VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

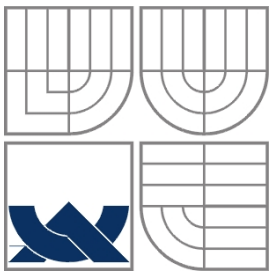
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

ALGORITMY PRO VYSOKORYCHLOSTNÍ SMĚROVÁNÍ V IP SÍTÍCH

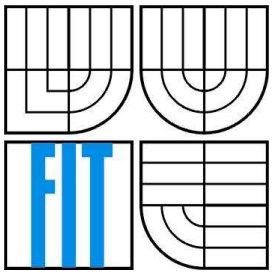
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE IVO HLAVATÝ
AUTHOR

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

ALGORITMY PRO VYSOKORYCHLOSTNÍ SMĚROVÁNÍ V IP SÍTÍCH

ALGORITHMS FOR HIGH-SPEED ROUTING IN IP NETWORKS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — IVO HLAVATÝ
AUTHOR

VEDOUCÍ PRÁCE — Ing. VIKTOR PUŠ
SUPERVISOR

BRNO 2009

Abstrakt

Práce se zabývá simulací algoritmů vyhledávajících v IP sítích nejdelší shodný prefix, konkrétně Trie, Tree Bitmap a Shape Shifting Trie. Algoritmy jsou implementovány softwarově a je zkoumána jejich paměťová a výpočetní náročnost.

Abstract

This work deals with simulation of algorithms finding the longest matching prefix in IP networks, particularly Trie, Tree Bitmap and Shape Shifting Trie. Algorithms are software implemented and explored about their memory and computational performance.

Klíčová slova

Trie, Tree Bitmap, Shape Shifting Trie, paměťové nároky, IP, nejdelší shodný prefix.

Keywords

Trie, Tree Bitmap, Shape Shifting Trie, memory requirements, IP, longest matching prefix

Citace

Hlavatý Ivo: Algoritmy pro vysokorychlostní směrování v IP sítích, bakalářská práce, Brno, FIT VUT v Brně, 2009

Algoritmy pro vysokorychlostní směrování v IP sítích

Prohlášení

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

……
Ivo Hlavatý
18.5.2009

Poděkování

Děkuji vedoucímu mé práce panu Ing. Viktoru Pušovi za jeho připomínky a návrhy, kterými mě podporoval v práci.

Obsah

Obsah 1
1 Úvod 2
2 Principy směrování v IP sítích 3
2.1 IP protokol 3
2.2 Adresování a směrování 4
3 Stromové algoritmy 6
3.1 Trie 6
3.1.1 Rozšíření 7
3.1.2 Hardwarová implementace 7
3.1.3 Vlastnosti získané ze simulace 8
3.2 Tree Bitmap 9
3.2.1 Pseudokód Tree Bitmap algoritmu 10
3.2.2 Rozšíření 11
3.2.3 Hardwarová implementace 12
3.2.4 Vlastnosti získané ze simulace 13
3.3 Shape shifting trie 19
4 Zhodnocení výsledků 22
5 Závěr 24

1 Úvod

Se současným bouřlivým rozvojem internetu dochází k rapidnímu zvyšování požadavků na přenosovou kapacitu, která v dohledné době přestane dostačovat. Důvodem je změna obsahu webových stránek. Dříve stahované textové informace nahradily videa, hudba a P2P sítě. Společnosti vyvíjející síťová zařízení jsou nuceny kromě zvyšování kapacity a rychlosti připojení přecházet na novou verzi IP protokolu, která s sebou nese i vyšší režijní náklady. Nestojíme tedy jen před otázkou vylepšování fyzických spojů, ale také před otázkou optimalizace vyhledávání ve směrovacích záznamech.

Tato práce se zabývá principy vyhledávání směrovacích informací a to konkrétně rozborem vyhledávacích algoritmů Trie, Tree Bitmap a Shape Shifting Trie. První 2 algoritmy jsou naimplementovány ve vyšším programovacím jazyce a je zkoumána jejich paměťová a výpočetní náročnost. Mimo základních verzí algoritmů jsou prezentována i některá jejich rozšíření s dopady na celkové paměťové a výpočetní požadavky. Třetí algoritmus je pouze vysvětlen a jeho vlastnosti jsou převzaty z odborné publikace.

V kapitole 2 jsou vysvětleny základy IP protokolu, jeho verzí a principy směrování v internetu. Kapitola 3 obsahuje popisy jednotlivých algoritmů a výsledků získaných ze simulace. Diskutuje se zde také hardwarová implementace a možnosti rozšíření. Nejvíce experimentování bylo prováděno s Tree Bitmap. V kapitole 4 se srovnávají a hodnotí výsledky simulací. Pátá kapitola obsahuje shrnutí celé práce.

2 Principy směrování v IP sítích

2.1 IP protokol

IP (Internet Protokol) [8] je datový protokol tvořící základ dnešního internetu. Přenáší data přes paketové sítě, ve kterých se zasílané informace rozdělí do bloků – datagramů. Jednotlivé datagramy poté putují po síti nezávisle přes síťová zařízení – převážně směrovače. Není nutné nijak dopředu navazovat spojení. Doručování paketů je prováděno jako best effort – „nejlepší úsilí“. Všechny zařízení po cestě se podle svých možností snaží paket poslat na další zařízení blíže k cíli. Datagramy nemusí k cíli dojít vůbec, mohou dojít vícekrát a v jiném pořadí než byly poslány. Internet protokol je tedy nespolehlivou službou. Spolehlivost přenosu zajišťují vyšší vrstvy. Pokud se pakety ztrácí často, je úbytek výkonu pozorovatelný, občasný výpadek paketu nemá pozorovatelný efekt.

