# Algoritmy pro klasifikaci paketů

**Viktor Puš**
Vypočetní technika a informatika, 1. ročník, prezenční studium
Školitel: Václav Dvořák

Fakulta informačních technologií, Vysoké učení technické v Brně
Božetechova 2, 612 66 Brno

ipus@fit.vutbr.cz

**Abstrakt.** Článek pojednává o problematice klasifikace paketů v počítačových sítích. Je naznačeno proč se jedná o netriviální problém a argumenty pro implementaci specializovaným hardware. Zatímco klasickým přístupem je využití ternárních asociativních pamětí, výzkum v této oblasti se zaměřuje spíše na algoritmické řešení problému. Článek popisuje algoritmus klasifikace paketů s konstantní časovou složitostí. Jsou uvedeny také nevýhody algoritmu a možnosti jeho zlepšení.

**Klíčová slova.** Klasifikace paketů, bezpečnost, FPGA, složitost

## 1 Úvod

S rozvojem počítačových sítí se filtrování provozu stalo jedním ze základních kroků při zabezpečování počítače nebo sítě. Základním zařízením pro filtrování paketů je bezestavový firewall. Ten pro každý paket nezávisle na základě dané množiny pravidel rozhoduje o vykonané akci. Základní akce jsou povolit/zakázat paket. Se zrychlováním sítí stoupají také nároky na rychlost takových zařízení. Ačkoliv existují softwarová řešení filtrace paketů [1, 2], jejich výkon nemůže postačovat nárokům dnešních vysokorychlostních sítí.

Klasifikační algoritmus obsahuje množinu pravidel uspořádanou podle priority. Každé pravidlo definuje podmínky pro významná políčka z hlavičky paketu. Typicky jde o zdrojovou a cílovou IP adresu, zdrojový a cílový TCP/UDP port a číslo protokolu. Podmínka může mít tvar přesné shody, prefixu (často pro IP adresy), nebo rozsahu (často pro porty). Cílem klasifikačního algoritmu je najít pro každý paket odpovídající pravidlo s nejvyšší prioritou.

Tradiční metoda klasifikace paketů v hardware využívá ternární asociativní paměti. Tato součástka přímo podporuje vyhledávání prefixů (nastavením nižších bitů do stavu X). Z toho důvodu se používá převod rozsahů na prefixy - každý rozsah lze vyjádřit jako jeden nebo více prefixů. TCAM sice podporují velmi jednoduchou a rychlou klasifikaci (jediným přístupem do paměti), ale mají také řadu nevýhod. Je to především vysoká cena a příkon, dále například nutnost použít široké datové sběrnice (klasifikační pravidlo může být reprezentováno datovým slovem širokým stovky bitů, stejně tak porovnávané hodnoty z hlavičky paketu).

Proto existuje snaha nacházet jiná řešení klasifikace paketů, založená na použití algoritmu namísto specializované součástky.

Zbytek článku má následující strukturu: Následující kapitola přibližuje některé dosud publikované přístupy a diskutuje jejich nevýhody. Třetí kapitola popisuje algoritmus publikovaný na konferenci FPGA 2009 jako jeden z nejnovějších výsledků v této oblasti. Další část článku shrnuje dosažené experimentální

výsledky algoritmu a diskutuje jeho nevýhody. Poslední kapitola shrnuje článek a diskutuje možnosti zdokonalování stávajících řešení.

## 2 Existující řešení

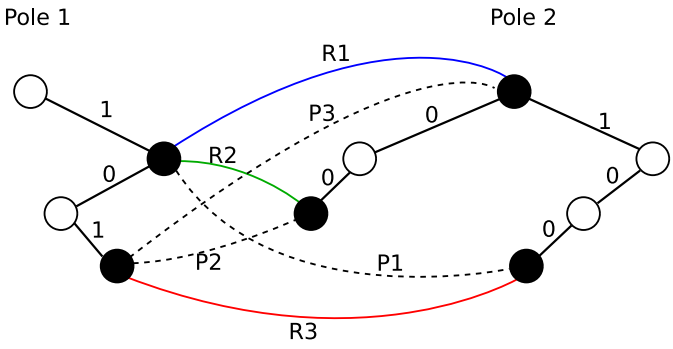
Problém klasifikace je obtížný i z teoretického hlediska [3]. Bylo například dokázáno, že při dosažení konstantní časové složitosti vyhledávání je prostorová složitost exponenciální [8]. S těmito omezeními se snaží vypořádat autoři mnoha publikovaných prací.

