VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY

DEPARTMENT OF COMPUTER SYSTEMS

KLASIFIKACE PAKETŮ S VYUŽITÍM TECHNOLOGIE FPGA

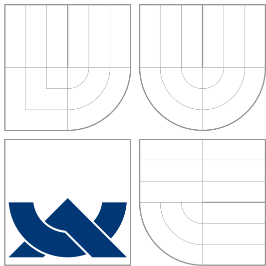
DIPLOMOVÁ PRÁCE

MASTER'S THESIS

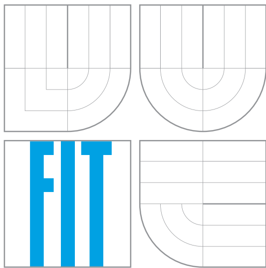
AUTOR PRÁCE — Bc. VIKTOR PUŠ

AUTHOR

BRNO 2008



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

KLASIFIKACE PAKETŮ S VYUŽITÍM TECHNOLOGIE FPGA

PACKET CLASSIFICATION USING THE FPGA TECHNOLOGY

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. VIKTOR PUŠ
AUTHOR

VEDOUCÍ PRÁCE — Ing. JAN KOŘENEK, Ph.D.
SUPERVISOR

BRNO 2008

Abstrakt

Tato diplomová práce se zabývá klasifikací paketů v počítačových sítích. Je popsán problém klasifikace paketů a jsou definovány požadavky, které musí splňovat klasifikační algoritmus. Dále je zavedena potřebná teorie a popsány současné přístupy ke klasifikaci paketů, spolu s kritikou současného stavu v této oblasti. Těžištěm práce je nový algoritmus klasifikace paketů založený na dekompozici problému. Unikátní vlastností algoritmu je konstantní časová složitost vzhledem k počtu přístupů do externí paměti. Je navržena implementace algoritmu v FPGA s využitím jedné externí paměti. Plánovaný prototyp může dosáhnout propustnosti 64 Gbit/s v nejhorším případě.

Klíčová slova

Klasifikace paketů, algoritmus, hardware

Abstract

This diploma thesis deals with packet classification in computer networks. The problem of packet classification is described, together with requirements for classification algorithm. Then, necessary theoretical background is introduced. Contemporary approaches to the classification are described, together with the critique of the current state of the field. The main focus of the work is the new algorithm of packet classification based on problem decomposition. Unique property of the algorithm is constant time complexity in terms of external memory accesses. Algorithm implemetation is proposed, using FPGA and one external memory. Planned prototype may achieve throughput of 64 Gbit/s in the worst case.

Keywords

Packet classification, algorithm, hardware

Citace

Viktor Puš: Klasifikace paketů s využitím technologie FPGA, diplomová práce, Brno, FIT VUT v Brně, 2008

Klasifikace paketů s využitím technologie FPGA

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Jana Kořenka.

....
Viktor Puš
12. května 2008

Poděkování

Děkuji vedoucímu své diplomové práce Ing. Janu Kořenkovi, Ph.D. za kvalitní vedení a odbornou pomoc, která přesahovala jeho povinnosti.

© Viktor Puš, 2008.
Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.

Obsah

1 Úvod **3**

2 Teoretický rozbor **5**
- 2.1 Vrstvový model TCP/IP . . . 5
- 2.2 Model klasifikace . . . 5
- 2.3 Klasifikační pravidla . . . 6
- 2.4 Geometrická reprezentace klasifikace . . . 7
- 2.5 Požadavky na klasifikační algoritmus . . . 7

3 Přístupy ke klasifikaci **9**
- 3.1 Naivní přístupy ke klasifikaci . . . 9
- 3.2 TCAM . . . 9
- 3.3 BitVector algoritmus . . . 10
- 3.4 RFC algoritmus . . . 11
- 3.5 Rozhodovací stromy . . . 12
- 3.6 Algoritmy založené na dekompozici problému . . . 13
  - 3.6.1 Vyhledání nejdelšího shodného prefixu . . . 13
  - 3.6.2 Bloomův filtr . . . 15
  - 3.6.3 Pseudopravidla . . . 16
  - 3.6.4 Naivní algoritmus kartézského součinu . . . 17
  - 3.6.5 DCFL . . . 18
  - 3.6.6 Rozdělení množiny pravidel na podmnožiny . . . 19

4 Navržený algoritmus **20**
- 4.1 Perfektní hashovací funkce . . . 21
- 4.2 Základní schéma výpočtu . . . 22
- 4.3 Analýza časové a paměťové složitosti . . . 24
- 4.4 Redukce paměťové náročnosti . . . 26
- 4.5 Algoritmus s paměťovou optimalizací . . . 27
- 4.6 Analýza optimalizovaného algoritmu . . . 27

5 Implementace **30**
- 5.1 Vyhledání nejdelšího shodného prefixu . . . 30
- 5.2 Výpočet hashovacích funkcí . . . 33
- 5.3 Tabulka vrcholů . . . 34
- 5.4 Tabulka pravidel . . . 34
- 5.5 Výstupní logika a asociativní paměť . . . 36
- 5.6 Příklad aplikace . . . 37

6 Závěr 39

Kapitola 1

Úvod

Počítačové sítě se neustále rozšiřují a zrychlují. Spolu s tímto rozvojem stoupá potřeba zajišťovat jejich bezpečnost. Bezpečnost počítačových systémů a sítí se realizuje na mnoha úrovních, jde o rozmanitý soubor technologií a služeb. Avšak rychlost počítačových sítí se zvyšuje rychleji než výpočetní výkon procesorů. Z toho vyplývá, že nestačí pouze navrhnout řešení po funkční stránce, ale mnohdy je také třeba zabývat se do detailu implementací tak, aby výsledné řešení mělo požadované vlastnosti za co nejnižší cenu.

Nad počítačovými sítěmi je nutné realizovat řadu rozmanitých aplikací. Jednou ze základních aplikací je firewall. Jde o systém, který podle jistých pravidel rozhoduje, jestli paket systémem projde, nebo bude ignorován (zahozen). Administrátor takto může selektivně povolovat a zakazovat jisté typy komunikace, vytváří tedy paketový filtr. Další aplikací je systém zajišťující kvalitu služeb (Quality of Service - QoS). Zde je třeba rozlišit různé typy provozu a některé upřednostnit nebo naopak omezit. Také při použití virtuálních sítí (Virtual Private Network - VPN) je třeba třídit pakety podle jistých pravidel. Vyjmenované síťové aplikace musí klasifikovat pakety podle jejich parametrů a následně určit způsob jejich dalšího zpracování. Tato práce se zabývá právě technikami klasifikace paketů v počítačových sítích.

Ve všech aplikacích klasifikace paketů existují značné nároky na rychlost řešení. Tyto nároky vyplývají z rychlosti současných počítačových sítí. Osobní počítače neposkytují dostatečný výpočetní výkon pro klasifikaci na dnešních 10 Gbit/s sítích, a vzhledem ke zrychlování počítačových sítí lze předpokládat, že v budoucnu bude tento problém ještě závažnější. Proto je nutné hledat řešení, kde programové vybavení počítače (software) je nahrazeno technickým vybavením (hardware).

V odborné literatuře je popsáno mnoho metod klasifikace paketů. Některá řešení jsou vhodná pouze pro softwarovou implementaci, novější práce však již předpokládají využití hardware pro dosažení vysokého výkonu. Přístupy ke klasifikaci mohou využívat např. rozhodovací stromy [9, 12], práci s prefixy a kartézskými součiny [15, 7, 14], nebo práci s dlouhými bitovými vektory [11]. Stranou algoritmických metod stojí ternární asociativní paměti (TCAM).

Pro hardwarovou implementaci je možné použít specializované čipy typu ASIC[1] nebo FPGA[2]. Výhodou čipů ASIC je vyšší rychlost a také větší množství dostupných zdrojů na čipu. Nevýhodou je jejich pevná struktura, kterou po vyrobení již nelze změnit. Naopak

[1] Application-specific integrated circuit
[2] Field-programmable gate array

čipy FPGA jsou sice pomalejší a nabízejí menší množství dostupných zdrojů, jsou ale rekonfigurovatelné. Taková vlastnost je velice výhodná zejména při výzkumu.

Ve druhé kapitole práce je popsán hierarchický model počítačových sítí a problematika klasifikace je definována formálněji a s většími detaily. Také jsou uvedeny požadavky, které klademe na algoritmus klasifikace paketů. Třetí kapitola popisuje některé metody klasifikace paketů. Ke každé metodě je uveden popis základního funkčního principu a krátké zhodnocení z pohledu paměťové a časové náročnosti a také z pohledu vhodnosti pro implementaci v hardware. Ve čtvrté kapitole je představen nový algoritmus klasifikace paketů založený na perfektních rozptylovacích (hashovacích) funkcích. Je navržena také optimalizace algoritmu za účelem snížení paměťových nároků řešení. Pátá kapitola navrhuje implementaci algoritmu s využitím technologie FPGA. V poslední kapitole jsou shrnuty výsledky a jsou diskutovány možnosti pokračování práce.

Kapitola 2

Teoretický rozbor

2.1 Vrstvový model TCP/IP

Na současné počítačové sítě lze pohlížet jako na hierarchii čtyř vrstev. Každá vrstva řeší konkrétní problémy a používá k tomu určené protokoly. V tabulce 2.1 jsou vrstvy stručně popsány spolu s příklady používaných protokolů.

Název vrstvy	**Účel**	**Příklady protokolů**
Aplikační vrstva	Komunikace jednotlivých aplikací, služby přímo pro uživatele	HTTP, FTP, DNS
Transportní vrstva	Abstrakce nad sítí, spolehlivý přenos dat	TCP, UDP
Síťová vrstva	Směrování v síti, předávání datagramů	IP, ARP, ICMP
Vrstva síťového rozhraní	Přístup k fyzickému médiu	Ethernet, Token ring

Tabulka 2.1: Čtyři vrstvy TCP/IP síťového modelu

Tato práce se zabývá technikami pro filtrování provozu na úrovni síťové vrstvy, jde tedy o klasifikaci paketů IP protokolu. Hlediska, která určují příslušnost paketu do třídy jsou hodnoty polí z hlavičky paketu. Typickým příkladem takového pole je zdrojová IP adresa, ale obecně může jít o libovolné pole. Ačkoliv pracujeme na síťové vrstvě, často se také používají informace z transportní vrstvy, jako například typ protokolu a TCP/UDP port. Je také možné klasifikovat podle informací z vrstvy síťového rozhraní, jako například MAC adresa u Ethernetu.

2.2 Model klasifikace

Klasifikační algoritmus obsahuje 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.

2.3 Klasifikační pravidla

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

- **Hodnota**: Pole musí mít přesně shodnou hodnotu.
- **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. Prefix odpovídá adrese sítě, zatímco adresa počítače je libovolná a podmínka tak vyhovuje IP libovolného počítače dané sítě.
- **Rozsah**: Je určen souvislý rozsah hodnot, které vyhovuje pravidlu. Takové podmínky se používají zejména ve spojení s TCP/UDP porty.

Všechny uvedené možnosti lze reprezentovat pomocí prefixů. Každý rozsah lze snadno převést na několik prefixů, ilustrace je na obrázku 2.1.

0011	0011
0100	01**
0101	
0110	
0111	
1000	100*
1001	

Obrázek 2.1: Převod rozsahu ⟨3, 9⟩ na tři prefixy: 0011, 01**, 100*

Počet prefixů, které se mohou při této operaci vygenerovat, je v nejhorším případě $2N - 2$, kde N je počet bitů dané dimenze[1] [9]. A protože přesnou hodnotu v podmínce pravidla lze chápat jako prefix maximální délky (žádné bity nejsou volitelné), lze po převodu rozsahů na prefixy všechny podmínky vyjádřit jako prefixy.

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 $p \in \mathbf{N}$ je priorita a a_x jsou prefixy.

Značení: Je-li třeba pravidla číslovat, používáme dolní index $(R_1, R_2, \ldots)$. K označení priority a jednotlivých prefixů daného pravidla se používá tečková notace, takže prefix a_1 pravidla R_2 lze napsat jako $R_2.a_1$.

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

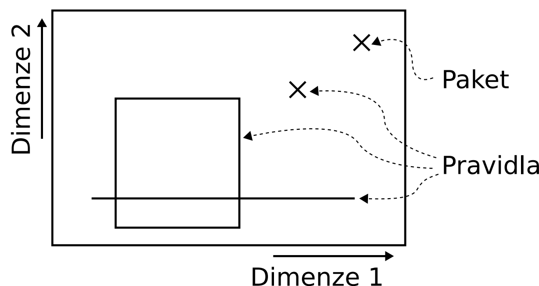
Klasifikátor je množina pravidel taková, že každá dvě pravidla mají rozdílnou prioritu.

$$K = \{R; R_i.p = R_j.p \Rightarrow i = j\} \tag{2.1}$$

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

2.4 Geometrická reprezentace klasifikace

Na klasifikaci lze také 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 2.2 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 2.2: 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.

2.5 Požadavky na klasifikační algoritmus

Klasifikační algoritmus musí splňovat jisté vlastnosti, aby byl vhodný pro použití v systémech zajišťujících počítačovou bezpečnost:

- **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 2.2 ukazuje počet přenesených paketů za sekundu pro typické přenosové rychlosti současných nebo budoucích sítí. Při výpočtu je uvažována délka paketu 64 B, což je minimum pro síť typu Ethernet.

