VYHLEDÁNÍ NEJDELŠÍHO SHODNÉHO PREFIXU V FPGA

Jiří Tobola
Výpočetní technika a informatika, 2. ročník, prezenční studium
Školitel: Vladimír Drábek

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

itobola@fit.vutbr.cz

Abstrakt. Článek shrnuje aktuální metody a poznatky pro řešení úlohy vyhledání nejdelšího prefixu v síťových zařízeních a navrhuje optimalizace s cílem efektivní implementace pro IPv6 a multidesetigigabitové sítě. Hlavním zaměřením je analýza současných metod z hlediska časové a paměťové složitosti a náročnosti na technické zdroje implementace. Dále jsou navrženy optimalizace algoritmů a hybridní algoritmus pro časově a prostorově efektivnější řešení daného problému. Článek je východiskem pro řešení disertační práce s cílem navrhnout, experimentálně ověřit a implementovat rychlý a efektivní algoritmus vyhledání nejdelšího síťového prefixu s využitím technologie FPGA a se zaměřením na IPv6 protokol.

Klíčová slova. Longest prefix match, LPM, IP lookup, IPv4, IPv6, FPGA, algoritmus.

1 Úvod

Rozvoj počítačových sítí a zvláště internetu přináší stále rychlejší technologie pro přenos dat. Zároveň s rychlostmi se v poslední době zvyšují i nároky na bezpečnost a sledování sítí. Právě vysoké přenosové rychlosti ale přinášejí problém, jak data efektivně zpracovávat, klasifikovat, monitorovat a analyzovat. Na rychlostech linek v řádech desítek gigabitů za vteřinu musí být řešeny úlohy směrování, filtrování dat, monitorování síťových toků, vyhledávání vzorů a další. Základní operací s pakety při všech těchto úlohách je klasifikace na základě IP adres a potažmo operace 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 ten prefix ze vstupní množiny, který odpovídá dané hodnotě a je nejdelší, tedy nejspecifičtější. Konkrétně v oblasti sítí se algoritmus uplatňuje při vyhledání do jaké IP podsítě daná IP adresa patří. Výsledky této operace jsou klíčovým bodem pro směrování v IP sítích, nicméně LPM se uplatňuje i v dalších síťových úlohách jako je například klasifikace paketů na základě více vstupních kritérií při využití dekompozičních metod.

Protože je operaci LPM nutné provádět na rychlostech odpovídajících desetigigabitovým sítím, což znamená milióny běhů algoritmu za vteřinu, je tato úloha typicky řešena hardwarovou akcelerací. Obecně mohou být tato hardwarová řešení založena na technologiích ASIC (Application-Specific Integrated Circuit) či FPGA (Field Programmable Gate Array). U hardwarově akcelerovaných LPM čipů se typicky pro směrovače uplatňují ASIC prvky orientované na nízkou ceny, vysoký výkon a velkou množinu pravidel (hraniční směrovače pracují s databázemi prefixů o velikostech stovek tisíc). Naopak u ostatních prvků (firewally, bezpečnostní sondy) se často uplatňují FPGA řešení, která nabízejí větší obecnost a kombinaci akcelerace více úloh (např. klasifikace a následně vyhledávání vzorů) při stejné výkonnosti ale menší velikosti podporovaných pravidel [1].

Metody vyhledávání nejdelšího shodného prefixu jsou v literatuře i v komerční praxi dobře zpracovaným tématem, nicméně v současné době při nástupu IPv6 protokolu se tato tématika znovu otevírá. Cílem této práce je proto analyzovat současné LPM metody, vyhodnotit jejich vhodnost pro implementaci v FPGA a posoudit jejich škálovatelnost pro IPv6. Na základě provedené studie pak navrhnout metodu novou či optimalizovat metodu současnou s cílem nalézt efektivnější implementaci pro IPv6 při dosažení multidesetigigabitových rychlostí a řádově tisícové množiny prefixů s využitím vnitřních pamětí v FPGA.