Velký zájem budí v posledních letech algoritmy založené na dekompozici problému. Tyto algoritmy totiž mají potenciál na dosažení velmi vysokých rychlostí jak z pohledu implementace, tak také teoreticky je zde možnost konstantní časové složitosti. Dekompoziční algoritmy dělí problém na několik menších, lépe zvládnutelných částí. První krok je často stejný nebo podobný, jedná se o operaci vyhledání nejdelšího shodného prefixu (LPM - Longest Prefix Match). Podobně jako při použití TCAM se zde tedy rozsahy v pravidlech musí převést na prefixy. Důležité je, že operace LPM se vykonává nad všemi významnými poli z hlavičky paketu nezávisle. Z toho vyplývá, že lze zpracovávat celou hlavičku paketu paralelně. Tento fakt podporuje vhodnost dekompozičních algoritmů pro hardwarovou implementaci.

LPM je základní operace při směrování v IP sítích, odpovídá totiž vyhledávání v routovací tabulce. Z toho důvodu existuje mnoho přístupů k tomuto problému, většinou vycházejících ze stromového algoritmu trie [5, 7, 10].

Po operaci LPM je nutné zkombinovat výsledky a najít správné pravidlo. Tato operace je specifická pro každý dekompoziční algoritmus a jde také o krok nejnáročnější především na paměť. Základní algoritmus kartézského součinu [11] používá předpočítanou tabulku obsahující správný výsledek pro všechny možné kombinace (kartézský součin) výsledků LPM. V tabulce je uloženo přímo číslo správného pravidla, takže jediný přístup do tabulky dává přímo výsledek klasifikace. Vzhledem k multiplikativnímu charakteru kartézského součinu však může být taková tabulka mimořádně velká. Navíc je do ní nutné přistupovat dostatečně rychle, tedy zřejmě bude tabulka implementovaná jako hashovací a je nutné navíc řešit kolize hashovací funkce.

Podstatným zdokonalením tohoto algoritmu je Multi-Subset Crossproduct Algorithm [6]. V tomto algoritmu je nepříznivý dopad multiplikativního charakteru kartézského součinu zmírněn rozdělením množiny pravidel na podmnožiny. Rozdělení je voleno tak, aby v jednotlivých podmnožinách existovalo malé množství možných výsledků LPM, čímž dochází také ke vzniku malého množství kombinací výsledků LPM, které je nutno pokrýt. Autoři práce zavádějí pojem *pseudopravidlo*, které demonstrujeme na příkladě:




Obrázek 1: Tři pravidla $R1, R2, R3$ a tři pseudopravidla $P1, P2, P3$.

Na obrázku 1 jsou naznačeny 2 struktury trie pro vyhledávání prefixů. Černé uzly obsahují platné prefixy, bílé uzly neobsahují žádný platný prefix. Barevnými oblouky jsou naznačena pravidla $R1, R2$ a

| Pravidlo | Pole 1 | Pole 2 | Správné pravidlo |
|---|---|---|---|
| $R1$ | 1* | * | |
| $R2$ | 1* | 00* | |
| $R3$ | 101 | 100 | |
| $P1$ | 1* | 100 | $R1$ |
| $P2$ | 101 | 00* | $R2$ |
| $P3$ | 101 | * | $R1$ |

Tabulka 1: Pravidla a pseudopravidla.

$R3$ odpovídající první části tabulky 1. Čárkovaně jsou potom naznačeny všechny zbylé kombinace, které však musí klasifikační algoritmus také správně zpracovat. Tyto kombinace se nazývají *pseudopravidla* a odpovídají nepokrytým hodnotám kartézského součinu množin platných prefixů.

Při podrobnějším pohledu je zřejmé že každé pseudopravidlo je speciálním případem nějakého pravidla. V posledním sloupci tabulky 1 je pro všechna pseudopravidla uveden správný výsledek klasifikace v případě, že výsledkem LMP je daná kombinace hodnot.