  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[2] je důležitým ukazatelem zabraná plocha FPGA nebo ASIC čipu. Opět zde jde o cenu výsledného řešení.

- **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 unikátní hodnoty 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.

Síťová technologie [Gbit/s]	**Paketová rychlost** [paketů/s]	**Doba zpracování paketu** [ns/paket]
1	1 954 125	512
4	7 812 500	128
10	19 531 250	51,2
40	78 125 000	12,8
100	195 312 500	5,12

Tabulka 2.2: Počet přenesených paketů délky 64 B pro různé síťové rychlosti.

Tato práce předpokládá využití jednoho čipu FPGA a jedné externí statické paměti. Počet podporovaných pravidel je do jednoho tisíce. Hlavním kritériem pro návrh nového algoritmu je rychlost klasifikace.

[2] Chceme-li dosahovat opravdu vysoký výkon, je hardwarová realizace algoritmu nutností.

Kapitola 3

Přístupy ke klasifikaci

Předchozí kapitola uvedla problém klasifikace paketů a vymezila podmínky, které musí algoritmus řešící popsaný problém splňovat. V této části práce je uvedeno několik metod pro klasifikaci paketů. Zatímco první dva uvedené algoritmy jsou pouze teoretickou možností, následují již propracovanější metody, které se objevily v odborných publikacích v několika posledních letech.

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 zcela nevodný z pohledu rychlosti. Pokud by množina pravidel obsahovala tisíc prvků, bylo by nutné udělat tisíc přístupů do paměti pro klasifikaci každého paketu. Tento algoritmus však používá nejméně paměti - stačí prostě uložit do paměti všechna pravidla.

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 zcela nereálné z důvodů paměťové náročnosti. 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

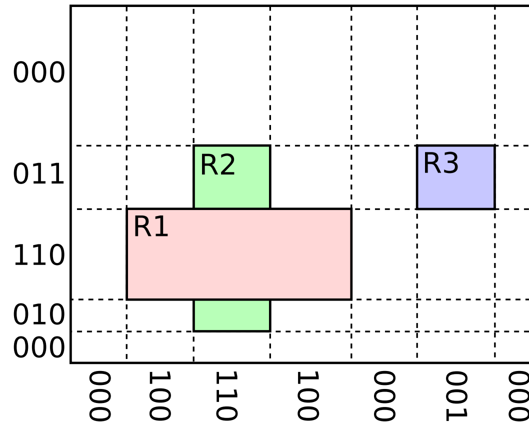
Asociativní paměť je zvláštním typem paměti, která podporuje vyhledávání podle obsahu. Vstupem je hledané slovo a výstupem je adresa, na které se slovo nachází. Paměť má u každého slova komparátory, které vyhodnocují shodu hledaného slova s uloženým. Ternární varianta asociativních pamětí (TCAM) pracuje kromě jedniček a nul ještě s třetím stavem, nazývaným *don't care*. Je-li v paměti na nějakém bitu 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 pravila a poté je vyhledávat jediným přístupem.

TCAM mají však omezenou kapacitu, poměrně velký příkon a také vysokou pořizovací cenu. I přes známé nevýhody jsou paměti TCAM často používány v komerčních zařízeních [12, 7]. Uvedené nevýhody ale také způsobují, že neustále existuje snaha nacházet a zkoumat algoritmy, které používají paměti s náhodným přístupem.

3.3 BitVector algoritmus

Algoritmus BitVector [11] 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 3.1).

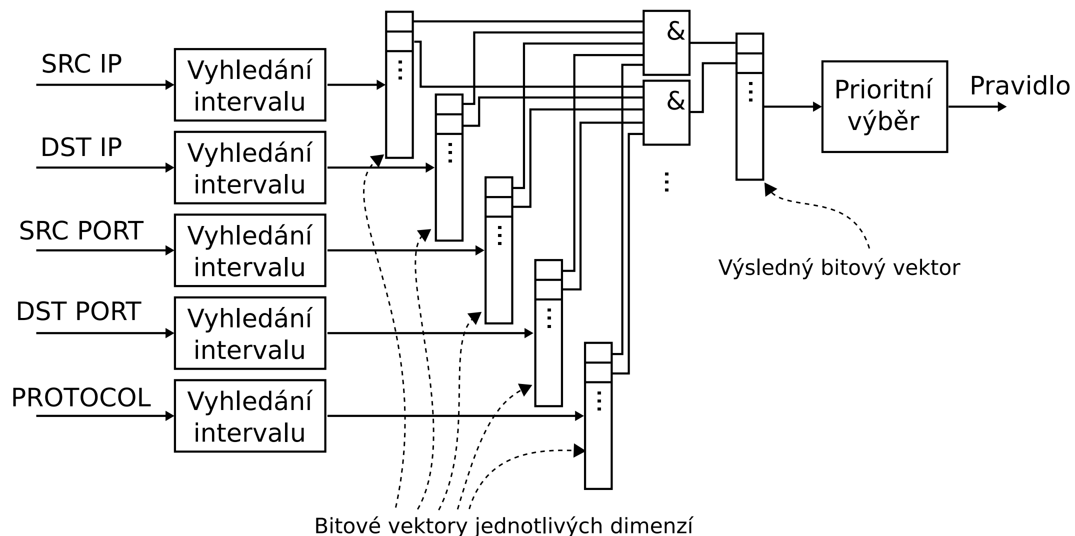


Obrázek 3.1: Výpočet vektorů pro dvě dimenze. Tři pravidla vytvořila 7 intervalů na vodorovné ose a 5 intervalů na svislé ose. Ke každému intervalu je přiřazen vektor tří bitů, protože máme tři pravidla.

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 3.2.

Byly navrženy některé úpravy tohoto algoritmu (například *Aggregated Bit Vector*[3], nebo *BV-TCAM* [13]), které dosahují lepších výsledků jak paměťové, tak časové náročnosti.

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



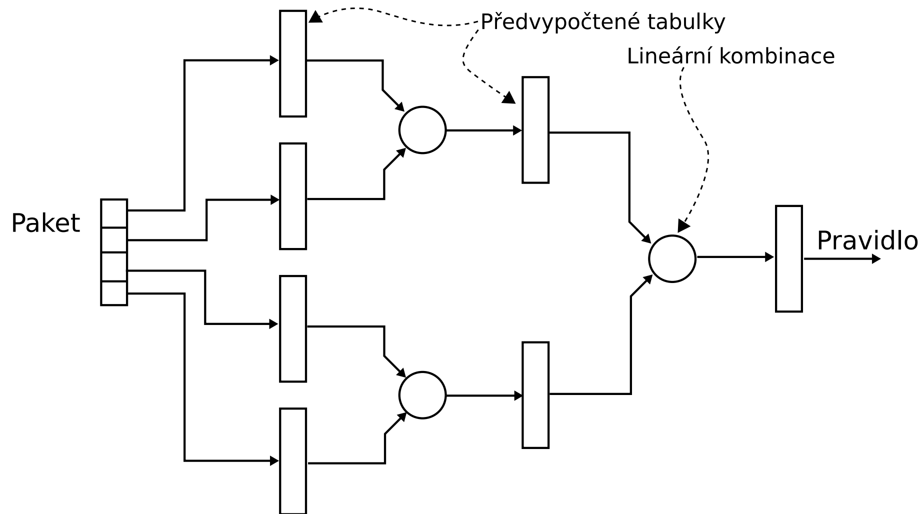
Obrázek 3.2: Schéma klasifikace algoritmem BitVector. Po vyhledání intervalu v každé dimenzi je proveden bitový součin všech vektorů a ve výsledném vektoru je nalezeno pravidlo s nejvyšší prioritou.

3.4 RFC algoritmus

Recursive Flow Classification (popsaný v [9]) 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. Velikost tabulek je výrazně menší než v případě adresování celým slovem. 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 3.3. Tento algoritmus poskytuje velice dobré výsledky z pohledu výkonnosti, ale jeho slabinou je paměťová náročnost, která je mnohokrát větší než u ostatních metod [12].

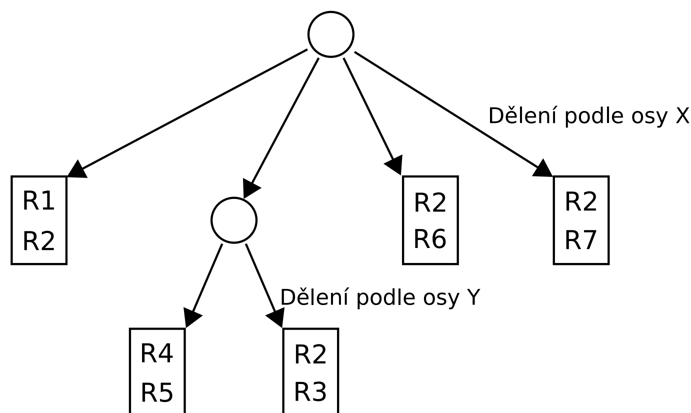


Obrázek 3.3: Schéma klasifikace algoritmem RFC. Na první úrovni jsou adresy do tabulek přímo hodnoty polí z hlavičky paketu, dále jsou výsledky kombinovány a použity jako adresy dalších tabulek. Poslední tabulka obsahuje pravidla.

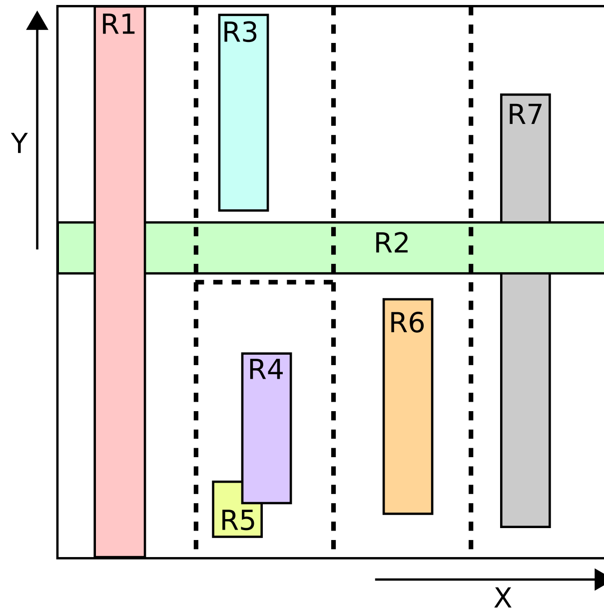
3.5 Rozhodovací stromy

Hierarchical Intelligent Cuttings (Hi-Cuts) [9] je algoritmus vycházející z geometrické reprezentace klasifikace. Spočívá v konstrukci rozhodovacího stromu. Při průchodu stromem je v každém uzlu prostor 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 3.4 ukazuje příklad jednoduchého rozhodovacího stromu.



Obrázek 3.4: 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 3.5: Plocha rozdělená podle rozhodovacího stromu z obrázku 3.4. 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.

Algoritmus Hypercuts má velice dobré výsledky jak z pohledu paměťové náročnosti, tak také nízkým počtem přístupů do paměti. Nevýhodou je, že celá datová struktura je zde uložena v jedné velké tabulce. Tím je do velké míry znemožněno paralelní zpracování – v jednom časovém okamžiku se zpracovává pouze jeden paket.

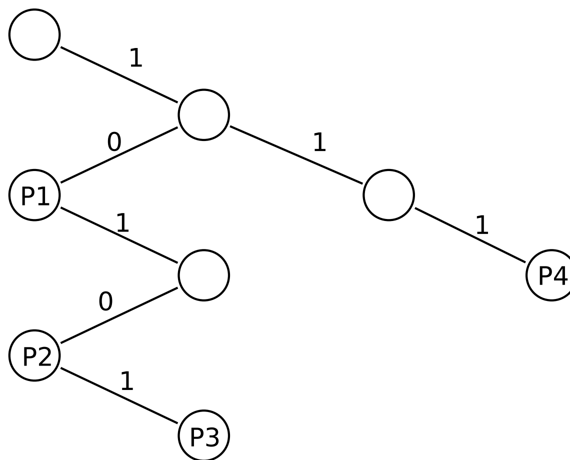
3.6 Algoritmy založené na dekompozici problému

Dekompoziční algoritmy řeší klasifikaci ve dvou základních krocích. 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*. Některé algoritmy používají datovou strukturu nazvanou Bloomův filtr, proto je jedna část kapitoly věnována tomuto tématu.

3.6.1 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 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, kterým podstromem se bude postupovat. V každém uzlu je označeno, zda zatím zpracované bity tvoří platný prefix. Pokud ano, je zde také uloženo číslo tohoto prefixu pro identifikaci. V průběhu výpočtu 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 jsme již zpracovali všechny bity. Časová složitost vyhledání je lineární s počtem bitů datového slova. Jednoduchý příklad struktury trie je na obrázku 3.6.



Obrázek 3.6: 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.