Struktura paketu [6] se skládá z hlavičky a vlastních přenášených dat. Minimální délka datagramu je 20 bytů (20-bytů hlavička + 0 bytů data) a maximální délka je 65,535 bytů – 16 bitové slovo. Některá síťová zařízení nemusí zvládat velké pakety a dochází k jejich fragmentaci na menší části.

2.1.1.1 Verze IP

Během vývoje Internet Protokolu vzniklo několik verzí označovaných IPvX [8]. Verze číslo 0 až 3 byly vývojové verze pro IPv4 používané mezi lety 1977 a 1979. Verze číslo 5 je experimentální streamovací protokol – Internet Stream Protocol. Verze číslo 6 až 9 byly navrhovány, aby nahradily stávající nejrozšířenější protokol IPv4, kterému dochází adresní prostor (32-bitové adresy). Verze číslo 6 byla vybrána jako oficiální následník IPv4 a v současné době startuje přechod na šestou verzi, která má téměř nevyčerpatelný adresní prostor (128-bitové adresy).

2.1.1.2 Formát IP adresy

Formát adresy pro IPv4 [6] se zapisuje po osmi oktetech v decimální formátu. Např. 192.168.1.1 .Teoreticky obsahuje 2^{32} jedinečných adres. Ovšem některé z nich jsou rezervované pro multicast, experimentální účely a adresy privátních sítí. Původně určoval první oktet příslušnost k dané podsíti (256 podsítí), s rostoucím počtem organizací a uživatelů připojujících se k internetu došlo k zavedení tříd A,B,C,D,E (třídní adresování) a v roce 1993 zavedení techniky CIDR (beztřídní adresování). V beztřídním adresování určuje prefix sítě libovolná délka bitů a označuje se jako maska podsítě(narozdíl od tříd, kde byla délka masky zvolena pevně).

Formát adresy pro IPv6 se zapisuje jako osm skupin čtyř hexadecimálních číslic oddělených dvojtečkou. Např. 2001:0db8:85a3:0000:0000:8a2e:0370:7334. Teoreticky obsahuje 2^{128} unikátních

adres. Prvních 64 bitů určuje masku podsítě a druhá polovina adresu hosta, která je generována z mac adresy.

2.2 Adresování a směrování

Každé síťové rozhraní v internetu má přidělenu jedinečnou IP adresu pomocí které komunikuje se zbytkem sítě. V každém datagramu je pak uvedena IP adresa odesílatele a příjemce. Směrováním [7] je označováno hledání cest v počítačových sítích. Úkolem je dopravit paket určenému zařízení co nejlepší cestou.

Směrovací tabulka (routing table) je základní datovou strukturou pro směrování. Obsahuje cílovou adresu a pravidla, podle kterých se rozhoduje, jaká akce se s pakety provede.

- **Cílová adresa** týkající se dotyčného záznamu může být buď adresa individuálního počítače, nebo může jít o cíl definovaný prefixem, tedy určitým počtem počátečních bitů adresy. Adresa s prefixem mívá podobu např. 213.211.56.65/22. Hodnota před lomítkem je adresa cíle, hodnota za lomítkem pak určuje počet významných bitů adresy. Uvedenému prefixu tedy vyhovuje každá adresa, která má v počátečních 22 bitech hodnotu 213.211.56. Tedy adresy 213.211.56.0 - 213.211.59.255
- **Pravidla** rozhodují, co provést s datagramy, jejichž adresa vyhovuje prefixu. Pokud je dotyčný stroj spojen přímo s adresátem, doručí se paket přímo jemu. V opačném případě je předán některému ze sousedních směrovačů, na kterém se celá akce znovu zopakuje.

Směrovací rozhodování probíhá pro každý přicházející datagram samostatně. Jeho cílová adresa se vyhledá ve směrovací tabulce. Z ní se vyberou všechny vyhovující záznamy (jejichž prefix vyhovuje cílové adrese datagramu) z nichž se použije ten s nejdelším prefixem. Toto pravidlo vyjadřuje přirozený princip, že konkrétnější záznamy (jejichž prefix je delší, tedy přesnější) mají přednost před obecnějšími.

2.2.1.1 Směrovací algoritmy a protokoly

Vznik a udržování směrovací tabulky je řízen přesnými pravidly a formáty neboli směrovacími protokoly [7]. Směrování může být statické, kde se tabulka vytvoří a dále se nemění. Nebo může být dynamické, kdy se routery učí topologii sítě a při výpadku zařízení jsou schopny upravit aktuálně své směrovací tabulky a najít pro paket jinou cestu. Mezi směrovací protokoly patří např. *RIP* – jednoduchý, v malých sítích nebo *OSPF* – ve velkých sítích a autonomních systémech.

2.2.1.2 Směrování v internetu

Celý internet je pro svoje obrovské rozměry rozdělen hierarchicky do více úrovní a tvoří ho tzv. autonomní systémy, které jsou vzájemně propojeny přes páteřní sítě. Je zřejmé, že jak postupujeme v hierarchii směrem výše, jsou kladeny vyšší nároky na přenosovou rychlost a stabilitu.

Příkladem autonomního systému může být síť jednoho poskytovatele internetu. Vůči zbytku internetu má jednotnou směrovací politiku a používá interní směrovací protokoly (Interior Gateway Protocol, IGP). Hlavním cílem těchto protokolů je pružnost a rychlá reakce na změny. Mezi zástupce IGP patří RIP, EIGRP, ISIS a OSPF.