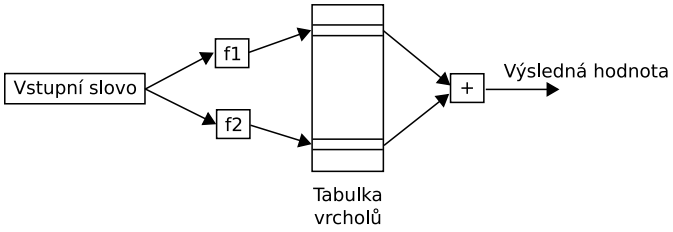
Problémem dekompozičních metod jsou tedy pseudopravidla, jejichž množství může být exponenciální vzhledem k počtu pravidel. V uvedeném článku je zároveň představena metoda pro snížení jejich počtu. Metoda je založena na pozorování, že v běžných množinách se vyskytuje několik pravidel, která způsobují extrémní nárůst počtu pseudopravidel (tzv. *spoilers*). Dokážeme-li tato pravidla najít, je možné je odstranit a pro klasifikaci podle nich použít jiný přístup, například malou TCAM na čipu.

# 3 Perfect Hashing Crossproduct Algorithm

Náš algoritmus [9] patří také do skupiny dekompozičních metod, prvním krokem je tedy operace LPM. Následující krok musí obsahovat mapování všech možných kombinací výsledků LPM (pseudpravidel) na správné číslo pravidla. Protože pseudopravidel je mnohem více než pravidel, půjde o mapování *many-to-one*. Je využita na míru zkonstruovaná hashovací funkce, která vychází z perfektní hashovací funkce.

Perfektní hashovací funkce jsou takové funkce, které neobsahují žádné kolize pro zadanou množinu klíčů. V jednodušším případě statické perfektní hashovací funkce je množina klíčů předem známá a je k dispozici čas na vytvoření potřebných datových struktur.

Algoritmus [4] je určen pro konstrukci právě takových funkí. Jeho vlastností však je, že lze předem pro každý klíč zvolit celé číslo, které má být výsledkem funkce. Je-li tedy některým klíčům přiřazeno stejné číslo, výsledná funkce nebude bezkolizní. Toho využíváme pro konstrukci hashovací funkce, která všechna pseudopravidla (reprezentovaná výsledky LPM v binární podobě) zobrazí právě na správné číslo pravidla. Vytvořená hashovací funkce tedy není ani zdaleka perfektní, naopak obsahuje velké množství řízených kolizí.

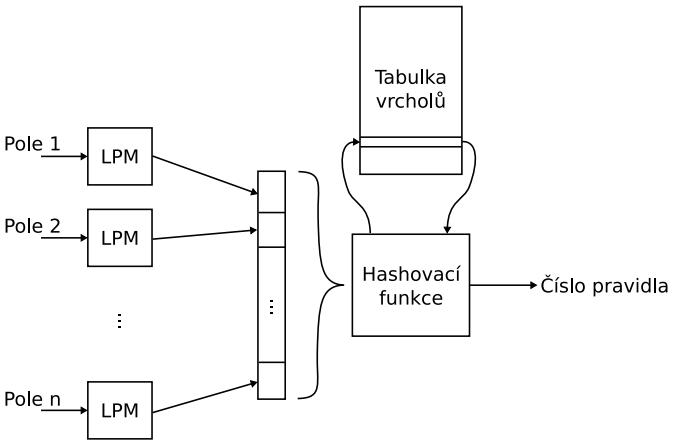



Obrázek 2: Princip perfektní hashovací funkce

Princip výpočtu hodnoty hashovací funkce je naznačen na obrázku 2. Nejprve jsou nad vstupem vypočteny dvě běžné hashovací funkce $f1, f2$. Jejich výsledky jsou použity jako adresa do tabulky, z níž jsou vyčtena dvě celá čísla. Po provedení běžného aritmetického součtu máme výsledek perfektní hashovací funkce.

Pro naplnění tabulky se používá konstrukce náhodného acyklického grafu, ve kterém klíče odpovídají hranám a vrcholy jsou ohodnoceny tak, aby jejich součet dával požadovanou hodnotu funkce pro hranu spojující tyto vrcholy. Experimentální výsledky ukazují, že pro to, aby byl náhodný graf s dostatečnou pravděpodobností acyklický je nutné, aby počet vrcholů byl alespoň dvojnásobkem počtu hran.

Na obrázku je struktura celého popsaného algoritmu PHCA:




Obrázek 3: Schéma Perfect Hashing Crossproduct Algorithm

# 4 Vlastnosti algoritmu PHCA

## 4.1 Časová složitost

Z obrázku 3 je vidět že lze vyhodnocovat odděleně vlastnosti první části - operace LPM a druhé části - hashovací funkce. V případě použití stromového algoritmu pro výpočet LPM je časová složitost lineární s počtem bitů vstupního slova. Protože ale tento počet bitů je předem známý a konstantní (např. IPv4 má vždy 32 bitů), dovolíme si časovou složitost výpočtu LPM zjednodušit na konstantní. Výpočet hashovací funkce je zřejmý z obrázku 2. Je nutné dvakrát vyčíslit běžnou hashovací funkci, dvakrát přistoupit do paměti a provést jeden součet. Všechny tyto operace mají konstantní časovou složitost.