Dále je popsán Tree Bitmap (publikovaný v [8]) jako příklad algoritmu, který:

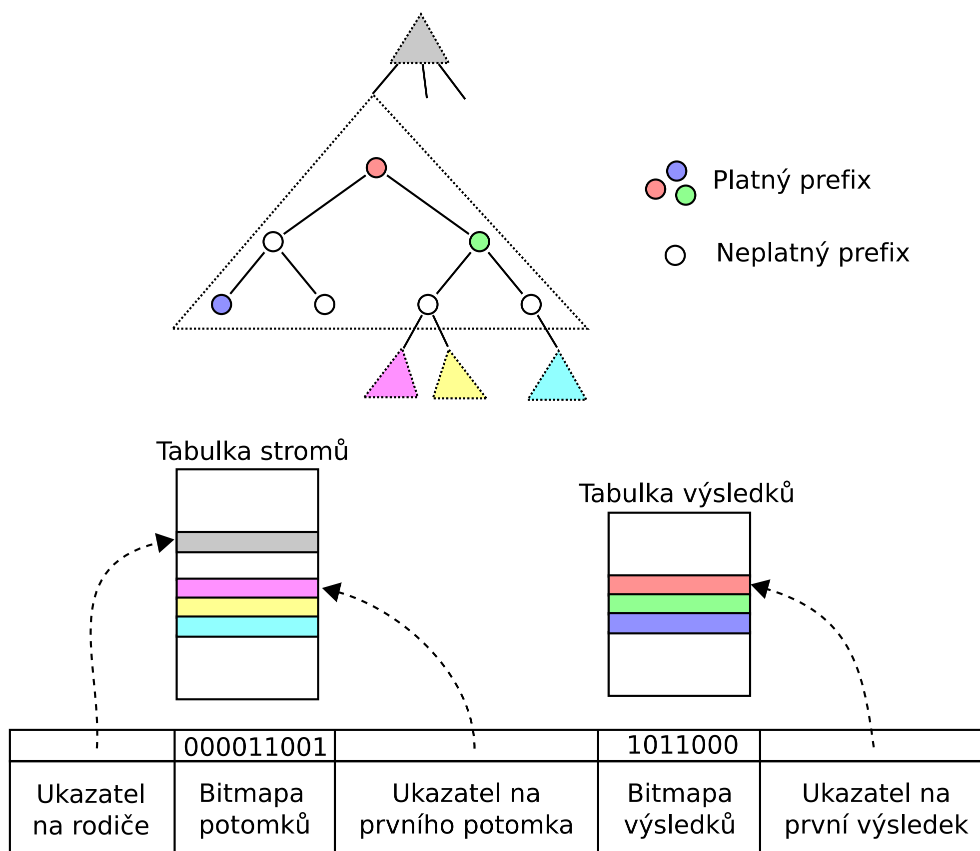
- je paměťově nenáročný.
- lze snadno realizovat hardwarově.
- lze parametrizovat nastavením arity uzlů.

Uvažujeme-li aritu uzlů trie rovnou mocnině dvojky, pak každý uzel reprezentuje binární podstrom jisté hloubky. Jedním uzlem datové struktury Tree Bitmap je právě takový podstrom. Protože jeden uzel Tree Bitmap reprezentuje několik uzlů jednoduché trie, může mít několik následníků – podstromů. Tree Bitmap předpokládá, že všichni následníci daného uzlu jsou v paměti umístěni seřazeni za sebou. Potom v každém uzlu je uložena pouze adresa prvního následníka. Nicméně uzly, které neobsahují žádné prefixy, ani nemají žádné následníky do paměti zbytečně neukládáme. Proto je v každém uzlu také uložena bitmapa, která obsahuje jeden bit pro každý možný podstrom. Bit je nastaven do jedničky, pokud odpovídající podstrom existuje, v opačném případě je v bitmapě uložena nula. Informace

o adrese prvního podstromu spolu s bitmapou stačí k určení adresy všech přímých podstromů daného uzlu.

Uzel také musí obsahovat čísla prefixů, které v něm končí. Podobně jako podstromy, také čísla prefixů jsou uložena v paměti za sebou. Uzel opět obsahuje pouze ukazatel na první výsledek, spolu s bitmapou platných prefixů.

Datovou strukturu lze v případě potřeby ještě rozšířit o zpětné ukazatele. S jejich pomocí lze procházet strom nejen shora dolů, ale také v opačném směru. Struktura Tree Bitmap rozšířená o zpětné ukazatele je na obrázku 3.7.



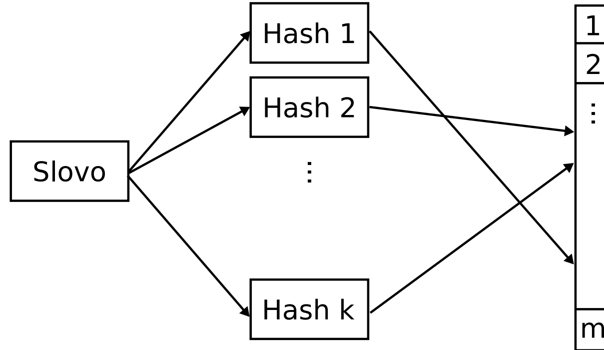
Obrázek 3.7: Modifikovaná datová struktura Tree Bitmap. Zobrazený podstrom obsahuje tři prefixy a má tři přímé následníky.

3.6.2 Bloomův filtr

Další datovou strukturou, kterou používají některé z následujících algoritmů, je *Bloomův filtr* [4]. Jedná se o způsob, jak ukládat slova do paměti velice efektivně, avšak za cenu toho, že následně je možné provádět pouze operaci přidání dalšího slova a zjištění přítomnosti daného slova. Není možné získat z paměti všechna uložená slova, stejně jako není v základní verzi algoritmu možné slova z paměti odebírat[1]. Navíc zde existuje pravděpodobnost chyby, kdy datová struktura při dotazu na existenci slova chybně vrátí kladnou odpověď.

[1]Takzvaný *počítaný Bloomův filtr* však umožňuje také mazání slov z paměti.

Bloomův filtr využívá množinu k různých hashovacích funkcí, kdy vstupem všech je hledané nebo vkládané slovo. Výstupem každé funkce je ukazatel do m-bitového pole. Je-li struktura prázdná, jsou všechny bity tohoto pole nastaveny do nuly. Přidáváme-li slovo, potom všechny bity, na které ukazuje alespoň jedna z k hashovacích funkcí, nastavíme do jedničky. Při dotazu na přítomnost slova ve struktuře potom zjišťujeme, zda všechny bity, na které ukazují hashovací funkce, jsou nastaveny do jedničky. Schéma výpočtu hashovacích funkcí jako indexů do bitového pole je na obrázku 3.8.



Obrázek 3.8: Schéma Bloomova filtru. k různých hashovacích funkcí paralelně počítá adresy do m-bitového pole.

Chyba může vzniknout v případě, že do paměti byla uložena taková kombinace slov, která nastaví do jedničky všechny bity, které jsou cíle hashovacích funkcí jiného slova. Ačkoliv jsme toto slovo do paměti neuložili, dotaz na jeho přítomnost vrátí pozitivní výsledek. Pravděpodobnost chyby po předchozím vložení n slov je

$$p = \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^{k}. \tag{3.1}$$

Je tedy vidět, že pravděpodobnost chyby klesá s velikostí bitového pole a stoupá s počtem již vložených slov. Pro dané m a n lze přibližně určit ideální k jako

$$k_i = 0,7\frac{m}{n}. \tag{3.2}$$

Potom je pravděpodobnost chyby přibližně

$$p_i = 0,6185^{\frac{m}{n}}. \tag{3.3}$$

3.6.3 Pseudopravidla

Pojem *pseudopravidlo*, zavedený v [7], označuje pravidla, která je nutné přidat do množiny pravidel tak, aby každá platná kombinace LPM byla pokryta. Vznik pseudopravidel demonstrujme na příkladě klasifikace podle dvou polí. Tabulka 3.1 obsahuje tři jednoduchá pravidla.

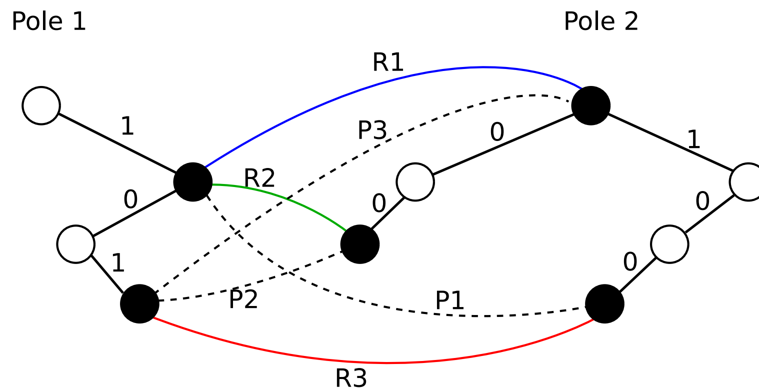
Na obrázku 3.9 jsou potom naznačeny trie pro LPM v obou dimenzích. Vyplněné uzly značí platné prefixy. Barevnými čarami jsou označeny kombinace vstupů, které byly definovány v původní množině pravidel. Čárkovaně jsou vyznačena pseudopravidla. Jde o doplnění množiny pravidel tak, aby i specifičtější výsledky LPM vracely správné pravidlo.

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

Tabulka 3.1: Původní pravidla

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

Tabulka 3.2: Přidaná pseudopravidla



Obrázek 3.9: Reprezentace pravidel a pseudopravidel (čárkovaně) pomocí dvou trie. Pseudopravidla jsou specifičtějšími případy pravidel.

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.

3.6.4 Naivní algoritmus kartézského součinu

Generovaná pseudopravidla připomínají kartézský součin, jelikož je nutné pro každé pravidlo pokrýt všechny specifičtější prefixy ve všech polích.

Autoři článku [7] zkoumali počet pseudopravidel, která vzniknou z množin pravidel použitých v několika reálných aplikacích. V citované práci je uvedeno, že pseudopravidel je *dvěstěkrát* víc než původních pravidel. Takový nárůst je nepříjemný z důvodu nároků na paměť. Nicméně pro menší množiny pravidel se lze smířit s větší paměťovou náročností.

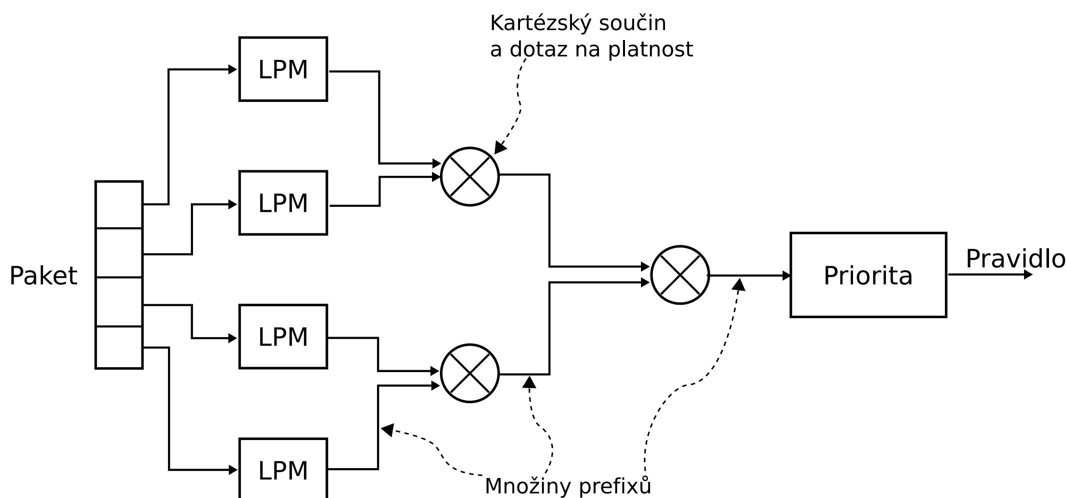
Potom lze klasifikovat pakety tak, že se nejprve nalezne LPM pro každé pole a výsledky (tedy prefixy) se spojí do jednoho datového slova [14]. Toto slovo je vstupem hashovací funkce. Výstupem je potom adresa do tabulky, ve které jsou uložena jak pravidla, tak pseudopravidla. Časová složitost takového dohledání se zdá být konstantní - stačí jediný výpočet hashovací funkce a jeden následný přístup do paměti. Z důvodů kolize hashovací funkce je však nutné, aby v položky obsahovaly navíc ukazatel na další prvek. Vznikne tak mnoho lineárních seznamů pravidel a pseudopravidel a může se zvýšit nutný počet přístupů do tabulky z důvodu procházení lineárního seznamu. Čím větší je paměť, tím menší je šance na kolizi hashovací funkce (ideální hashovací funkce má rovnoměrné rozložení).

První nevýhodou popsaného algoritmu je paměťová náročnost, protože je nutné uložit zhruba dvěstěkrát více pravidel, než kolik jich klasifikátor původně obsahoval. Další komplikací je nutnost udržování lineárních seznamů, která komplikuje výpočet[2] a zhoršuje časovou složitost až na lineární.

3.6.5 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. Taková operace je vlastně 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. 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 3.10. Výsledkem posledního stupně je množina pravidel odpovídajících danému paketu.



Obrázek 3.10: 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

[2] V případě hardwarové implementace to znamená zvětšení plochy na čipu.