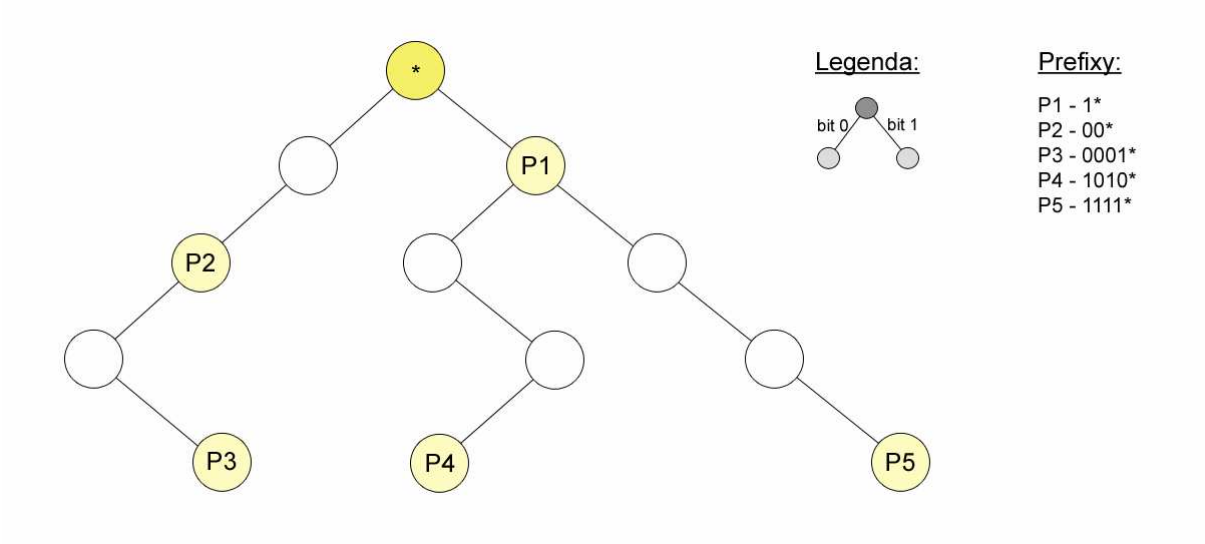
Předávání směrovacích informací mezi autonomními systémy zajišťují externí směrovací protokoly (Exterior Gateway Protocol, EGP). Vyznačují se vysokou stabilitou a pomalou reakcí na změny. Oproti IGP jsou podstatně konzervativnější. Prakticky jediným používaným protokolem z této skupiny je BGP [7].

3 Stromové algoritmy

Mezi základní algoritmy pro vyhledání nejdelšího shodného prefixu patří Trie, ta tvoří základ pro Tree Bitmap a Shape Shifting trie. Všechny tyto algoritmy pracují na principu sestavení routovací tabulky do stromu a mohou využívat mnoho vylepšení, které výpočet velmi urychlí. Velkou roli zde také hraje arita (střída), neboli kolik bitů z adresy zpracovává zařízení zároveň. Protože tyto operace nejsou prováděny softwarově, ale hardwarově, nemůžeme navrhnout libovolnou koncepci a musíme se řídit zákonitostmi a omezeními logických obvodů.

3.1 Trie

Nebo také prefixový strom je základní algoritmus pro vyhledání nejdelšího shodného prefixu. Využívá stromovou strukturu, která obsahuje informace přímo ve své konstrukci a to dva ukazatele na další uzly stromu a označení zda uzel tvoří platný prefix respektive číslo tohoto prefixu. V průběhu výpočtu se zpracovávají bity postupně po jednom (tzv. Unibit trie) a podle hodnoty bitu se rozhoduje, kterým podstromem se bude pokračovat. Algoritmus probíhá tak dlouho, dokud je možné dále postupovat stromem (potřebný strom existuje), nebo jsme již zpracovali všechny bity. Při procházení se uchovává poslední platný prefix, který je na konci předán jako výsledek. Časová složitost vyhledávání je lineární. Nevýhoda algoritmu je, že potřebuje 32 náhodných přístupů do paměti v nejhorším případě [1][3].



Obr 3.1: Jednoduchý příklad Trie.

3.1.1 Rozšíření

Toto byla základní Trie, existuje však mnoho rozšíření, která urychlují výpočet či šetří pamětí. Naším cílem je upravit algoritmus tak, aby zaujímal co nejméně paměti a prováděl co nejméně iterací.

Například rozšíření CPE [1] (Controlled Prefix Expansion) urychluje výpočet. Hlavní myšlenka je zpracování několika bitů zaráz, tzv. střída. V nejhorším případě se tím sníží počet náhodných přístupů do paměti tolikrát, po kolika bitech načítáme data. Pokud tedy například načítáme v jednom okamžiku 4 bity, v nejhorším případě budeme 8x přistupovat do paměti. Ovšem toto rozšíření není zadarmo a stojí nás větší množství paměti. Problém nastává s prefixy, které nejsou násobky střídy. Principem je expandování prefixu do všech možných X-bitových hodnot. Vezměme si délku kroku 4 bity a strom jako na obrázku 3.1. Rozšíříme uzel P1 – 1* na 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111 a reprezentujeme P1 osmi prefixy. Avšak prefixy 1010(P4) a 1111(P5) už existují, proto mají přednost před nově expandovanými. Obecně tedy expandujeme každý prefix, který není násobkem střídy do tolika prefixů, které tuto podmínku splňují a expandované uzly kolidující s již existujícími delšími uzly zahazujeme.