Z uvedeného vyplývá, že PHCA je algoritmus klasifikace paketů s konstantní časovou složitostí.

## 4.2 Prostorová složitost

Ačkoliv algoritmus nikam neukládá pseudopravidla (na rozdíl od MSCA), jejich počet stále ovlivňuje velikost potřebné paměti. Je tomu tak proto, že při konstrukci hashovací funkce je nutné mít v tabulce vrcholů přibližně dvakrát víc položek než je hran - v našem případě pseudopravidel. I když položka tabulky je jen jedno celé číslo (např. 16 bitů), velikost tabulky může narůstat exponenciálně.

## 4.3 Experimentální výsledky

Tabulka 2 obsahuje experimentální výsledky pro reálné (fw1-fw4) a syntetické (synth1-synth6) množiny pravidel. Pro vygenerování syntetických pravidel byl použit nástroj ClassBench [12].

| Množina | Pravidel | Tabulka vrcholů [B] |
|---|---|---|
| fw1 | 32 | 740 |
| fw1 | 58 | 3 424 |
| fw3 | 103 | 252 220 |
| fw4 | 171 | 116 356 |
| synth1 | 40 | 2 740 |
| synth2 | 49 | 6 601 |
| synth3 | 49 | 5 035 |
| synth4 | 70 | 2 451 |
| synth5 | 82 | 22 495 |
| synth6 | 100 | 3 827 |

Tabulka 2: Velikost potřebné paměti pro uložení hashovací funkce algoritmu PHCA. Předpokládá se odstranění 16 pravidel do TCAM.

## 5 Diskuze, závěr

Uvedené výsledky ukazují, že klasifikace paketů není triviální problém. Zatímco legitimním požadavkem uživatele je konstantní paketová rychlost zpracování, její dosažení je možné jedině za cenu značného nárůstu potřebné paměti.

V současnosti existuje metoda optimalizace paměťových nároků dekompozičních algoritmů založená na identifikaci a odstranění pravidel, která způsobují největší nárůst počtu pseudopravidel. Ačkoliv tato metoda nemůže snížit třídu složitosti problému, dokáže v reálných situacích zmenšit paměťové struktury o jeden až dva řády.

Cílem disertační práce je snížení paměťových nároků algoritmů klasifikace paketů pomocí dalších optimalizací. Tomu musí předcházet podrobná analýza příčin vzniku pseudopravidel a sada experimentů nad reálnými sadami pravidel. V několika publikovaných pracech již lze nalézt zmínky o tom, že množiny pravidel vykazují jistou strukturu a společné vlastnosti. Budou-li v typických filtrovacích sadách pravidel identifikovány opakující se vzory nebo zvyklosti, bude také možné využít této znalosti ke konstrukci lepších klasifikačních algoritmů.

Kromě optimalizace lze také zkoumat možnosti kompromisu - snížení velikosti potřebné paměti za cenu zpomalení algoritmu. Tím by vznikla užitečná možnost vyladit algoritmus na potřebnou paketovou propustnost při znalosti výpočetního výkonu (zejména maximální frekvence) použité technologie.

## Poděkování

Tato práce vznikla za podpory výzkumného záměru MSM0021630528 Výzkum informačních technologií z hlediska bezpečnosti.

## Reference

[1] Netfilter: firewalling, NAT and packet managing for Linux. http://www.netfilter.org/ , June 2008.

[2] PF: The OpenBSD Packet Filter.
http://www.openbsd.org/faq/pf/ , June 2008.

[3] F. Baboescu, S. Singh, and G. Varghese. Packet classification for core routers: Is there an alternative to CAMs? In *INFOCOM*, 2003.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

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

[9] Viktor Puš and Jan Kořenek. Fast and scalable packet classification using perfect hash functions. In *FPGA '09: Proceeding of the ACM/SIGDA international symposium on Field programmable gate arrays*, pages 229–236, New York, NY, USA, 2009. ACM.

[10] Haoyu Song, Jonathan Turner, and John Lockwood. Shape shifting tries for faster ip route lookup. *Network Protocols, IEEE International Conference on*, 0:358–367, 2005.

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

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