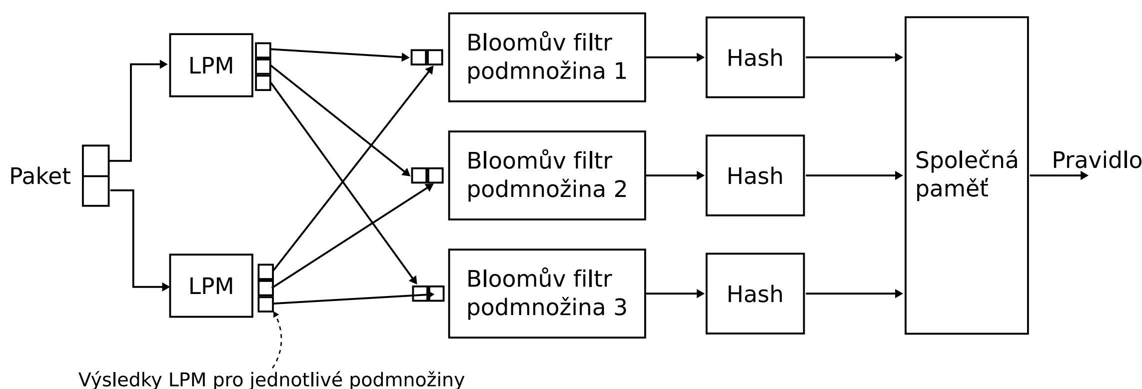
těchto operací naráz. Navíc použití Bloomových filtrů přináší možnost chyby, kterou je nutné v pozdějších fázích výpočtu eliminovat.

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

3.6.6 Rozdělení množiny pravidel na podmnožiny

Algoritmus popsaný v [7] 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 zvětšil 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 3.11.



Obrázek 3.11: Schéma algoritmu založeného na rozdělení množiny pravidel do podmnožin. (Zobrazena klasifikace podle dvou polí, tři podmnožiny pravidel)

Také algoritmus využívající podmnožiny pravidel 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.

Kapitola 4

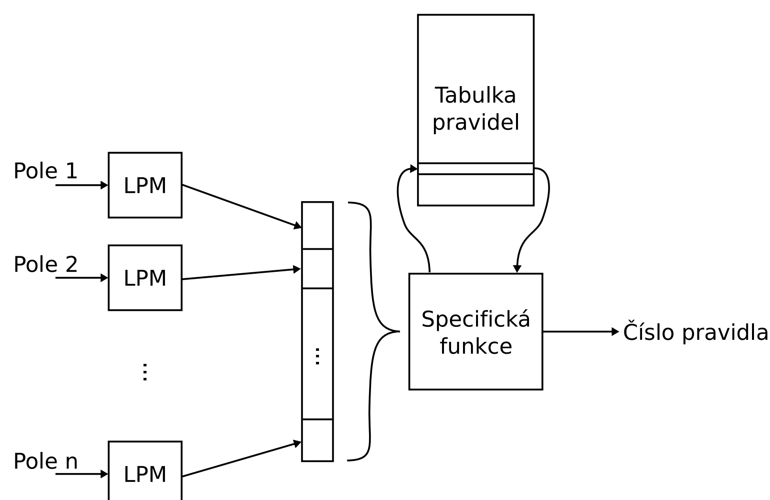
Navržený algoritmus

V předchozí kapitole byla největší část věnována klasifikačním algoritmům založeným na dekompozici problému. Dekompoziční metody přinášejí několik výhod. V první řadě je to dobrá podpora paralelního zpracování, protože operace vyhledání nejdelšího shodného prefixu jsou na sobě nezávislé. Odborné práce z posledních let ukazují značný potenciál pro další výzkum v oblasti klasifikačních algoritmů založených na dekompozici.

Dekompoziční metody se skládají z několika kroků:

- Vyhledání nejdelšího shodného prefixu pro všechny dimenze. Operace LPM jsou na sobě navzájem nezávislé, proto mohou probíhat paralelně. Algoritmy hledání nejdelšího shodného prefixu jsou v literatuře velmi dobře popsány. Pro malá pole (např. 8 bitů) lze hledání nahradit přístupem do připravené tabulky.
- Konkatenace jednotlivých výsledků do jediného datového slova.
- Hledání, zda vzniklé datové slovo odpovídá nějakému pravidlu nebo pseudopravidlu. Poslední část je specifická pro každý klasifikační algoritmus.

Kapitola 4 navrhuje nový algoritmus vycházející z popsaného modelu. Obrázek 4.1 ukazuje typickou strukturu klasifikačního algoritmu založeného na dekompozici problému.



Obrázek 4.1: Model výpočtu dekompozičních metod.

Jak bylo uvedeno výše, prvním krokem v dekompozičních metodách je operace vyhledání nejdelšího shodného prefixu ve všech polích. Prefixům jsou přidělena čísla podle nějakého klíče, například vzestupně podle výskytu v množině pravidel. Čísla jsou reprezentována jistým počtem bitů a je možné provést operaci konkatenace (zřetězení) a vytvořit jedno široké datové slovo. Vzniklé slovo může odpovídat nějakému pravidlu, pseudopravidlu, nebo nemusí odpovídat žádnému pravidlu ani pseudopravidlu. Taková situace nastane v případě, že paket neodpovídá žádnému pravidlu. Také pravidla jsou očíslována, například podle priority. Úkolem klasifikačního algoritmu je zjistit číslo pravidla, kterému paket odpovídá, nebo oznámit, že paket neodpovídá žádnému pravidlu.

Navrhovaný algoritmus řeší popsaný problém tak, že předem zkonstruuje funkci, která zobrazuje každé pravidlo R a všechna jeho asociovaná pseudopravidla právě na číslo pravidla R. K nalezení požadované funkce se použije algoritmus perfektního hashování, popsaný v následující části práce.

Pakety, které nevyhovují žádnému pravidlu, se zobrazí na libovolné číslo a taková situace je ošetřena dodatečnou kontrolou. Pokud je v množině pravidel *univerzální pravidlo*, tedy pravidlo pokrývající všechny možné pakety, potom každý paket odpovídá nějakému pravidlu. Toto pravidlo ale generuje největší množství pseudopravidel, proto jej z množiny pravidel odstraňujeme a jeho vliv zajistíme v pozdější fázi klasifikace.

4.1 Perfektní hashovací funkce

V navrženém algoritmu se využívá metoda pro nalezení bezkolizní (perfektní) hashovací funkce za předpokladu, že je předem známá množina klíčů. Využíváme metodu popsanou v [6]. Jejím principem je sestrojení neorientovaného grafu, kde každá hrana odpovídá jednomu vstupnímu slovu a každý uzel odpovídá výsledkům dvou různých hashovacích funkcí. Činnost algoritmu lze popsat v následujících krocích:

1. Vstup: K klíčů, ke každému klíči je přiřazeno číslo, které má být výsledkem hashovací funkce. (Na toto číslo se klíč hashuje.)

2. Vytvoř graf s $N = c \times K$ vrcholy, kde $c > 1$.

3. Zvol dvě různé hashovací funkce $f1, f2$ s obory hodnot $< 0, N - 1 >$.

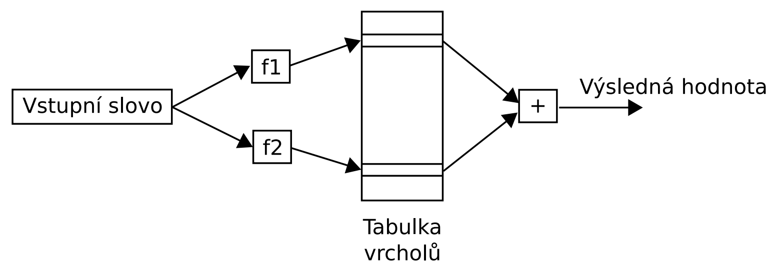
4. Pro každý klíč k vypočti $h1 = f1(k), h2 = f2(k)$, vytvoř neorientovanou hranu spojující vrcholy $h1$ a $h2$ a ohodnoť tuto hranu požadovanou výstupní hodnotou přiřazenou ke k.

5. Zjisti zda je graf acyklický. Pokud ne, zvětši c a pokračuj krokem 2.

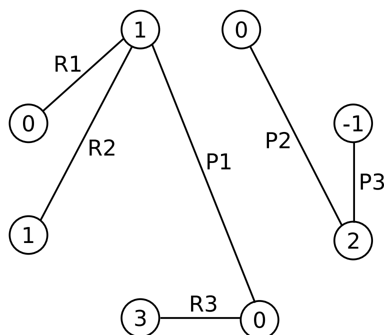
6. Každý vrchol ohodnoť tak, aby pro každou hranu platilo, že její ohodnocení je součtem ohodnocení vrcholů, které spojuje. Díky tomu, že graf je acyklický, je možné toho dosáhnout prostým prohledáním grafu do hloubky, se začátkem v libovolném uzlu.

7. Funkce $f1, f1$ a tabulka ohodnocení vrcholů jsou výsledky hledání hashovací funkce.

Samotný výpočet hashovací funkce je poté velice jednoduchý. Stačí nad vstupním slovem vypočíst hodnoty obou hashovacích funkcí $f1, f2$, z tabulky zjistit ohodnocení obou vrcholů a obě ohodnocení sečíst. Výpočet je znázorněn na obrázku 4.2. Každý vrchol je reprezentován jedním celým číslem se znaménkem, počet bitů závisí na rozsahu požadovaných výstupních hodnot. Na obrázku 4.3 je příklad výsledného grafu.



Obrázek 4.2: Výpočet perfektní hashovací funkce.



Obrázek 4.3: Příklad acyklického grafu vzniklého při hledání hashovací funkce.

4.2 Základní schéma výpočtu

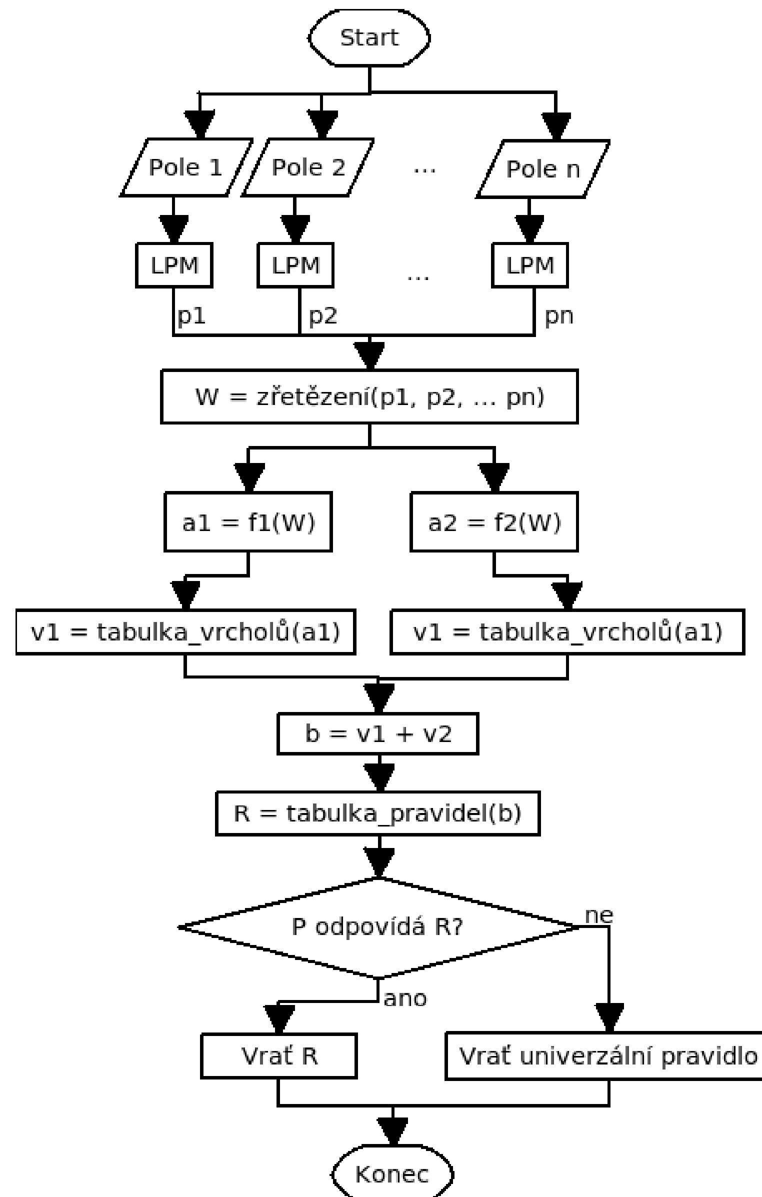
Pro účely klasifikace paketů jsou vstupními slovy perfektní hashovací funkce pravidla a pseudopravidla ve formě zřetězených výstupů operací vyhledání nejdelšího shodného prefixu. Požadovaný výstup pro každé pravidlo a pseudopravidlo potom je správné číslo pravidla. Je tak nalezena funkce, která pravidlo a všechna jeho pseudopravidla zobrazí na číslo tohoto pravidla. Tato funkce tedy není bezkolizní, naopak jsou zavedeny záměrné kolize právě tak, jak jsou potřeba pro danou množinu pravidel. Pseudopravidla nejsou v navrženém algoritmu nikde uložena, používají se pouze ke konstrukci hashovací funkce.

Klasifikace paketů podle navrženého algoritmu se skládá z následujících kroků:

1. Vstup: Hodnoty extrahované z hlavičky paketu.
2. Pro každé pole je provedena operace vyhledání nejdelšího shodného prefixu. Operace LPM jsou navzájem nezávislé a mohou tedy probíhat paralelně.
3. Výsledky jsou zřetězeny do jednoho datového slova. V hardwaru jde o triviální operaci.
4. Nad vzniklým datovým slovem jsou vypočteny dvě různé hashovací funkce. Hashovací funkce by měly být kvalitní, tedy měly by mít rovnoměrné rozložení. Příkladem takové kvalitní funkce je algoritmus CRC. Pro získání dvou různých funkcí ze stejného algoritmu stačí před samotným výpočtem funkce přičíst ke vstupním slovům dvě různé konstanty.
5. Z tabulky vrcholů jsou vyčtena ohodnocení vrcholů.