Další rozšíření se zabývají redukcí potřebného místa v paměti pro sestavený strom. Např. metoda Leaf Pushing [1] ukládá do uzlu stromu buď ukazatel na další podstrom nebo ukazatel do tabulky výsledků. Pokud obsahuje uzel obě tyto informace, „protlačí“ se ukazatel do tabulky výsledků níže stromem. Pro největší efektivitu algoritmu je třeba využít obě (a další) rozšíření zároveň.

3.1.2 Hardwarová implementace

Ačkoliv se tento algoritmus ve směrovačích v této podobě nepoužívá, předpokládejme na chvíli, že ano a diskutujme jeho nároky. Základními požadavky dneška je rychlost. Při rychlostech spojů 10 Gbps potřebujeme speciální paměti. Tyto paměti jsou ovšem velmi drahé, malé(desítky MB) a náročné na energii [5]. Pokud implementujeme výše zmíněný algoritmus, expandování uzlů je velmi nepříjemná vlastnost, která plýtvá paměťmi. Potřebný alokovaný prostor roste s větším počtem zpracovaných bitů v jedné iteraci. I když existuje mnoho dalších optimalizací, které redukují paměťové nároky, pořád není Trie vhodná pro hardwarovou implementaci.

3.1.2.1 Typy pamětí používaných ve směrovačích

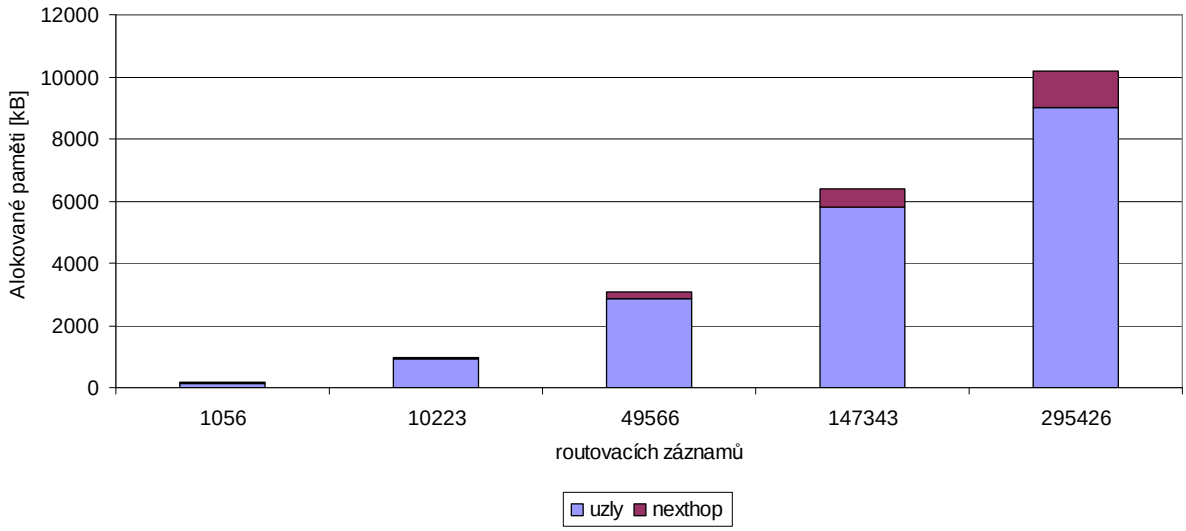
V dnešní době se v páteřních a hraničních oblastech uplatňují např. paměti QDR™-II– Quad Data Rate. Jedná se o další generaci SRAM pamětí pro síťový průmysl, která disponuje oddělenými vstupy a výstupy operující každý na DDR. Tyto paměti se díky konkurentnímu přístupu hodí nejvíce do aplikací, u kterých se poměr čtení/zápis blíží 1. QDR™- III paměti pracují na 500+ MHz a dosahují kapacity až 288 MB [4].

3.1.3 Vlastnosti získané ze simulace

Simulace probíhala se skutečnými routovacími záznamy z protokolu BGP [9]. Originální tabulka obsahovala 295426 záznamů, pro potřeby testování byly pomocí této tabulky vytvořeny sady routovacích záznamů - řádově 1000, 10 000, 50 000 a 150 000 záznamů. Tyto sady se snaží zachovat rozsah adres a byly sestaveny vybráním pouze N-tých záznamů, kde N je násobkem vhodně zvolené konstanty(2,6, ...). Program zkoumal paměťové nároky a výpočetní náročnost algoritmu Trie (Unibit Trie). Protože pracujeme se softwarovou implementací s využitím prvků vyšších programovacích jazyků (C++), nemá smysl uvádět čas strávený výpočtem, ale jediným směrodatným měřítkem nám bude počet kroků programu.

alokovaných nexthop adres	1056	10223	49566	147343	295426
velikost alokovaných nexthop adres [kB]	4,22	40,89	198,26	589,37	1181,7
počet alokovaných uzlů	12174	77520	238827	484743	749788
velikost alokovaných uzlů [kB]	146,09	930,24	2865,92	5816,92	8997,46
celková velikost alokované paměti (32-bitový ukazatel) [kB]	150,31	971,13	3064,19	6406,29	10179,16
celková velikost alokované paměti (co nejmenší ukazatel) [kB]	26,98	223,51	834,76	2072,16	3757,81