Článek je dále členěn do několika kapitol, jenž jsou organizovány následovně. Kapitola 2 shrnuje motivaci pro řešení dané problematiky. V kapitole 3 jsou představeny aktuálně používané algoritmy pro řešení úlohy LPM a kapitola 4 navrhuje optimalizace a heuristiky s cílem nalézt vhodný algoritmus pro vyhledávání prefixů v FPGA při dosažení požadovaných parametrů. Závěrečná kapitola 5 shrnuje aktuální stav disertační práce a přibližuje plán dalších prací.

2 Motivace

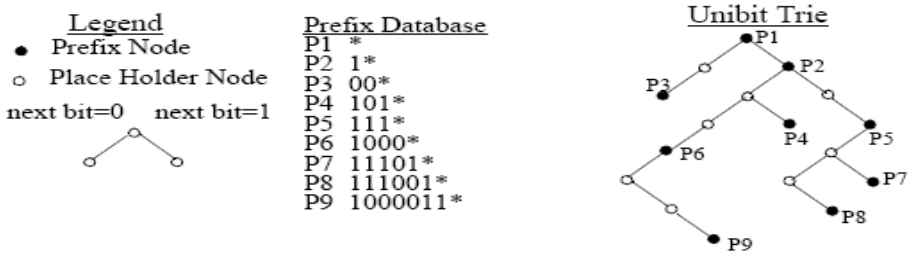
Přestože je úloha LPM typicky spojována pouze se směrovači, základní motivací pro tuto práci je jiná úloha, a to klasifikace paketů pomocí dekompozičních metod. Tyto klasifikační metody využívají rozdělení problému na dvě úlohy – vyhledání nejdelších shodných prefixů nad jednotlivými položkami z hlaviček paketů a následné určení odpovídajícího pravidla. Příkladem dekompozičního algoritmu je Perfect Hashing Crossprodukt Algorithm [2]. Při implementaci v FPGA je omezujícím prvkem tohoto algoritmu použitá metoda LPM – Tree Bitmap. Cílem této práce je proto identifikovat vhodnější metodu, která bude dále podporovat protokol IPv6 a bude efektivně využívat omezené paměťové zdroje dostupné na čipech FPGA. Samozřejmostí musí být podpora vysokorychlostních sítí, což znamená 15 miliónů vyhledání na desetigigabitové síti a desetinásobek na sítích příští generace.

3 Současné LPM algoritmy

Pokud pomineme speciální řešení typu vyhýbání se dané operaci (např. pomocí paměti cache, nebo přepínaná na základě značek – např. MPLS) nebo speciální hardwarové architektury typu TCAM je možné algoritmy rozdělit do dvou skupin – algoritmy založené na struktuře trie a ostatní. Dále jsou jednotlivé algoritmy popsány a analyzovány z hlediska hardwarové implementace.

3.1 Trie

Klasický algoritmus vyhledání nejdelšího prefixu se nazývá trie z anglického retrieval. Někdy se dle principu na kterém funguje nazývá i prefixový strom. Algoritmus využívá stromovou datovou strukturu, která obsahuje vyhledávané prefixy přímo ve své konstrukci. Strom je konstruován z uzlů, kde každý uzel obsahuje právě dva ukazatele na další uzly, přičemž jedna z těchto dvou hran odpovídá bitu 0, druhá bitu 1.



Obrázek 1: Struktura Trie.

V průběhu výpočtu se zpracovávají postupně bity hledaného prefixu od nejvýznamnějšího k nejméně významnému. V každém kroku výpočtu se podle daného bitu rozhoduje, kterým podstromem se bude postupovat. V uzlech stromu je přitom uloženo, zda zatím zpracovaní bity obsahují platný prefix, a pokud ano tak je v uzlu uloženo i jeho číslo. V průběhu celého výpočtu je zaznamenáván poslední platný prefix a při zpracování všech bitů je tento předán dále jako výsledek. Výpočet algoritmu je ukončen pokud jsou zpracovány všechny bity ze vstupu, nebo pokud s danou hodnotou bitu není možné dále postupovat stromem (příslušný podstrom neexistuje).