6. Ohodnocení jsou sečtena, tím se získá číslo pravidla.
7. Z tabulky pravidel je vyčteno pravidlo a porovnáno se skutečnými hodnotami v hlavičce paketu. Pokud je porovnání úspěšné, je výsledkem klasifikace toto pravidlo. V opačném případě je výsledkem univerzální pravidlo, pokud takové existuje v množině pravidel.

Vývojový diagram algoritmu je na obrázku 4.4.



Obrázek 4.4: Základní schéma navrženého algoritmu klasifikace paketů.

Je vidět, že algoritmus neobsahuje zpětnou vazbu je tedy možné snadno využít princip zřetězené linky. Všechny dílčí části algoritmu je možné poměrně snadno realizovat v hardware.

4.3 Analýza časové a paměťové složitosti

Pro analýzu paměťových nároků jsou použity čtyři reálné množiny pravidel z univerzitní sítě (fw1..fw4), spolu se šesti množinami vygenerovanými pomocí nástroje ClassBench [17] (synth1..synth6). Základní vlastnosti použitých množin jsou v tabulce 4.1.

Množina	Pravidel	Zdrojová adresa	Cílová adresa	Zdrojový port	Cílový port	Protokol
fw1	32	13	2	10	22	4
fw2	58	26	24	4	1	2
fw3	103	28	48	36	1	4
fw4	171	84	84	1	6	3
synth1	40	12	19	10	22	5
synth2	49	35	41	8	22	3
synth3	49	26	14	14	1	4
synth4	70	27	62	1	48	3
synth5	82	20	37	3	3	4
synth6	100	73	85	1	54	4

Tabulka 4.1: Počty unikátních prefixů v jednotlivých dimenzích.

Analýzu paměťových nároků lze rozdělit na tři části:

- **Datové struktury pro operace vyhledání nejdelšího shodného prefixu.** Naše experimenty (viz. tabulka 4.1), podobně jako analýzy v článcích o klasifikaci paketů ukazují, že počet unikátních hodnot v jednotlivých dimenzích je většinou poměrně malý. Díky tomu je možné využít paměti na čipu.

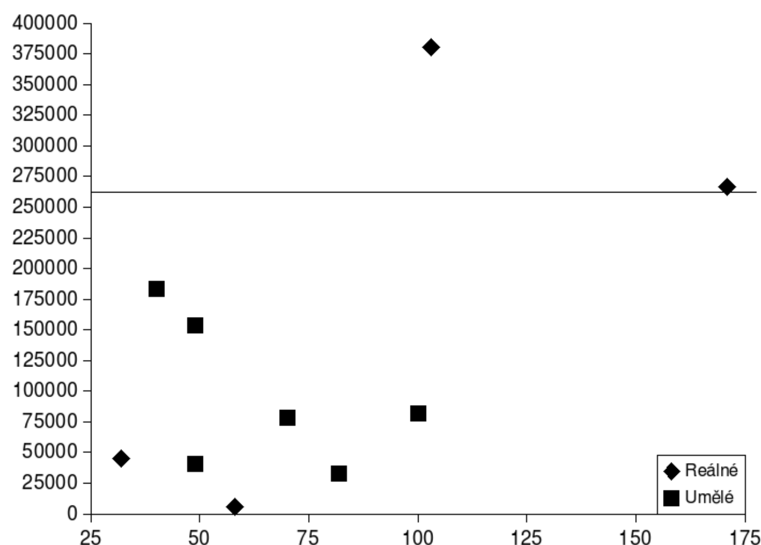
- **Tabulka pravidel.** Tato práce uvažuje množiny řádově do tisíce pravidel. Takové množství lze také snadno uložit v blokových pamětech čipu FPGA.

- **Tabulka vrcholů.** Počet vrcholů acyklického grafu musí být větší než počet jeho hran. Nicméně algoritmus nalezení perfektní hashovací funkce potřebuje často poněkud větší režii, experimenty ukazují, že potřebný počet vrcholů je zhruba dvojnásobný oproti počtu pseudopravidel (tabulka 4.2, graf 4.5). Z grafu je vidět, že počet vrcholů si vyžaduje použití větší paměti, než kolik je k dispozici uvnitř FPGA čipu. Proto navržený algoritmus předpokládá použití jedné externí statické paměti.

Při analýze časové složitosti algoritmu lze vycházet z vývojového diagramu uvedeného výše. Diagram neobsahuje zpětnou vazbu, proto při výpočtu nevzniká žádný cyklus. Pro vyhledání nejdelšího shodného prefixu předpokládáme použití algoritmu Tree Bitmap. Uvedený algoritmus pro LPM má lineární časovou složitost s počtem bitů vstupního slova. Protože je však počet bitů každého pole z hlavičky paketu předem známý, lze vyhledání nejdelšího shodného prefixu paralelizovat tak, aby operace LPM nezpůsobovala zpomalení celého algoritmu. Doba hledání je zhora omezena konstantou, operaci LPM lze tedy považovat za operaci s konstantní časovou složitostí. Podobně doba výpočtu hashovací

Množina	Pravidel	Pseudopravidel	Vrcholů	c
fw1	32	22 467	44 998	2
fw1	58	2 942	6 000	2
fw3	103	165 301	380 429	2,3
fw4	171	126 709	266 448	2,1
synth1	40	83 491	183 768	2,2
synth2	49	69 947	153 991	2,2
synth3	49	20 335	40 670	2
synth4	70	37 457	78 806	2,1
synth5	82	16 274	32 710	2
synth6	100	37 242	81 932	2,3

Tabulka 4.2: Expanze pseudopravidel a režie perfektní hashovací funkce.



Obrázek 4.5: Počty vrcholů pro různé množiny pravidel. Vodorovná čára označuje počet 18 bit slov v blokových pamětech FPGA Virtex-5 LX110 [2].

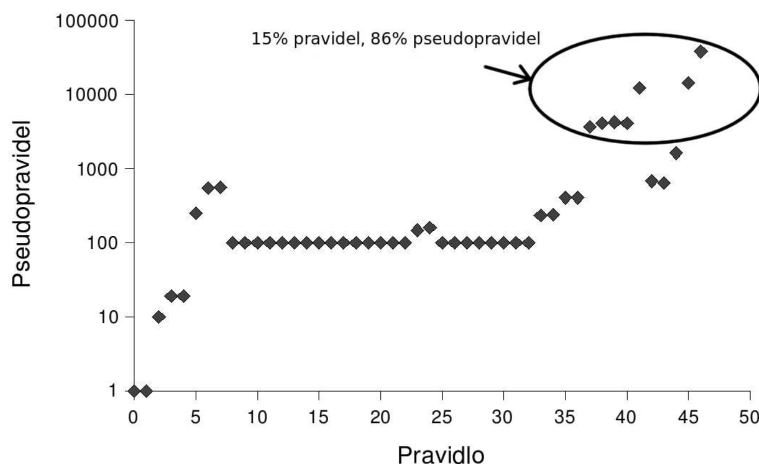
funkce závisí na velikosti vstupu, ale protože je velikost vstupu konstantní a předem známá, lze považovat dobu výpočtu za konstantní. Ostatní části celého algoritmu klasifikace jsou zřetězení, součet, přístupy do tabulek a porovnání paketu proti pravidlu. Vyjmenované operace jsou snadno realizovatelné s konstantní časovou složitostí. Z toho vyplývá, že byl nalezen algoritmus klasifikace paketů s konstantní časovou složitostí.

Za předpokladu, že všechny části algoritmu, které jsou uvnitř čipu, jsou implementovány jako zřetězená linka a v případě potřeby paralelizovány, je výkon algoritmu určen rychlostí externí statické paměti. Pro každý paket je nutné přistoupit do této paměti právě dvakrát (vyčíst dva vrcholy). Jako příklad lze uvést paměť typu QDR–II od firmy Cypress [1]. Datová sběrnice má šířku 18 bit, frekvence je 250 MHz DDR, vždy s dávkou délky 2. Jeden

přístup do paměti tedy trvá 1 takt (4 ns) a je vyčteno 36 bitů. Protože algoritmus musí přistoupit na 2 různá místa v paměti pro každý paket, je sběrnice vytížena po dobu 8 ns. Z toho plyne, že lze klasifikovat 125 milionů paketů za sekundu. Pro pakety délky 64 B je potom rychlost zpracování 64 Gbit/s.

4.4 Redukce paměťové náročnosti

Uvedený algoritmus má nevýhodu v množství potřebných paměťových zdrojů. Pro vylepšení této vlastnosti je nutné zkoumat příčiny a charakteristiky vzniku pseudopravidel. Obrázek 4.6 ukazuje příklad, kolik pseudopravidel se generuje z jednotlivých pravidel. Je vidět, že většina pravidel generuje pouze relativně malé množství pseudopravidel. Existuje však několik pravidel, z nichž vzniká enormní množství pseudopravidel. Typicky jsou to obecná pravidla s nízkou prioritou. Právě proto, že jde o obecná pravidla, existuje velmi mnoho jejich speciálních případů – pseudopravidel, což je *přímá příčina* vzniku pseudopravidel.



Obrázek 4.6: Počet vygenerovaných pseudopravidel z jednotlivých pravidel. Pravidla jsou na vodorovné ose seřazena podle priority. Svislá osa má logaritmické měřítko.

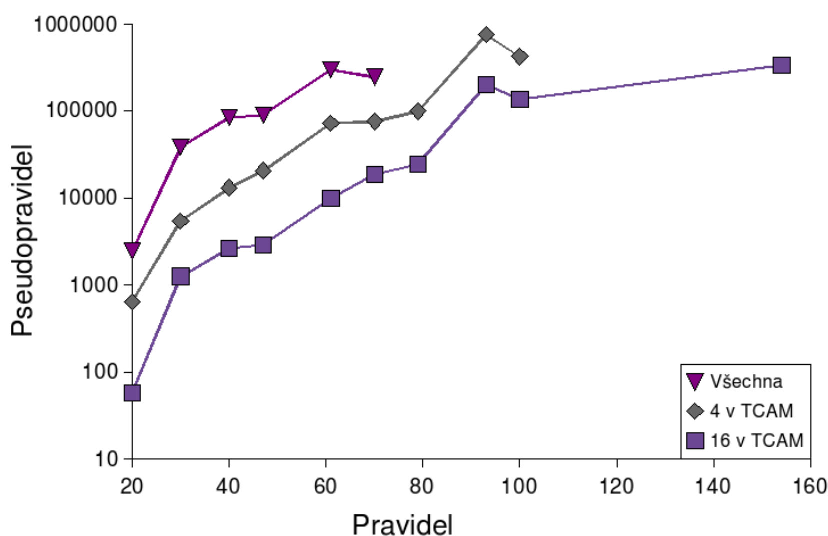
Existuje však také *nepřímá příčina* vzniku pseudopravidel. Zde jsou důležité počty unikátních prefixů v jednotlivých dimenzích. Jedná se zejména o porty, kde díky převodu rozsahů na prefixy může jediné pravidlo přidat několik unikátních prefixů. Potom všechna pravidla, která pokrývají celý rozsah, musí vygenerovat pseudopravidla pokrývající jednotlivé prefixy.

Pokud například jedno pravidlo definuje rozsah portů $< 1024; 65\,535 >$, potom je zadaný rozsah převeden na 6 prefixů. Pokud by všechna ostatní pravidla definovala úplný rozsah portů, tedy $< 0; 65\,535 >$, potom se pro každé takové pravidlo musí vygenerovat nejméně šest pseudopravidel, aby byly pokryty všechny prefixy. A protože toto platí pro každou dimenzi, může narůstat počet pseudopravidel velmi významně.

Pokud dokážeme určit malé množství pravidel, která způsobují velký nárůst paměťových nároků algoritmu, potom můžeme tato pravidla odstranit z množiny pravidel. Odstraněná pravidla mohou být umístěna v malé asociativní paměti přímo na čipu. Tím je možné ušetřit významné množství paměti v tabulce vrcholů.

Problémem zůstává algoritmus nalezení těch několika pravidel, která způsobují vznik nejvíce pseudopravidel. Nejpřesnější, ale časově velice náročnou metodou je vyzkoušení všech možností, tedy postupné odebírání jednotlivých pravidel a vyhodnocování počtu pseudopravidel.

Jednoduchou metodou je odebrání těch pravidel, k nimž je asociováno největší množství pseudopravidel. Jde sice o poměrně efektivní způsob, ale nedokáže identifikovat pravidla, která způsobují expanzi pseudopravidel tím, že zvětšují počty unikátních prefixů v jednotlivých dimenzích. Bere tedy ohled pouze na přímou, ale už ne na nepřímou příčinu vzniku pseudopravidel. Na obrázku 4.7 je vidět, kolik pseudopravidel lze ušetřit odebráním několika málo pravidel. Pří výběru pravidel k odebrání se bral ohled pouze na přímou příčinu vzniku pseudopravidel. Je zřejmé, že tímto způsobem lze počet pseudopravidel snadno řádově snížit.