Tabulka 3.2: Paměťové nároky Trie



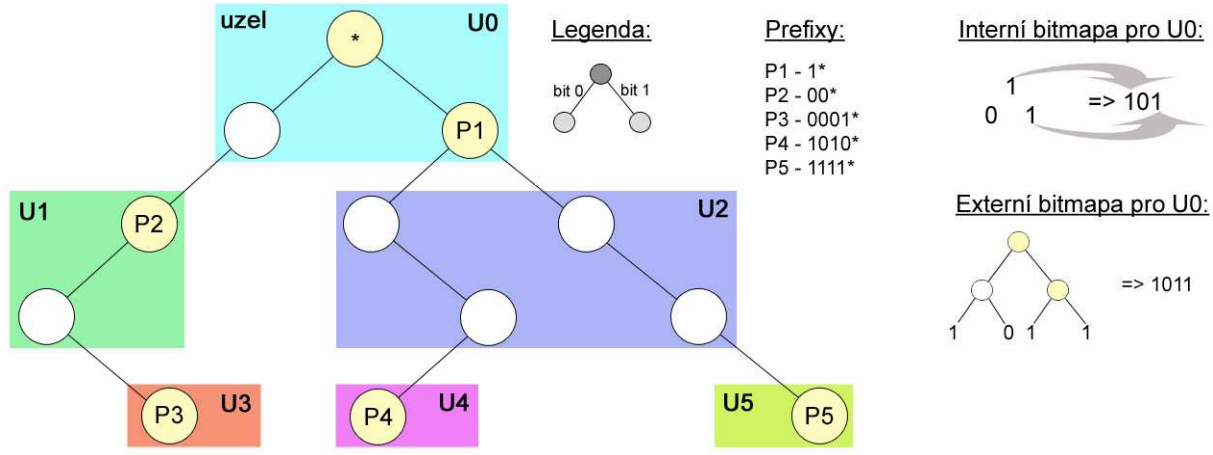
Obr. 3.3: Velikost alokované paměti pro jednotlivé sady se znázorněným podílem směrovacích záznamů a uzlů.

Při simulaci provedl program průměrně 23,4 kroku na vyhledání každého jediného záznamu a nezáleželo na velikosti směrovací tabulky. Vzorek simulovaného datového toku obsahoval 2 různé IP adresy z každé podsítě příslušné sady routovacích záznamů.

3.2 Tree Bitmap

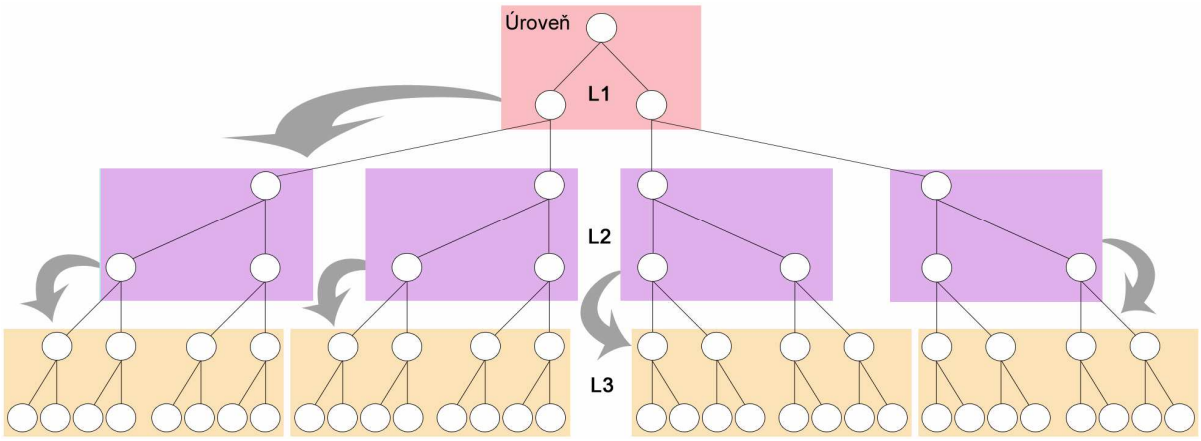
Je další z algoritmů na vyhledání nejdelšího shodného prefixu. Jedná se o stromový algoritmus vycházející z Trie. Vyznačuje se především nízkými nároky na kapacitu paměti a snadnou hardwarovou realizací. Základní myšlenky tohoto algoritmu [1]:

- Jednotlivé podstromy jsou v paměti uloženy za sebou, to umožňuje uchovávat ukazatel na první podstrom a zbytek se vypočítá jako offset. Obr. 3.5. Tento přístup redukuje počet ukazatelů a hlavně snižuje velikost uzlů. Nevýhodou je, že alokátor paměti musí umět pracovat s různě velkými uzly.
- Pro každý uzel máme 2 bitmapy – interní a externí. Interní bitmapa reprezentuje strom v linearizovaném formátu. Obsahuje bit 1 tam, kde je platný prefix. Externí bitmapa obsahuje bity ke všem možný teoretickým podstromům. Bit 1 značí, že je možné postupovat dále ve vyhledávání nejdelšího shodného prefixu.
- Ponechat velikost shluku podstromů(uzlů) co nejmenší, aby se redukovala velikost čtené paměti pro danou střídu. Uzel se skládá z interní a externí bitmapy, ukazatelem na uzel potomka, ukazatelem do pole nexthop adres – routovací tabulky.
- Tím, že nexthop ukazatele umístíme do jiného pole, potenciálně potřebujeme 2 přístupy do paměti, jeden pro průchod stromem a jeden pro nahlédnutí do pole výsledků. Tento problém řeší technika líného vyhledávání, kdy si při průchodu stromem ukládáme ukazatel do paměti výsledků a přistupujeme sem až po skončení algoritmu.



Obr. 3.4 : Jednoduchý příklad Tree Bitmap s délkou kroku 2.