Výhodou tohoto algoritmu někdy nazývaného také unibit Trie je jednoduchá datová struktura, rychlé aktualizace vyhledávaných prefixů a nízké paměťové nároky. Nevýhodou je rychlost, která v nejhorším případě znamená 32 náhodných přístupů do paměti. Existuje celá řada modifikací tohoto základního algoritmu, přičemž nejčastěji se tyto modifikace zaměřují na zpracování více bitů v jednom kroku pomocí zvýšení arity uzlů stromu, či kompresi uložené datové struktury.

3.2 Controlled Prefix Expansion (CPE)

Hlavní myšlenkou algoritmu představeného v [3] je použít strukturu trie, která zpracovává více bitů najednou kvůli rychlosti. Protože ne každý prefix musí být násobkem počtu bitů, které se zpracovávají tak všechny prefixy jsou expandovány na délku n. Např. prefix 0* při zpracování tří bitů v jednom taktu expandujeme na prefixy: 000, 001, 010, 011. Samozřejmě pokud existuje delší konkrétní prefix, který odpovídá některé expandované hodnotě, tak použijeme místo expandované hodnoty tento.

V každém uzlu algoritmu jsou uloženy prefixy a odkazy na následující uzel/podstrom pro všechny možné varianty vstupu. Výhodou tohoto algoritmu je vyšší rychlost, nevýhodou pak zejména při větším počtu bitů zpracovávaných najednou plýtvání paměti. Optimalizací, která zmenší potřebnou paměť na polovinu, je tzv. technika „leaf pushing“. Hlavní myšlenkou je neukládat pro každý možný vstup prefix i ukazatel na následovníka ale pouze jednu z těchto hodnot. Pokud byly v původním záznamu uloženy obě hodnoty tak prefix přesuneme do následujících listů viz obrázek:

3.3 Lulea Compressed Tries

Metoda Lulea [4] vychází z optimalizovaného konceptu CPE a dále se zaměřuje na efektivnější paměťovou optimalizaci. Hlavní myšlenkou je nahrazení všech elementů se stejnou hodnotou, které jsou uloženy v paměti za sebou, jedinou hodnotou. Díky této technice dochází v praxi k velké redukci uložených elementů v rámci uzlu. Pro snadné procházení strukturou jsou vynechané elementy vyznačeny na bitmapě, která je novou součástí každého uzlu. Výhodou této metody neúčinná komprese a tedy nižší paměťové nároky. Nevýhodou je velmi pomalá operace přidání nového prefixu.

3.4 LC Tries

Algoritmus byl představen v [5] a zabývá se jiným typem komprese trie zpracovávající více bitů v jednom taktu. Hlavní myšlenka je využití proměnného počtu bitů v každém kroku, tak, aby prefixy vždy byly uloženy v listech. Nevýhodou je nedeterministický počet kroku a tím pádem nevhodnost pro hardwarovou implementaci.