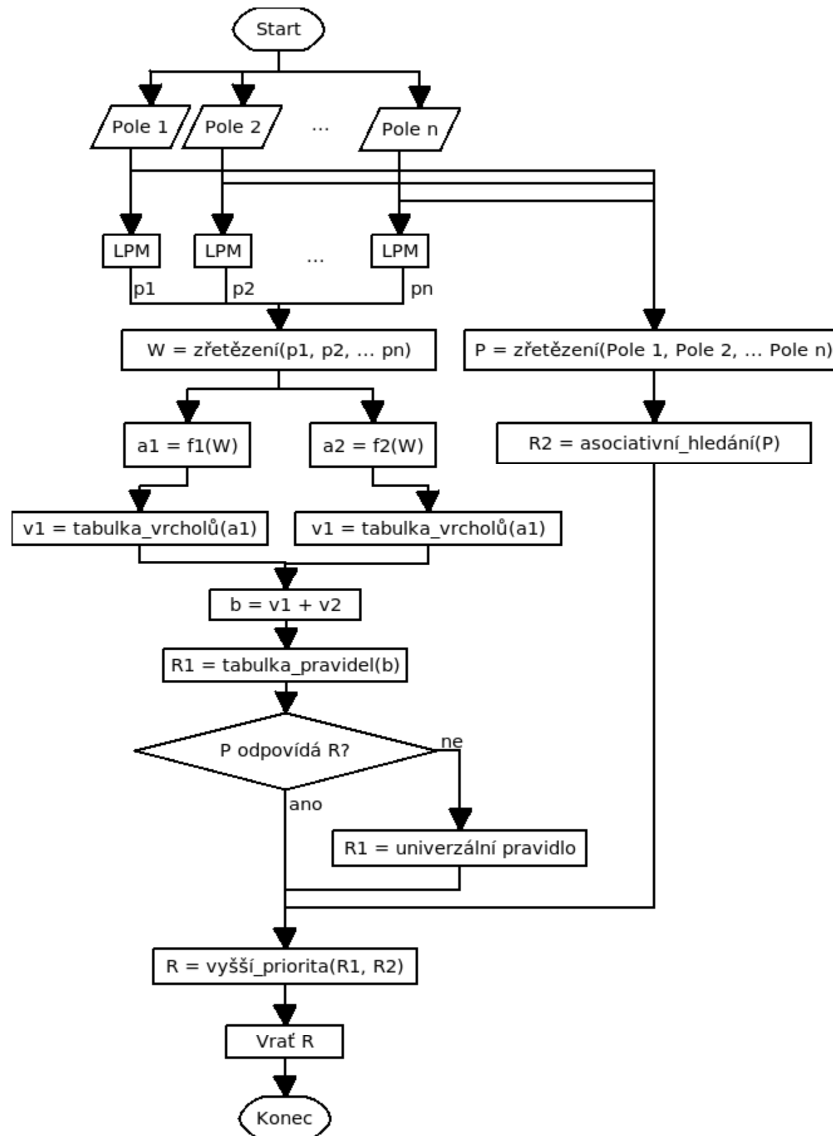
Obrázek 4.7: Počty pseudopravidel v případě, že se vygenerují všechna, nebo se odstraní 4 nebo 16 pravidel, která generovala největší množství pseudopravidel. Svislá osa má logaritmické měřítko.

4.5 Algoritmus s paměťovou optimalizací

Optimalizace využívá malou asociativní paměť na čipu. Do této paměti jsou umístěna pravidla, která generovala nejvíce pseudopravidel. Přistupuje se do ní paralelně s ostatními kroky algoritmu. Před koncem algoritmu se porovnají výsledky obou větví algoritmu a výsledek s vyšší prioritou se použije jako výstup algoritmu. Vývojový diagram je na obrázku 4.8.

4.6 Analýza optimalizovaného algoritmu

Časové vlastnosti algoritmu zůstávají stejné jako v předchozím případě. Přibyl krok vyhledávání v asociativní paměti, který ale lze provádět paralelně s ostatními úlohami, takže neovlivňuje výkonnost algoritmu. Díky této asociativní paměti spotřebuje hardwarová implementace algoritmu větší plochu na čipu. Na obrázku 4.9 je obvod provádějící porovnání

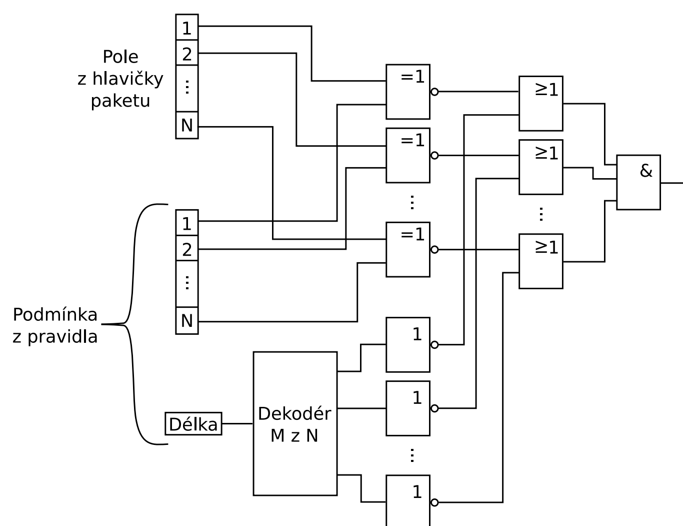


Obrázek 4.8: Schéma navrženého algoritmu klasifikace paketů s paměťovou optimalizací.

jednoho pole z hlavičky paketu. Takové porovnání je nutné provést pro všechna pole. Analýza zdrojů na čipu je uvedena v další kapitole. Před koncem je ještě navíc porovnání výsledků z obou větví a výběr vyšší priority, ale tento krok je snadný a je možné využít zřetězenou linku k překrytí jeho doby.

Zajímavé je porovnání paměťových nároků základní a vylepšené verze tohoto algoritmu. Tabulka 4.3 ukazuje jak se snížil počet pseudopravidel po odstranění 16 pravidel, ke kterým bylo připojeno nejvíc pseudopravidel. Při výběru pseudopravidel k odstranění tedy byl brán ohled pouze na přímou příčinu vzniku pseudopravidel. Nepřímá příčina (počty unikátních prefixů) nebyla zohledněna, proto výsledky nemusí být nejlepší možné.

Z posledního sloupce tabulky je vidět, že i v nejhorším případě se počet pseudopravidel snížil téměř třikrát.



Obrázek 4.9: Porovnání jednoho N-bitového pole z hlavičky paketu s hodnotou uloženou v pravidlu.

Množina	**Pravidel**	**Pseudopravidel**	**Pseudopravidel***	**k**
fw1	32	22 467	169	132,94
fw1	58	2 942	669	4,40
fw3	103	165 301	55 966	2,95
fw4	171	126 709	24 467	5,18
synth1	40	83 491	582	143,46
synth2	49	69 947	1 607	43,53
synth3	49	20 335	1 038	19,59
synth4	70	37 457	530	70,67
synth5	82	16 274	4 952	3,29
synth6	100	37 242	869	42,86

Tabulka 4.3: Snížení počtu pseudopravidel po odstranění 16 pravidel (sloupec s hvězdičkou)

Kapitola 5

Implementace

Při implementaci je nutné brát ohled zejména na použitou technologii. Hlavními požadavky jsou dosažitelná frekvence a množství zabraných zdrojů na čipu FPGA. Nedosahuje-li některá část algoritmu potřebnou rychlost zpracování, je nutné takovou část paralelizovat, čímž se zvyšuje množství zabraných zdrojů. Celý výpočet musí využívat princip zřetězené linky. Důležitým parametrem je také celková latence, tedy doba od vstupu extrahovaných polí z hlavičky paketu do systému až do výstupu čísla použitého pravidla. Díky použití zřetězené linky sice celková latence neovlivňuje propustnost řešení, ale má vliv například na velikost vyrovnávací paměti na pakety. Pakety je totiž nutné dočasně uložit (bufferovat) zatímco probíhá klasifikace.

5.1 Vyhledání nejdelšího shodného prefixu

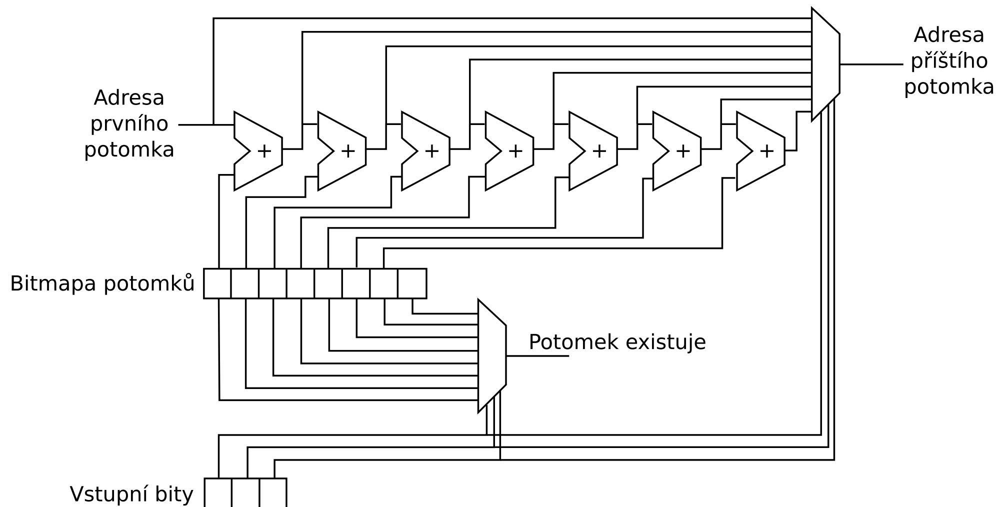
Operace LPM se musí provést nad všemi rozhodujícími políčky z hlavičky paketu. Všechny LPM se v hardware přirozeně provádějí paralelně. To je možné díky skutečnosti, že operace LPM jsou na sobě nezávislé – není nutné mít výsledek jedné k nastartování druhé operace.

Podmínky pro IP adresy přirozeně definují konkrétní hodnoty a prefixy. Podmínky pro porty definují většinou konkrétní hodnoty nebo rozsahy, ty jsou však převedeny na prefixy. V těchto polích má dobré vlastnosti některý specializovaný algoritmus vyhledání nejdelšího shodného prefixu.

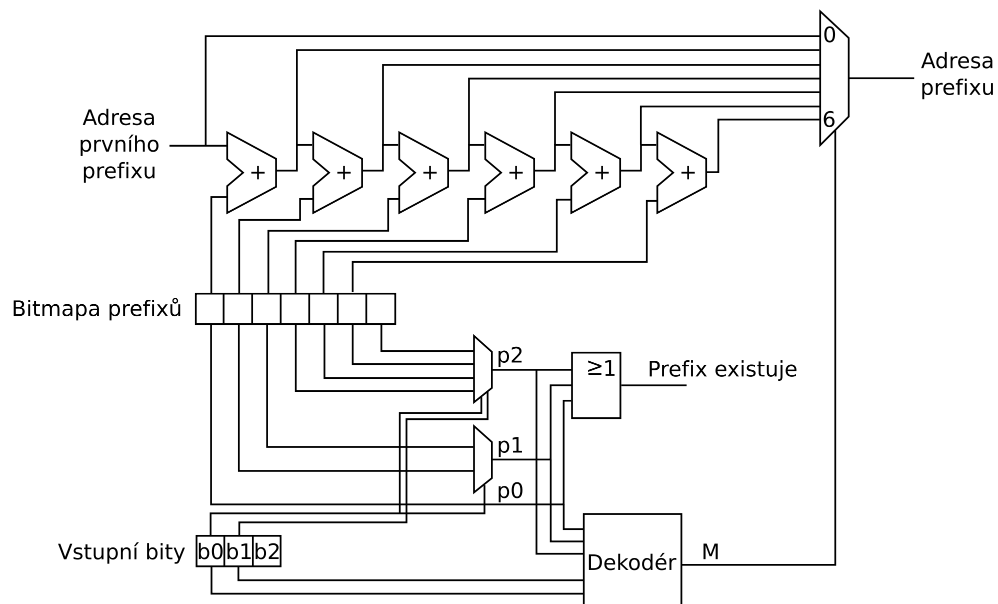
Některá pole ale ze svojí podstaty neobsahují prefixy. Například podmínka na protokol téměř nikdy neobsahuje rozsahy ani prefixy, téměř vždy jde o konkrétní hodnotu, nebo o libovolnou hodnotu. Podobně třeba v případě klasifikace podle MAC adres jsou rozsahy či prefixy definovány velmi výjimečně. Z toho vyplývá, že pro každé pole je vhodné navrhnout specifický způsob implementace operace vyhledání nejdelšího shodného prefixu.

Pro pole přirozeně definující prefixy je zvolen algoritmus Tree Bitmap, popsaný výše. Důvodem použití algoritmu Tree Bitmap je snadnost jeho hardwarové implementace a také možnost nastavení vlastností změnou arity uzlů. Na obrázcích 5.1 a 5.2 jsou obvody realizující dílčí výpočty v algoritmu Tree Bitmap. Celý obvod je potom řízen stavovým automatem. Pro jednoduchost jsou nakresleny obvody pro Tree Bitmap který zpracovává tři bity v jednom kroku. Pro implementaci je vhodnější varianta zpracovávající čtyři bity v jednom kroku. Pro IP adresy verze 4 je tedy potřeba provést maximálně osm kroků algoritmu.

Pro pole, která jsou definována na malém počtu bitů je nejjednodušší způsob použití malé tabulky, která obsahuje výsledek pro každou možnou hodnotu (např. 8 bit pole protokolu, stačí tabulka s 256 položkami). Hodnota pole je tedy přivedena na adresové vodiče



Obrázek 5.1: Obvod pro výpočet adresy potomka (příštího podstromu) v algoritmu Tree Bitmap.