Vyhledávací algoritmus je vcelku jednoduchý. Začneme kořenovým uzlem. Můžeme si to představit tak, že sestavíme z interní bitmapy dočasně strom. Projdeme na pozici Q, kde uzel přechází do dalšího uzlu. Nahlédneme do externí bitmapy a zjistíme jestli se může pokračovat dále – v případě bitu 1. Poté spočítáme počet bitů 1 (označme počet K) nalevo od Q v externí bitmapě. Uzel, na který budeme pokračovat spočítáme U = y + (K * B), kde y je ukazatel na uzly potomků, B je velikost uzlu. Předtím, než budeme pokračovat k následujícímu uzlu, zkontrolujeme interní bitmapu, jestli se v ní nenachází prefixy korespondující s posloupností bitu na vstupu. Pokud ano, pouze si uložíme pozici nexthop adresy ve směrovací tabulce, k přečtení adresy dojde až po ukončení algoritmu. V následujících uzlech opakujeme stejný postup. Algoritmus je ukončen, pokud na místě pokračovatele uzlu v externí bitmapě je bit 0.



Obr. 3.5 Ukázka práce s ukazateli a uspořádání paměti při zaplnění celého stromu a střídě 2. Šipka reprezentuje ukazatel a barevné plochy určují objem paměti adresovaný tímto jediným ukazatelem.

Nyní si ukážeme praktickou ukázku. Mějme strom z obr. 3.4 a na vstupu posloupnost bitů 1001. Začneme v kořenovém uzlu U0, posloupnost bitů nás dostane na třetí bit v externí bitmapě -> bit 1, tzn. pokračování v dalším uzlu. Do proměnné si uložíme pozici P1 v routovací tabulce. Spočítáme počet jedniček nalevo od třetího bitu v externí bitmapě a přesuneme se podle vypočítaného offsetu na uzel U2. Znovu nahlédneme do externí bitmapy U2 uzlu a zjistíme, že na první pozici je bit 0, tudíž není možné pokračovat stromem. Prohlédneme tedy ještě interní bitmapu, jestli se v ní nevyskytuje prefix(nevyskytuje) a přečteme z routovací tabulky nexthop adresu pro prefix P1 a algoritmus ukončíme.

3.2.1 Pseudokód Tree Bitmap algoritmu

Tato sekce popisuje pseudokód vyhledávacího algoritmu. Níže jsou uvedeny datové struktury, které jsou určeny pro obecnou střídu. Velikost polí je závislá na zvolené třídě od níž se odvíjí také počet uzlů ve stromu. Pseudokód pracuje s pamětí obsahující 3 pole. První pole - pole_strida[] obsahuje rozdělenou vyhledávanou IP adresu po částech velikosti střídy. Druhé pole – pole_uzlu[] tvoří

všechny uzlu stromu poskládané v paměti za sebou. Třetí pole – routovaci_tabulka[] je pole obsahující směrovací záznamy (nexthop záznamy). Struktura uzlu je následující:

```
uzel{
  externi_bitmapa,
  interní_bitmapa,
  uzel_potomek, /* ukazatel do pole uzlu */
  routovaci_tabulka /* ukazatel do routovací tabulky */
}
```

Algoritmus:

```
uzel:= pole_uzlu[0]; /* uzel je aktuální zkoumaný uzel, začíná se u
kořenového uzlu*/
i:= 1; /* i je index do pole_strida[]*/
do forever
  if (funkceStrom(uzel.interni_bitmapa, pole_strida[i]) is not equal null)
  then  /* existuje delší shodný prefix, uložíme si na něj ukazatel */
    NejdelsiPrefix:= uzel.routovaci_tabulka +
    Jednicky(uzel.interni_bitmapa, funkceStrom(uzel.interni_bitmapa,
    pole_strida[i]));
  if (externi_bitmapa[pole_strida[i]] = 0)
  then /* neexistuje další uzel */
    NextHop:= routovaci_tabulka[NejdelsiPrefix]; /* technika líného
    přístupu přistupuje do paměti s routovacími záznamy až nakonec*/
    break; /* konec vyhledávání */
  else /* existuje další uzel, přesun k potomkovi */
    uzel:= pole_uzlu[uzel.uzel_potomek + Jednicky (uzel.externi_bitmapa,
    pole_strida[i])];
  i=i+1; /* index pro další sadu bitů (střída) */
end do; [1]
```

V algoritmu se vyskytují dvě další funkce. FunkceStrom() pomocí interní bitmapy nalezne pozici nejdelšího shodného prefixu v daném uzlu a vrátí ukazatel na tento výsledek. Funkce Jednicky jednoduše sečte počet bitů 1 v bitovém vektoru po daný index. Algoritmus skončí, pokud již nejde dále stromem sestupovat na další uzel (v externí bitmapě je bit 0 na daném indexu) a současně se přistoupí do paměti výsledků pro nexthop adresu.

