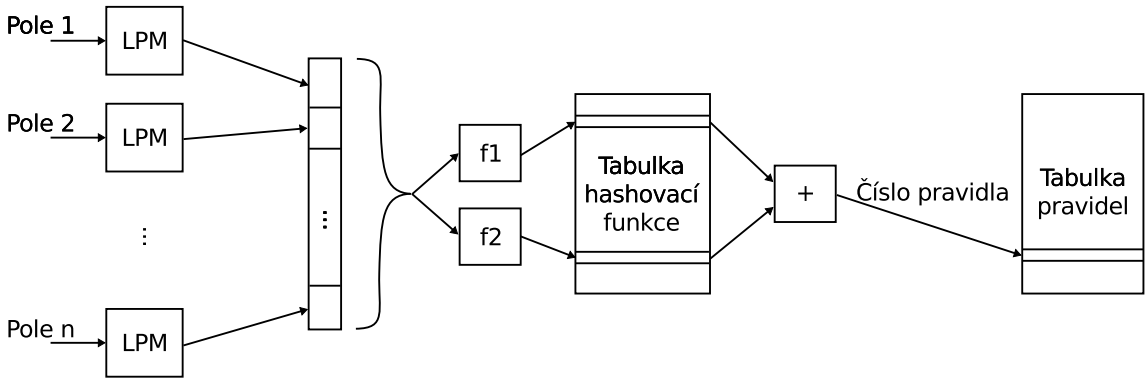
Také algoritmus MSCA je, podobně jako DCFL, velice výhodný z pohledu rychlosti i využité paměti. Za slabinu bychom zde mohli považovat využití Bloomových filtrů, které připouštějí chybu typu *false positive*, tedy chybná detekce přítomnosti slova v množině. Tyto chyby jsou sice následně odstraněny kontrolou obsahu hlavní paměti, ale způsobují zvyšování počtu přístupů do paměti, a tedy v nevýhodné situaci mohou degradovat výkon algoritmu.

Tento algoritmus je však zajímavý z toho důvodu, že přináší dva způsoby, jak snižovat množství pseudopravidel. Prvním způsobem je samotné rozdělení klasifikátoru na podmnožiny. Druhou optimalizací je použití malé TCAM (řádově na jednotky či desítky pravidel), kam se uloží ta pravidla, která způsobovala největší nárůst počtu pseudopravidel. Pozorováním bylo totiž zjištěno, že největší počet pseudopravidel je způsoben jenom malým množstvým pravidel. Pokud se tedy několik těchto pravidel odstraní z klasifikátoru, a jejich vliv se zajistí jinak (např. malou TCAM), počet pseudopravidel je výrazně omezen. Některé způsoby identifikace pravidel, která generují nejvíce pseudopravidel, jsou popsány v [7].

PHCA

Perfect Hashing Crossproduct Algorithm [10] se zaměřuje na odstranění nutnosti ukládat pseudopravidla, jak se to děje v naivním algoritmu kartézského součinu a v MSCA. Schéma algoritmu je naznačeno na obr. 15. Předem je zkonstruována hashovací funkce na míru tak, aby všechny kombinace výsledků LPM mapovala přímo na správné číslo pravidla. Je k tomu využit algoritmus konstrukce perfektní (tedy bezkolizní) hashovací funkce [3]. Její vyčíslení spočívá ve vyčíslení dvou různých běžných hashovacích funkcí $f1$ a $f2$, dvou přístupech do tabulky, a jednom aritmetickém součtu. Tabulka je předem vypočtena na základě klíčů a požadovaných výsledků hashovací funkce.

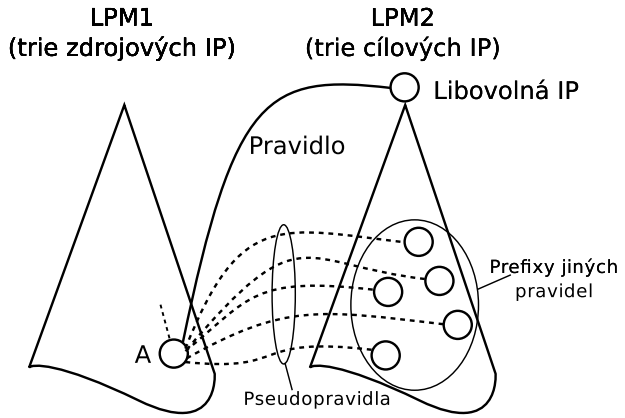
Hashovací funkce však v tomto případě zkonstruována záměrně tak, aby obsahovala velké množství kolizí. To je z toho důvodu, že počet pseudopravidel je výrazně vyšší než počet pravidel. Kolize hashovací funkce tedy slouží k tomu, aby se všechna pseudopravidla mapovala právě na pravidlo kterému odpovídají. Výjimkou jsou pakety neodpovídající žádnému pravidlu. Pro ně není výsledek hashovací funkce ošetřen, ale hashovací funkce přesto vždy vrátí nějaký výsledek, tedy vždy označí některé pravidlo. Tuto potenciální chybu lze odstranit prostým porovnáním paketu proti označenému pravidlu.