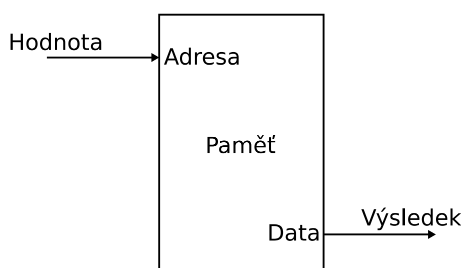
Obrázek 5.2: Obvod pro výpočet adresy nalezeného prefixu v algoritmu Tree Bitmap. Funkce dekodéru z obrázku je popsána v tabulce 5.1.

paměti, která byla předem naplněna výsledky operace LPM. Na obrázku 5.3 je schéma zapojení paměti. Protože jsou uloženy výsledky pro všechny možné hodnoty, je možné podporovat pravidla s rozsahy i prefixy.

Pro pole, která jsou definována na větším počtu bitů by ale přímo vypočtená tabulka byla příliš veliká (např. MAC adresa na 48 bitech). Pokud v množině pravidel existuje jen malé množství unikátních hodnot, pak je možné použít malou asociativní paměť. Pokud není pole z hlavičky paketu v paměti nalezeno, je vrácena hodnota označující libovolná data

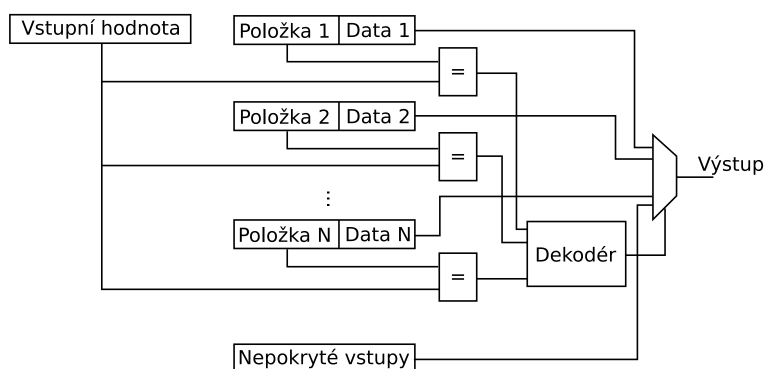
p0	p1	p2	b0	b1	M
1	0	0	X	X	0
X	1	0	0	X	1
X	1	0	1	X	2
X	X	1	0	0	3
X	X	1	0	1	4
X	X	1	1	0	5
X	X	1	1	1	6

Tabulka 5.1: Pravdivostní tabulka dekodéru z obrázku 5.2



Obrázek 5.3: Zapojení paměti pro přímé vyhledání.

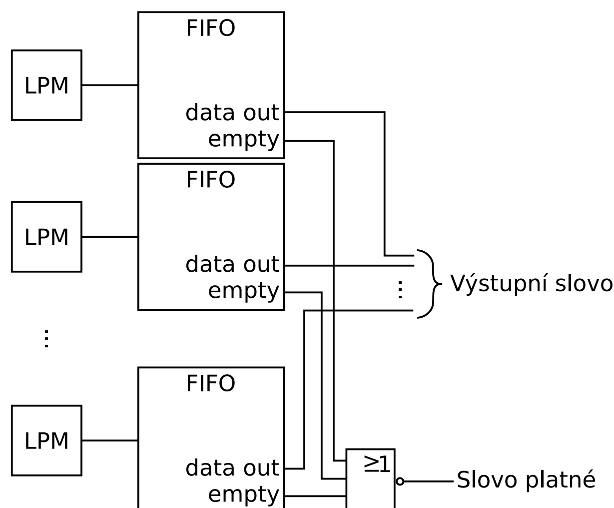
v hlavičce paketu. Na obrázku 5.4 je možné schéma takové asociativní paměti. Nevýhodou asociativní paměti na tomto místě je však omezení na počet unikátních hodnot v tabulce pravidel. Jejich počet je napevno dán zvolenou velikostí asociativní paměti. Není tedy možné zadávat pravidla ve formě rozsahů nebo prefixů, protože ty by se musely převádět na konkrétní hodnoty a tak by snadno docházelo k definování příliš velkého počtu hodnot.



Obrázek 5.4: Asociativní paměť s N položkami obsahuje N komparátorů na rovnost.

Výsledky jednotlivých LPM jsou spojeny do jediného širokého datového slova. Takové spojení je v hardware snadná operace, je potřeba pouze dorovnat případnou rozdílnou

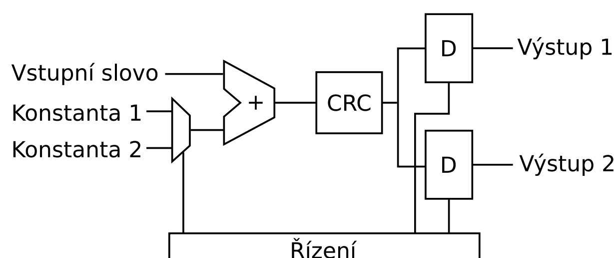
latenci jednotlivých LPM pomocí vyrovnávacích pamětí realizovaných jako fronty FIFO[1] (obrázek 5.5).



Obrázek 5.5: Srovnání různých latencí LPM pomocí front FIFO a konkatenace výsledků do jednoho datového slova.

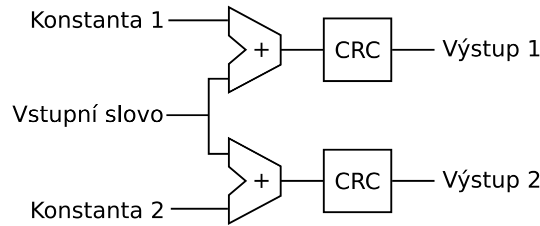
5.2 Výpočet hashovacích funkcí

Dalším krokem algoritmu je výpočet dvou různých hashovacích funkcí. Lze použít jediný algoritmus výpočtu, různost funkcí se zajistí přičtením dvou různých konstant před samotným výpočtem. V síťových aplikacích je často potřeba počítat kontrolní součet CRC [5, 10], který má vhodné vlastnosti i pro hashování. Z toho důvodu použijeme modul výpočtu CRC i pro hashování v klasifikaci paketů. V závislosti na rychlosti výpočtu a na požadované propustnosti algoritmu je možné použít jedinou vypočtení jednotku, která musí provést dva výpočty pro každý paket (obrázek 5.6). Na výstupu je potom nutné uchovat výsledek v klopném obvodu typu D. Druhou možností je použití dvou nezávislých výpočetních jednotek, které pracují paralelně a každá z nich provede jeden výpočet pro každý paket (obrázek 5.7).



Obrázek 5.6: Výpočet dvou hashovacích funkcí s jedinou výpočetní jednotkou. Pomalejší varianta.

[1]First In First Out



Obrázek 5.7: Výpočet dvou hashovacích funkcí se dvěma výpočetními jednotkami. Rychlejší varianta.

5.3 Tabulka vrcholů

Výstupy hashovacích funkcí slouží jako adresy do tabulky vrcholů. Protože tabulka vrcholů je paměťově nejnáročnější část výpočtu, neuvažujeme její zdvojení. Pro každý paket do ní přistupujeme dvakrát, přístupy jsou serializovány multiplexorem. Předpokládáme použití rychlé externí statické paměti. Výstupy z paměti jsou zachyceny v klopných obvodech a je proveden součet. Tím získáme číslo pravidla, které zároveň slouží jako adresa do tabulky pravidel.

5.4 Tabulka pravidel

Tabulka pravidel je realizována jako paměť uvnitř čipu FPGA složená z blokových pamětí. Je nutné zvolit vhodný formát uložení pravidel. Protože se jedná o hardwarovou implementaci, je formát pevný a na jednotlivé bity jsou připojeny obvody realizující další zpracování.

- **IP adresy** se v počítačové praxi často zapisují ve formátu (hodnota, maska). Maska obsahuje jedničky na pozicích, kde jsou platné bity adresy. Na obrázku 5.8 je obvod který určuje shodu vstupní IP adresy s adresou zadanou pomocí hodnoty a masky.

  Nevýhodou použití masky je poměrně velké množství spotřebované paměti. Maska obsahuje stejný počet bitů jako samotná IP adresa, pro IP verze 4 je tedy potřeba 64 bitů na každou adresu. Pro IP verze 6 je to dokonce 256 bitů. Výhodnější proto může být uložení pouze délky prefixu. Pro IP verze 4 může délka prefixu nabývat hodnot 0 až 32. To je 33 různých hodnot, je tedy potřeba 6 bitů k zakódování informace o délce prefixu. Podobně pro IP verze 6 je potřeba 8 bitů. Při určování shody je v tomto případě nejjednodušší použít dekodér, který masku vypočítá ze zakódované délky. Zapojení bylo uvedeno již na obrázku 4.9. Tento způsob vyžaduje méně paměti, ale více logiky na čipu.

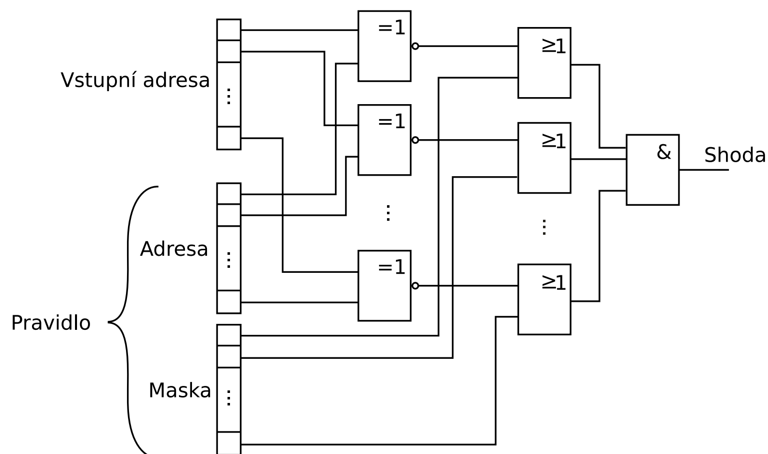
- **Porty** mívají v pravidlech nejčastěji formu rozsahu. Je možné převést rozsahy na prefixy a použít stejný způsob uložení a porovnání jako u IP adres. Protože jeden rozsah lze převést na více prefixů, zvětšuje se při tomto převodu počet pravidel, která je nutné uložit.

  Druhou možností je uložení přímo mezních hodnot intervalu. Díky tomu není již nutné ukládat více pravidel. Ke každému portu musí však být uloženy dvě hodnoty – horní a dolní mez intervalu. Pokud je podmínka pouze jediná hodnota, potom jsou obě

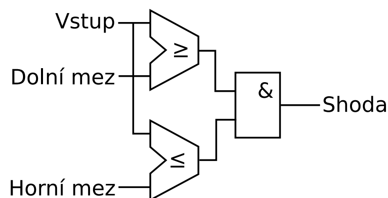
hodnoty shodné. Pokud pravidlo dovoluje libovolnou hodnotu portu, jsou meze nastaveny na nejmenší a nejvyšší možné číslo zobrazitelné na příslušném počtu bitů (16 bitů, tedy hodnoty 0 a 65 535). Na obrázku 5.9 je obvod realizující porovnání hodnoty portu s podmínkou uloženou ve formě rozsahu. Je vidět potřeba dvou aritmetických komparátorů. Podobně jako při výpočtu hashovací funkce, je možné zapojit pouze jedinou jednotku a použít ji dvakrát pro každé porovnání.

- **Ostatní pole** jsou v tabulce pravidel uložena podle svojí povahy. Pole definující rozsahy nebo prefixy jsou uložena podobně jako IP adresy či porty. Pokud pole nedefinuje rozsahy nebo prefixy (například protokol či MAC adresa), je uložena přímo požadovaná hodnota spolu s jedním bitem určujícím, jestli pravidlo definuje podmínku pro toto pole, nebo jestli je přípustná libovolná hodnota.

- **Priorita** musí být také uložena ke každému pravidlu. Je reprezentována jako celé kladné číslo.



Obrázek 5.8: Porovnání hodnoty IP adresy s pravidlem uloženým ve formátu (hodnota, maska). Maska je uložena v invertované podobě.



Obrázek 5.9: Porovnání hodnoty portu s pravidlem. Port je v pravidlu uložen jako horní a dolní mez rozsahu.

Tabulka 5.2 ukazuje příklad výpočtu velikosti jednoho pravidla v paměti pravidel.

Pole	**Bitů**
Zdrojová IP	38
Cílová IP	38
Zdrojový port	32
Cílový port	32
Protokol	9
Priorita	10
Celkem	**159**

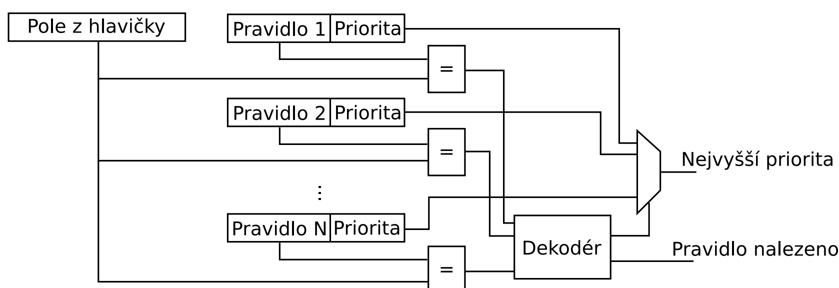
Tabulka 5.2: Příklad výpočtu velikosti položky v tabulce pravidel.