3.2.2 Rozšíření

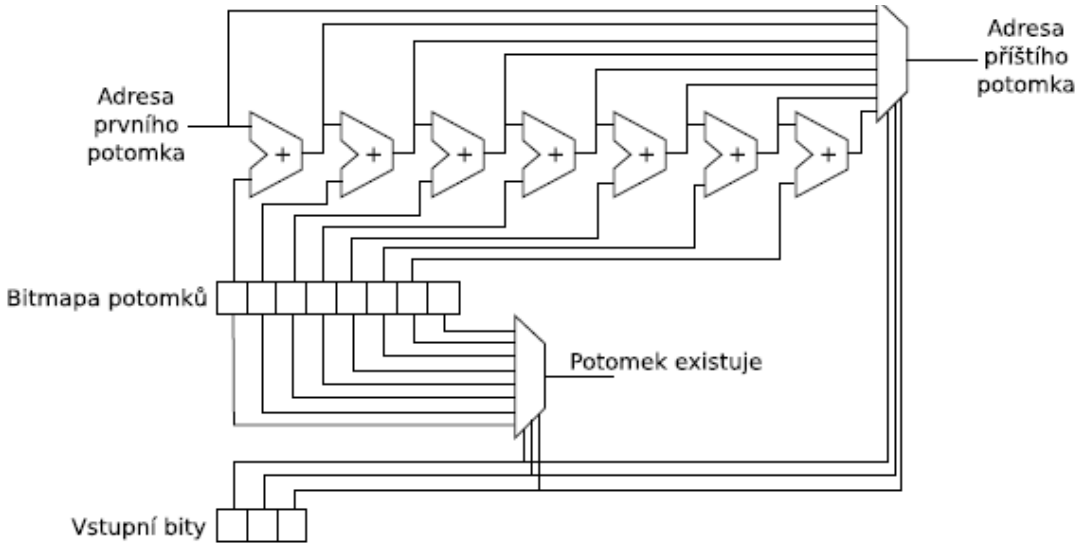
Na Tree Bitmap existuje celá řada rozšíření. Mezi ty základní řadíme např. Optimalizaci koncových uzlů [1]. Konkrétně jde o problém s adresami, jejichž maska je dělitelná střídou. Dostáváme se do situace, kdy se pro uložení jediného ukazatele musí vytvořit úplně nový uzel a tento ukazatel se

umístí do kořene. Veškeré ostatní položky zůstanou nevyužity. Můžeme sice analyzovat adresy a upravit střídu tak, aby byla dělitelná s co nejméně záznamy, ale toto řešení není moc efektní. Lepším řešením je již zmiňovaná Optimalizace koncového uzlu. Předpokládejme tedy uzel, který nemá žádné potomky (v externí bitmapě nejsou bity 1). V tomto případě si vyrobíme speciální uzel nazvaný „koncový“ a externí bitmapu substituujeme jako pokračovaní interní bitmapy. Tímto vylepšením jsme získali prostor pro uložení bitů. Významný důvod pro použití těchto uzlů je také ten, že pro typickou střídu 4 a 8 potřebuje každá 32 bitová maska svůj vlastní uzel čili další úroveň. Kromě plýtvání místem nám vyvstává důležitější problém - zvýšení vyhledávací doby v nejhorším případě.

Téměř každý IP vyhledávací algoritmus dokážeme urychlit využitím Initial Array optimalizace [1]. Hlavní myšlenkou toho rozšíření je pokrytí začátku stromu permanentním vhodně zvoleným polem. Příkladem nám může být střída 4 a inicializační pole o velikosti 8. Prvních 8 bitů IP adresy použijeme jako index do pole o velikosti 256 (2^8). Ačkoliv alokujeme trvale prostor, který z části nevyužijeme (až 8 kB), je to rozumná cena za méně kroků vyhledávacího algoritmu. Navíc v hardwarové implementaci můžeme umístit inicializační pole přímo do paměti na čipu, i když zbytek záznamů bude mimo čip.

3.2.3 Hardwarová implementace

Tento algoritmus se velmi hodí pro hardwarovou implementaci. Ačkoliv je třeba velké množství iterací při práci s bitmapami, ve skutečnosti je to v hardwaru jen otázka jednoduchých kombinačních obvodů (obr. 3.6). Nutno zdůraznit, že střída (počet bitů zpracovaných zaráz) se musí volit uvážlivě. Nejlépe jako mocnina dvou, ideálně nejvýše 8 bitů. Logické obvody mohou kvůli fyzikálním omezením zpracovat jen 256 bitovou mapu. Díky paralelnímu přístupu ve výpočtech dosahuje algoritmus opravdu vysokých rychlostí. Střída vyšší než 8 bitů by už pak musela být zpracovávána spíše softwarově, což by bylo prakticky nepoužitelné.



Obrázek 3.6: Obvod pro výpočet adresy potomka (příštího podstromu) pro střídu 3 [3].

3.2.4 Vlastnosti získané ze simulace

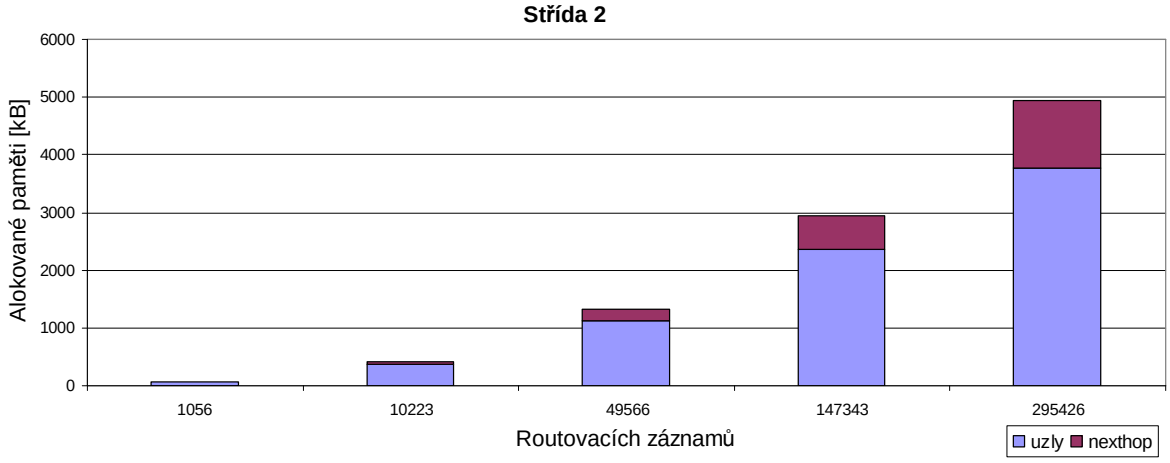
Simulace probíhala se skutečnými routovacími záznamy z protokolu BGP [9]. Originální tabulka obsahovala 295 426 záznamů, pro potřeby testování byly pomocí této tabulky vytvořeny další sady routovacích záznamů - řádově 1000, 10 000, 50 000 a 150 000 záznamů. Tyto sady se snaží zachovat rozsah adres a byly sestaveny vybráním pouze N-tých záznamů, kde N je násobkem vhodně zvolené konstanty(2,6, ...). Program zkoumal paměťové nároky a výpočetní náročnost algoritmu Tree Bitmap v jeho základní verzi a ve verzi s rozšíření Initial Array. Velikost střídy byla zvolena 2, 4, 8 (pro obě verze) a 16 pro základní. I když je střída 16 prakticky nevyužitelná, jsou její vlastnosti zajímavým přínosem pro analyzované vlastnosti algoritmu. Testování proběhlo pro každou střídu, každou testovací sadu a verzi algoritmu zvlášť.