Obrázek 15: Schéma algoritmu PHCA. Závěrečné porovnání paketu proti vybranému pravidlu není zobrazeno.

Další optimalizace

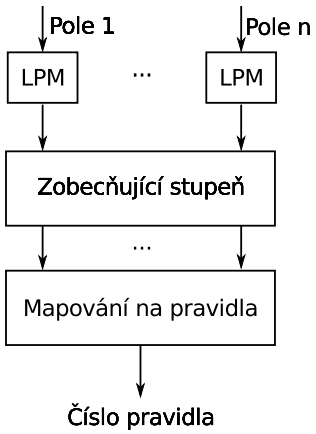
Algoritmus popsaný v [11] se dále zabývá snižováním počtu pseudopravidel. Vychází z pozorování, že pravidla často nedefinují podmínku pro všechna pole z hlavičky paketu. Přesněji: Definují podmínku jako libovolnou hodnotu. Pseudopravidla takového pravidla potom musí pokrýt všechny specifické prefixy v daném poli. Situace pro dvě dimenze (zdrojovou a cílovou IP adresu) je naznačena na obr. 16.



Obrázek 16: Vznik velkého množství pseudopravidel z jednoho obecného pravidla

Je navržen zobecňující výpočetní krok, jakýsi filtr, který zpracovává výsledky LPM. Tento krok je vložen jako mezikrok do některého existujícího algoritmu (obr. 17). Zobecňující krok na základě definovaných zobecňujících pravidel nahrazuje specifické výsledky hodnotou *ANY*. Tato hodnota reprezentuje libovolný prefix, je tedy nejobecnější a nejméně přesná. Nahrazením dochází ke ztrátě určité informace. Existují proto omezení na to, kdy lze takové nahrazení provést, aby bylo stále možné paket správně klasifikovat.

Ve výsledku se na výstupu zobecňujícího kroku může objevit méně různých kombinací než na vstupu, a tím dochází ke zmenšení počtu pseudopravidel. Následuje krok který nějakým způsobem mapuje pseudopravidla na pravidla (PHCA, nebo jiný algoritmus). Protože však na vstupu tohoto kroku je méně možných kombinací než bez použití optimalizace, dochází v tomto kroku k úspoře paměti.



Obrázek 17: Vložení zobecňujícího kroku do výpočtu.

4 Shrnutí

V tomto technickém reportu byly představeny některé základní přístupy ke klasifikaci paketů. Zkoumáme-li vlastnosti uvedených algoritmů, zjistíme že algoritmy založené na kartézském součinu polí (naivní, DCFL, MSCA a PHCA) se nejvíce blíží ke konstantní časové složitosti. Například algoritmus PHCA obsahuje tři základní kroky:

- *Operace LPM.* Je-li implementována jako stromový algoritmus podobný trie, má složitost lineární s počtem bitů vstupního slova. Nicméně tento počet je konstantní a předem známý, takže lze LPM považovat za operaci s konstantní složitostí. Navíc existují algoritmy (např. [4]), které slibují skutečně konstantní složitost operace LPM.
- *Zkonstruovaná hashovací funkce.* Její vyčíslení je velice snadné, jde o vyčíslení dvou běžných hashovacích funkcí, dva přístupy do paměti, a prostý aritmetický součet. Všechny části tohoto kroku mají konstantní časovou složitost.
- *Kontrola.* V tomto kroku je nutné porovnat paket proti jednomu pravidlu. Jde o triviální operaci s konstantní časovou složitostí.