5.5 Výstupní logika a asociativní paměť

Po vyčtení pravidla z tabulky pravidel a jeho porovnání se skutečnými hodnotami z hlavičky paketu mohou nastat dvě možnosti:

- Paket odpovídá pravidlu. Nalezené pravidlo pokračuje do další části výpočtu.
- Paket neodpovídá pravidlu. Pro další zpracování se použije univerzální pravidlo, tedy pravidlo s nejnižší prioritou, které pokrývá všechny pakety.

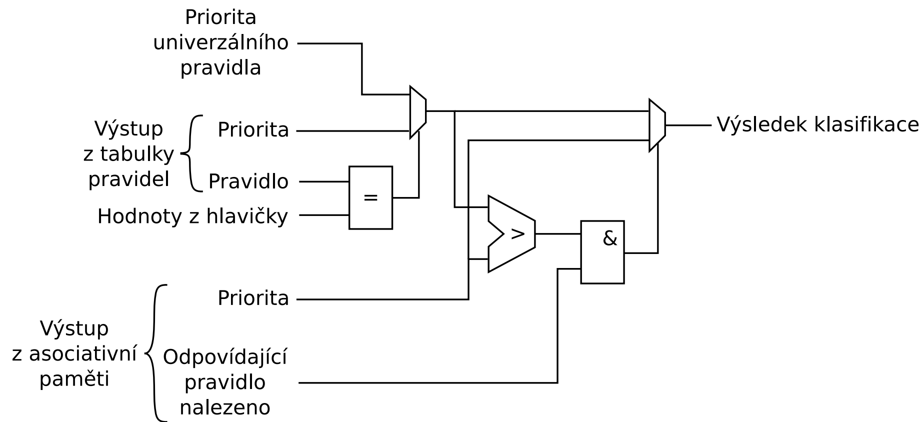
V optimalizované verzi algoritmu je paralelně s celou dosud popsanou větví výpočtu zapojena asociativní paměť, která obsahuje pravidla způsobující nadměrné množství pseudopravidel. Formát uložení pravidel v této paměti může být stejný jako v tabulce pravidel. Ke každé položce asociativní paměti je ale připojen samostatný obvod pro porovnání paketu s daným pravidlem. Jeho výsledkem je rozhodnutí zda paket odpovídá danému pravidlu. Dekodér vybere pravidlo s nejvyšší prioritou a priorita je výsledkem hledání v asociativní paměti. Blokové schéma asociativní paměti je na obrázku 5.10.



Obrázek 5.10: Blokové schéma asociativní paměti pro pravidla.

Výstupy z asociativní paměti jsou k dispozici dříve než výstupy z první větve algoritmu. Proto je opět nutné použít vyrovnávací paměť ve formě fronty FIFO. Posledním krokem je porovnání priorit pravidel nalezených v obou větvích algoritmu. Jako výsledek je pochopitelně vybráno pravidlo s vyšší prioritou. Za výsledek se považuje priorita nalezeného

pravidla, protože priorita jednoznačně identifikuje pravidlo – žádná dvě pravidla nemají stejnou prioritu. Další zpracování paketu na základě výsledku klasifikace přesahuje rámec této práce. Na obrázku 5.11 je schéma výstupní části obvodu.



Obrázek 5.11: Výstupní část obvodové realizace algoritmu klasifikace paketů.

5.6 Příklad aplikace

Navrhujeme klasifikaci podle pěti polí: zdrojová a cílová IP adresa verze 4, zdrojový a cílový port, protokol. Cílem je dosáhnout propustnosti 32 Gbit/s. Pro pakety délky 64 bajtů je paketová rychlost 62,5 Mp/s. Předpokládáme implementaci v FPGA Virtex-5 s pracovní frekvencí 125 MHz. Doba taktu je tedy 8 ns. Z toho vyplývá, že každé dva takty musí být dokončena klasifikace jednoho paketu.

LPM pro IP adresy je realizováno algoritmem Tree Bitmap zpracovávajícím čtyři vstupní bity v jednom kroku. Pro každou adresu je tedy nutné provést maximálně osm kroků algoritmu. Je-li implementována zřetězená jednotka, která má N stupňů zřetězené linky a v každém taktu přistoupí jednou do paměti, trvá jí výpočet N operací LPM celkem

$$T_N = N.8.8\,\text{ns}. \tag{5.1}$$

Protože výpočet jedné operace LPM může trvat maximálně 16 ns, je nutné použít K jednotek, takže

$$K.N.16\,ns = N.8.8\,\text{ns}. \tag{5.2}$$

Z toho vyplývá, že je nutné použít čtyři zřetězené jednotky pro každou z dvojice IP adres.

Čipy Virtex-5 obsahují dvouportové blokové paměti s kapacitou 36 Kbit. Lze je nakonfigurovat na datovou šířku maximálně 36 bitů, potom obsahují 1024 položek. Jeden uzel algoritmu Tree Bitmap pro zpracování 4 bitů v jednom kroku musí obsahovat 16-bitovou masku potomků, 15-bitovou masku výsledků, ukazatel na prvního potomka a ukazatel na první výsledek. Je tedy nutné použít dvě blokové paměti spojené paralelně pro dosažení datové šířky 72 bitů. Protože blokové paměti mají dva nezávislé porty, je možné ke každé takto vzniklé dvojici pamětí připojit dvě jednotky pro LPM. Jednotky jsou čtyři, takže pro každou z dvojice IP adres je nutné použít čtyři blokové paměti. Počet podporovaných

IP adres je větší než 128, protože v nejhorším případě sdílí adresy pouze několik prvních uzlů a dále se strom rozvětvuje tak, že každá adresa projde celkem osmi uzly. Pokud má zřetězená linka čtyři stupně a každá IP adresa v ní musí vykonat osm cyklů, potom latence LPM pro IP adresu je 32 taktů.

LPM pro porty je realizováno stejným algoritmem Tree Bitmap jako IP adresy. Rozdíl je v tom, že čísla portů jsou pouze 16-bitová. Algoritmu tedy stačí provést maximálně čtyři kroky k nalezení výsledku. Proto také stačí poloviční množství jednotek. Pro každý z dvojice portů jsou potřeba dvě jednotky LPM a dvě blokové paměti. Počet podporovaných prefixů portů je větší než 256, vysvětlení je podobné jako u IP adres.

Algoritmus Tree Bitmap používá kromě tabulky uzlů také tabulku výsledků. Tu lze ve skutečnosti vynechat, protože jediná vlastnost, kterou požadujeme pro ohodnocení výsledků je, aby bylo unikátní. Tuto vlastnost splňuje už adresa, takže přímo adresa může být výsledkem LPM a vstupem další části algoritmu.

LPM pro protokol implementujeme přímo tabulkou, protože pole protokolu je pouze na osmi bitech. Je tedy potřeba 256 položek, na to stačí jedna bloková paměť.

Pro výpočet hashovací funkce máme k dispozici modul CRC, který je schopen zpracovat 64 bitů vstupu v každém taktu. Tato datová šířka je dostatečná pro spojení všech pěti výsledků LPM do jednoho datového slova. Protože stačí klasifikovat paket každé dva takty a je nutné vypočíst dvě hashovací funkce pro každý paket, stačí použití jediného modulu CRC. Latence jednotky není větší než 5 taktů.

Tabulka vrcholů je implementována jako externí statická paměť, předpokládáme celkovou latenci 4 takty včetně řadiče paměti. Další takt latence přináší sčítačka. Latenci výstupní části obvodu předpokládáme 4 takty, takže dostáváme celkovou latenci $32 + 5 + 4 + 4 = 45$ taktů, tedy 380 ns. Při rychlosti 32 Gbit/s se za tu dobu přenese 12 160 bitů dat. To je minimální velikost, kterou musí mít paměť pro dočasné uložení paketů. Protože však zároveň 32 Gbit/s znamená přenos 256 bitů za jeden takt (8 ns), musí mít vyrovnávací paměť jedno čtecí a jedno zápisové rozhraní s datovou šířkou alespoň 256 bitů. Toho lze dosáhnout spojením osmi blokových pamětí s datovou šířkou 36 bitů. Celkem tedy má vyrovnávací paměť kapacitu 288 Kbit.

Celkové paměťové nároky uvnitř čipu jsou shrnuty v tabulce 5.3. Pro srovnání, čip Virtex-5 LX110 obsahuje 128 blokových pamětí.

Jednotka	**36 Kbit blokových pamětí**
LPM zdrojová IP adresa	4
LPM cílová IP adresa	4
LPM zdrojový port	2
LPM cílový port	2
LPM protokol	1
Dočasné uložení paketů	8
Celkem	**21**

Tabulka 5.3: Výpočet spotřebovaných blokových pamětí pro příklad klasifikace podle pěti polí.

Kapitola 6

Závěr

V této diplomové práci jsem navrhnul nový algoritmus klasifikace paketů v počítačových sítích. Tomu předcházelo studium dosud známých metod publikovaných v odborné literatuře. Z různých přístupů k problému jsem zvolil skupinu algoritmů založených na dekompozici problému, které jsem prostudoval velice podrobně a analyzoval jsem jejich výhody a nevýhody. Současně jsem nastudoval existující metody pro perfektní hashování.

Na základě získaných informací jsem navrhl nový algoritmus klasifikace paketů. Hlavním kritériem při návrhu byla rychlost celého řešení. Navržený algoritmus má konstantní časovou složitost vzhledem k počtu pravidel. Pro každý paket je nutné udělat právě dva přístupy do externí paměti. Díky tomu je algoritmus obtížně napadnutelný útočníkem – každý paket zatěžuje systém stejně, nelze jej zahltit jistým typem paketů.

Za nevýhodu algoritmu lze považovat paměťovou náročnost. Proto jsem analyzoval vliv jednotlivých pravidel na paměťovou náročnost a navrhl jsem optimalizovanou verzi algoritmu. Ta vychází z pozorování, že většina pravidel nezpůsobuje příliš veliký nárůst paměťových nároků. Existuje však několik pravidel, která generují enormní nárůst spotřebované paměti. Principem optimalizace je nalezení několika klasifikačních pravidel, která způsobují největší nárůst paměťových nároků. Tato pravidla jsou vyňata z množiny pravidel a umístěna v malé asociativní paměti na čipu. Díky této optimalizaci je možné řádově snížit paměťové nároky algoritmu.

Dále jsem navrhl hardwarovou architekturu s využitím technologie FPGA. Původní zadání práce předpokládalo použití karty Combo6X pro ověření funkčnosti řešení. Při použití dvou statických pamětí na frekvenci 100 MHz lze na kartě Combo6X dosáhnout rychlosti klasifikace až 100 Mp/s. V průběhu práce jsem se však na pokyn vedoucího zaměřil na modernější kartu ComboV2, kde při osazení vhodnými pamětmi typu QDR–II je možné dosáhnout rychlosti klasifikace 125 Mp/s s využitím jediné paměti. Pro nejkratší pakety tak dostáváme propustnost 64 Gbit/s.

V průběhu práce se ukázala šířka a komplexnost studované problematiky. Návrh architektury popisuje pouze samotný klasifikační algoritmus. Pro funkční systém je však zapotřebí mnoho dalších součástí, například obsluha síťových rozhraní, zajištění komunikace s hostitelským počítačem, nebo softwarové generování konfiguračních údajů pro klasifikační jednotku. Implemetace takového systému bude vyžadovat celý vývojový tým, proto není součástí této práce.

Algoritmus je možné dále zkoumat a zdokonalovat. Je třeba zaměřit se na způsob výběru odstraněných pravidel v optimalizované verzi algoritmu, případně na další možnosti snížení paměťové náročnosti algoritmu. Také konkrétní architektura jednotlivých částí může být

optimalizována podle různých hledisek, zejména pro zvýšení rychlosti a snížení plochy na čipu.

Hlavní myšlenka nového algoritmu je intenzivní předzpracování dat před samotným výpočtem. Na začátku je nutné relativně pomalé hledání vhodné funkce. Jakmile je taková funkce nalezena, lze ji použít pro extrémně rychlé hledání. Jsem přesvědčen že popsaný princip, spolu s ideou záměrných kolizí hashovací funkce, může najít uplatnění ve více oblastech výpočetní techniky, než jen klasifikace paketů. Výsledky mojí práce byly sepsány a poslány k publikaci na konferenci FPL 2008.

Literatura

[1] 36-mbit qdr-ii sram 2-word burst architecture. Cypress Semiconductor Company.

[2] Virtex-5 product table. Xilinx, Inc.

[3] 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.

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

[5] Giuseppe Campobello, Giuseppe Patane, and Marco Russo. Parallel crc realization. *IEEE Transactions on Computers*, 52(10):1312–1319, 2003.

[6] 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.

[7] 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.

[8] Will Eatherton, George Varghese, and Zubin Dittia. Tree bitmap: hardware/software ip lookups with incremental updates. *SIGCOMM Computer Communication Review*, 34(2):97–122, 2004.

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

[10] Tomas Henriksson and Dake Liu. Implementation of fast crc calculation. In *ASPDAC: Proceedings of the 2003 conference on Asia South Pacific design automation*, pages 563–564, New York, NY, USA, 2003. ACM.

[11] 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.

[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.

[17] David E. Taylor and Jonathan S. Turner. Classbench: a packet classification benchmark. *IEEE/ACM Trans. Netw.*, 15(3):499–511, 2007.