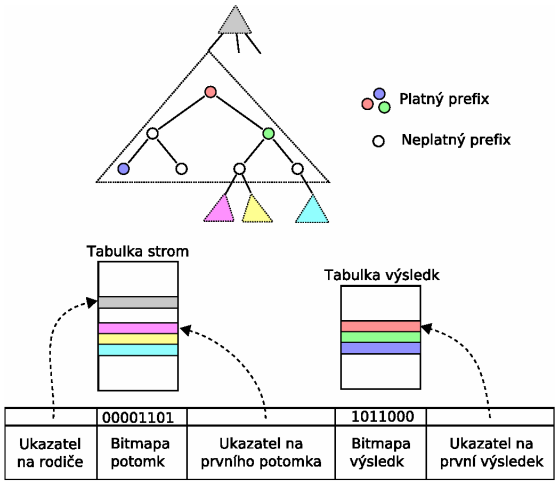
3.5 Tree Bitmap

Tento algoritmus byl představen v [6] a jedná se dnes o jeden z nejčastěji využívaných algoritmů pro hardwarové implementace LPM. Jeho výhodou je paměťová nenáročnost, relativně snadná hardwarová realizace a možnost parametrizace arity uzlů a tedy zpracovávaných bitů za takt. Časová složitost je lineární s počtem bitů vstupního slova, protože je však délka vstupního slova konstantní, je i výsledná časová složitost algoritmu konstantní.

Algoritmus pracuje se strukturou trie kde arita uzlu je rovna mocnině čísla dvě. Jeden uzel Tree Bitmap tedy reprezentuje několik uzlů jednoduché struktury trie. Každý takový uzel odpovídá

binárnímu podstromu s hloubkou, která se rovná mocnině arity uzlu. Každý uzel může mít několik následníků (podstromů), kteří jsou uloženi v paměti za sebou. Vynechány jsou přitom ti následníci kteří neobsahují žádné prefixy ani nemají žádné další následníky a platnost či neplatnost podstromu je zakódována v jednoduché bitmapě. Díky této optimalizaci může být v uzlu uložen ukazatel pouze na prvního svého následníka a bitmapa platných následníků. Pokud je v bitmapě uložena jednička následník/podstrom existuje, v opačném případě nikoliv. Na základě adresy prvního následníka a bitmapy snadno určíme adresy všech přímých podstromů daného uzlu.

Kromě svých následníků uzel dále obsahuje čísla prefixů, která v něm končí. Ty jsou podobně jako následníci uloženy v paměti za sebou a pro jejich identifikaci stačí ukazatel na první výsledek a bitmapa platných prefixů. Datovou strukturu lze v případě potřeby dále rozšířit o zpětné ukazatele s jejichž pomocí lze strom procházet i v opačném směru.



Obrázek 2: Datová struktura Tree Bitmap - uzel obsahující tři prefixy a tři následníky.

3.6 Shape Shifting Tree

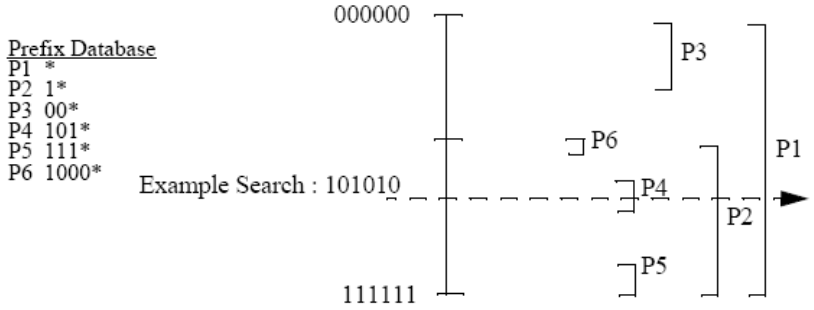
Tento algoritmus představený v [7] vychází z předchozího algoritmu Tree Bitmap a dále jej optimalizuje. Stejně jako v Tree Bitmap jsou následující uzly/podstromy i výsledné prefixy uloženy v paměti za sebou a je využita interní a externí bitmapa označující platné odkazy. Navíc je ve struktuře zavedena nová bitmapa – Shape, která kóduje tvar trie zakódované daným uzlem. Pomocí tohoto rozšíření je umožněno ještě efektivnější uložení uzlů v paměti, kdy na základě tvaru uzlů (dlouhá nevětvená cesta vs větvící se podstrom) se zvolí ideální kódování. Omezí se tedy dlouhé, úzké cesty, které znamenaly plýtvání paměti.