Protože všechny kroky algoritmu mají konstantní složitost, lze tvrdit že algoritmus PHCA má konstantní časovou složitost vyhledání. Pro uložení všech potřebných dat je však potřeba paměť, která roste exponenciálně s počtem pravidel (pseudopravidla!). Abychom tedy dodrželi definici složitosti, je nutné uvést, že časová složitost vytvoření potřebných datových struktur je exponenciální.

Exponenciální prostorová složitost je hlavní nevýhodou těchto algoritmů. Bylo navíc dokázáno, že je-li časová složitost vyhledání konstantní, nemůže být prostorová složitost nižší než exponenciální [9]. Protože prostorová složitost má tyto své teoretické limity, zaměřuje se výzkum na možnosti praktického snížení potřebné paměti. Jde o různé optimalizace stávajících algoritmů, založené na zkoumání struktury typických klasifikátorů. Existující optimalizace algoritmů jsou shrnuty v tabulce 5.

Reference

[1] Florin Baboescu and George Varghese. Scalable packet classification. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 199–210, New York, NY, USA, 2001. ACM.

[2] Burton Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 1970.

Optimalizace	**Vliv na rychlost**	**Přidané nároky na paměť**
Rozdělení na podmnožiny	Může vyústit v násobný přístup do tabulky pravidel	Delší výsledky LPM, Bloomovy filtry pro každou podmnožinu
Malá TCAM	Vyhledání v TCAM probíhá paralelně s hlavním výpočtem	TCAM na pravidla (dlouhé datové slovo)
Zobecňující filtr	Vložený krok v pipeline	Několik malých CAM na porovnání výsledku LPM (krátké datové slovo)

Tabulka 5: Známé přístupy ke snižování velikosti potřebné paměti klasifikačních algoritmů

[3] Zbigniew J. Czech, George Havas, and Bohdan S. Majewski. An optimal algorithm for generating minimal perfect hash functions. *Information Processing Letters*, 43(5):257–264, 1992.

[4] Sarang Dharmapurikar, Praveen Krishnamurthy, and David E. Taylor. Longest prefix matching using bloom filters. In *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 201–212, New York, NY, USA, 2003. ACM.

[5] Sarang Dharmapurikar, Haoyu Song, Jonathan Turner, and John Lockwood. Fast packet classification using bloom filters. In *ANCS '06: Proceedings of the 2006 ACM/IEEE symposium on Architecture for networking and communications systems*, pages 61–70, New York, NY, USA, 2006. ACM.

[6] Pankaj Gupta. *Algorithms for Routing Lookups and Packet Classification*. PhD thesis, Stanford University, 2000.

[7] Michal Kajan. Optimalizace klasifikačních algoritmů založených na kartézském součinu. diplomová práce, FIT VUT, Brno, 2008.

[8] T. V. Lakshman and D. Stiliadis. High-speed policy-based packet forwarding using efficient multi-dimensional range matching. *SIGCOMM Comput. Commun. Rev.*, 28(4):203–214, 1998.

[9] Mark H. Overmars and A. Frank van der Stappen. Range searching and point location among fat objects. *J. Algorithms*, 21(3):629–656, 1996.

[10] Viktor Puš. Klasifikace paketů s využitím technologie fpga. diplomová práce, FIT VUT, Brno, 2007.

[11] Viktor Puš, Jan Kořenek, and Juraj Blaho. Memory optimization for packet classification algorithms. In *ANCS '09: Proceedings of the 5th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, New York, NY, USA, 2009. ACM.

[12] Sumeet Singh, Florin Baboescu, George Varghese, and Jia Wang. Packet classification using multidimensional cutting. In *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 213–224, New York, NY, USA, 2003. ACM.

[13] Haoyu Song and John W. Lockwood. Efficient packet classification for network intrusion detection using fpga. In *FPGA '05: Proceedings of the 2005 ACM/SIGDA 13th international symposium on Field-programmable gate arrays*, pages 238–245, New York, NY, USA, 2005. ACM.

[14] V. Srinivasan, G. Varghese, S. Suri, and M. Waldvogel. Fast and scalable layer four switching. *SIGCOMM Comput. Commun. Rev.*, 28(4):191–202, 1998.

[15] D. Taylor and J. Turner. Scalable packet classification using distributed crossproducting of field labels. In *IEEE INFOCOM 2005, 24th Annual Joint Conference of the IEEE Computer and Communications Societies.*, pages 269–280, July 2005.

[16] David E. Taylor. *Survey and Taxonomy of Packet Classification Techniques*. Washington University in Saint Louis, 2004.