Mimo efektivnější uložení přináší algoritmus také vyšší rychlost vyhledání neboť se omezuje hloubka stromu a tím se zvyšuje rychlost průchodu struktury. Další navrženou optimalizací je přepínání mezi TreeBitmap a Shape Shifting Trie pomocí jednoho řídícího bitu v uzlu na základě předchozí analýzy, která struktura je pro daný uzel vhodnější. Hlavními přínosy metody jsou menší paměťové nároky a vyšší rychlost algoritmu. Také je jako u jedné z mála metod diskutován přínos využití tohoto algoritmu při vyhledání IPv6 prefixů.

3.7 Binární vyhledávání na intervalech

Tento algoritmus [8] nahlíží na databázi IP prefixů jako na množinu rozsahů. Na obrázku níže je naznačen příklad pomocí vzorové databáze a grafické reprezentace prefixů. Délka adresy je v příkladu

6 bitů. Z obrázků je patrné, že jednotlivé prefixy mohou být vzájemně vnořené. Hledáme nejdelší prefix, který se v grafické reprezentaci vyznačuje tím že je nejmenší (reprezentuje nejmenší rozsah).



Obrázek 3: Binární vyhledávání na intervalech.

Grafickou reprezentaci můžeme snadno převést na tabulku, do které si seřadíme všechny prefixy expandované na konci samými nulami a jedničkami, tak aby vykrývaly celý prostor. V tabulce pak můžeme podobně jako v grafické reprezentaci jednoduše vyhledávat – prefix najdeme pokud zařadíme hledanou adresu mezi dva záznamy, přičemž hledaným výsledkem je právě ten první z nich. Po přípravě tabulky můžeme použít jakoukoliv vyhledávací metodu jako binární vyhledávání nebo B-strom. Nevýhodou této metody je dlouhá doba updatu vyhledávaných prefixů, kde je nutné pokaždé celou databázi přepočítat a seřadit. Výhodou je velmi dobrý poměr použité paměti k použitým prefixům. Počet paměťových přístupů pro IPv4 zhruba odpovídá ostatním metodám nicméně směrem k IPv6 nabízí logaritmický nárůst místo lineárního u metod založených na struktuře trie.

3.8 Binární vyhledávání na prefixech

Tato metoda je založena na myšlence modifikovat tabulku prefixů tak, ať lze použít ověřený, rychlý a úsporný způsob pro uložení a vyhledání konkrétních polí – hashování. Protože (zatím) není známa metoda, která by umožňovala hashovaní na prefixech proměnné délky, využívá tato metoda představená v [9] hashování při hledání prefixů stejné délky. Nevyhledává se přitom v prefixech všech možných délek postupně ale využívá se standardní binární vyhledávání s logaritmickým počtem kroků. V porovnání s ostatními metodami je tato metoda horší v objemu potřebné paměti, délce aktualizace pravidel, ale na druhou stranu nabízí velmi nízký počet přístupů do paměti.

4 Hybridní algoritmus pro IPv6

Většina metod představených v posledních letech se zaměřuje na efektivnější uložení tabulky prefixů, neboť počty pravidel na páteřních prvcích v sítích rychle stoupají a pohybují se v řádu desetitisíců až sta tisíc. Důležitým kritériem samozřejmě zůstává i propustnost, kdy dneska za základ se považuje 10Gb/s (propustnost jednoho portu na aktivním prvku) a také rychlost aktualizace pravidel, neboť internet je stále rychlejší a dynamičtější.

S ohledem na tyto nároky se v poslední době vyvíjejí zejména metody založené na struktuře trie a zaměřují se na optimalizaci zmíněných parametrů. Většina prací je přitom zaměřena na internetový protokol ve stávající verzi 4. Přechod na protokol IPv6 bude přitom znamenat prohledávání 4krát delšího adresového pole. Složitost metod založených na trie je však lineární, a pokud jsme ve struktuře unibit trie museli při vyhledání IPv4 prefixu přistoupit do paměti 32, u IPv6 to bude až 128 přístupů, což je znatelný nárůst. Možným řešením je využít metod, jejichž časová složitost je vzhledem k délce vstupního pole lepší než lineární, např. logaritmická. Důležitým prvkem při přechodu na IPv6 je i struktura 128 bitové adresy, kde prvních 64 bitů tvoří adresu sítě, zbytek je identifikátor rozhraní.

Dalším klíčovým poznatkem získaným analýzou reálných i syntetických množin filtrovacích množin pravidel je značný poměr koncových IP adres vůči adresám podsítí. Pokud se při vyhledávání konkrétních IP adres používá jakákoliv trie metoda, tak vyhledání vždy trvá maximální možný počet taktů a zejména vytvořená trie struktura zahrnující tyto adresy tvoří strom s plnou délkou (32/128 bitů) a při nelokálnosti adres je paměťově neefektivní.

Na základě těchto poznatků se jako vhodné řešení úlohy jeví hybridní LPM algoritmus, který by pracoval následovně. Vstupní množina prefixů je nejprve analyzována a rozdělena na dvě skupiny – adresy podsítí a standardní koncové adresy. Pro koncové IP adresy by se využila efektivní a rychlá metoda – klasické hashování. Díky struktuře IP adresy je přitom možné hashovat pouze podle koncových 64 bitů. Pro dosažení rychlých výsledků i u zbývajících adres podsítí je možné zvolit buď binární vyhledávání na intervalech nebo optimalizovat algoritmus Shape Shifting Tree s využitím rekonfigurace v FPGA tak, aby prefixy způsobující největší paměťovou expanzi byly umístěny do malé asociativní paměti na čipu a zbytek byl vyhledáván standardní metodou.

5 Závěr

Dosavadní práce je soustředěna zejména na detailní prostudování tématiky a návrh možných způsobů řešení pro dosažení vytyčeného cíle – nalezení optimalizovaného LPM algoritmu pro FPGA. Další práce bude soustředěna na nalezení vhodných benchmarkových množin prefixů pro IPv6 a realizaci experimentů pro potvrzení či vyvrácení vhodnosti navržených optimalizací. Metoda, která bude při experimentech vyhodnocena jako nejvýhodnější, bude implementována na FPGA kartách COMBO [1] a bude použita pro zlepšení parametrů 10GE filtru NIFIC [1].

Poděkování

Tento příspěvek vznikl v rámci výzkumného záměru Výzkum informačních technologií z hlediska bezpečnosti, MSM0021630528.

Reference

[1] WWW stránka projektu Liberouter, http://www.liberouter.org (červen 2009).

[2] Viktor Puš: Klasifikace paketů s využitím technologie FPGA, diplomová práce, Brno, FIT VUT, 2008.

[3] V. Srinivasan, G. Varghese, “Faster IP Lookups using Controlled Prefix Expansion,” SIGMETRICS ‘98.

[4] M. Degermark, A. Brodnik, S. Carlsson, S. Pink, “Small Forwarding Tables for Fast Routing Lookups” Proc. ACM SIGCOMM ‘97, , Cannes (14 - 18 September 1997).

[5] S. Nilsson and G. Karlsson, "Fast address lookup for Internet routers",, Proceedings of IFIP International Conference of Broadband Communications (BC'98), Stuttgart, Germany, April 1998.

[6] 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.

[7] H. Song, J. Turner and J. Lockwood: „Shape Shifting Tries for Faster IP Route Lookup“,. Proceedings of the IEEE International Conference on Network Protocols (ICNP), Boston, MA, Nov. 6, 2005, pp. 358-367.

[8] B. Lampson, V. Srinivasan, G. Varghese, “IP Lookups using Multiway and Multicolumn Search,” Proc. IEEE Infocom ‘98.

[9] M. Waldvogel, G. Varghese, J. Turner, and B. Plattner, “Scalable High Speed IP Routing Lookups,” Proc. ACM SIGCOMM ‘97, pp. 25-37, Cannes (14-18 September 1997).