Výsledky paměťové náročnosti jsou uvedeny v tabulce 3.7, počet kroků programu zobrazuje obr. 3.16. Protože se podstatná část uzlů skládá z ukazatelů, můžeme ovlivnit velikost uzlů vhodně zvolenou velikostí ukazatele. Např. pro 10 000 směrovacích záznamů nebudeme potřebovat 32-bitový ukazatel, ale jen 14-bitový. Tabulka tedy obsahuje velikost pro základní 32-bitový ukazatel a velikost pro nejmenší možný ukazatel. Dále se předpokládá, že směrovací informace a uzly v tabulce jsou umístěny v různých pamětech, tedy velikosti obou ukazatelů se mohou lišit. Tímto na první pohled potenciálním plýtváním místem (umístění obou ukazatelů ve stejné paměti by znamenalo menší společný ukazatel) podporujeme hardwarovou implementaci. Navíc rozdíl je minimální. Pokud není uvedeno jinak, jsou velikosti alokované paměti v grafech uváděny s 32-bitovým ukazatelem.

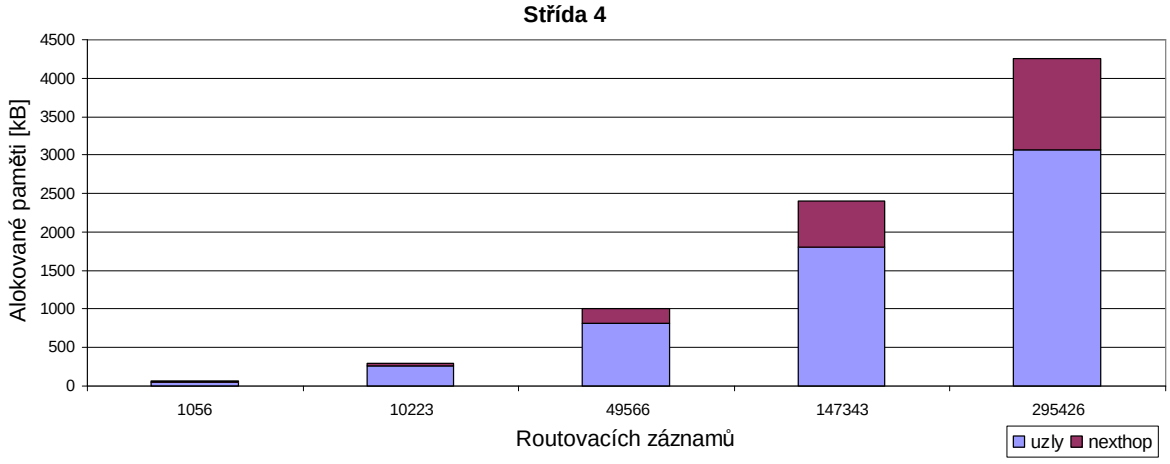
		uzly		celková paměť	
směrovací informace	střída	počet	velikost [kB]	ukazatel 32 bitů [kB]	nejmenší ukazatel [kB]
1056 adres / 4,22 [kB]	2	6212	55,91	60,13	29,07
	4	3234	38,81	43,03	26,46
	8	1685	121,32	125,54	116,70
	16	981	16080,55	16084,78	16079,50
10223 adres / 40,89 [kB]	2	39983	359,85	400,74	230,81
	4	21064	252,77	293,67	201,51
	8	11329	815,67	856,58	805,60
	16	5708	93565,54	93606,43	93580
49566 adres / 198,26 [kB]	2	125655	1130,90	1329,16	842,25
	4	67981	815,77	1014,04	750,61
	8	38057	2740,10	2938,37	2786,14
	16	11424	187262,20	187460,47	187412
147343 adres / 589,37 [kB]	2	262032	2358,29	2947,66	2030,55
	4	150882	1810,58	2399,96	1871,87
	8	94749	6821,93	7411,30	7067,83
	16	16488	270271,30	270860,67	270797,00
295426 adres / 1181,7 [kB]	2	418590	3767,31	4949,01	3588,60
	4	256117	3073,40	4255,11	3390,71
	8	177865	12806,28	13987,98	13387,70
	16	21657	355001,50	356183,25	356102,00

Tabulka 3.7: Využití paměti pro všechny testované sady a střídy

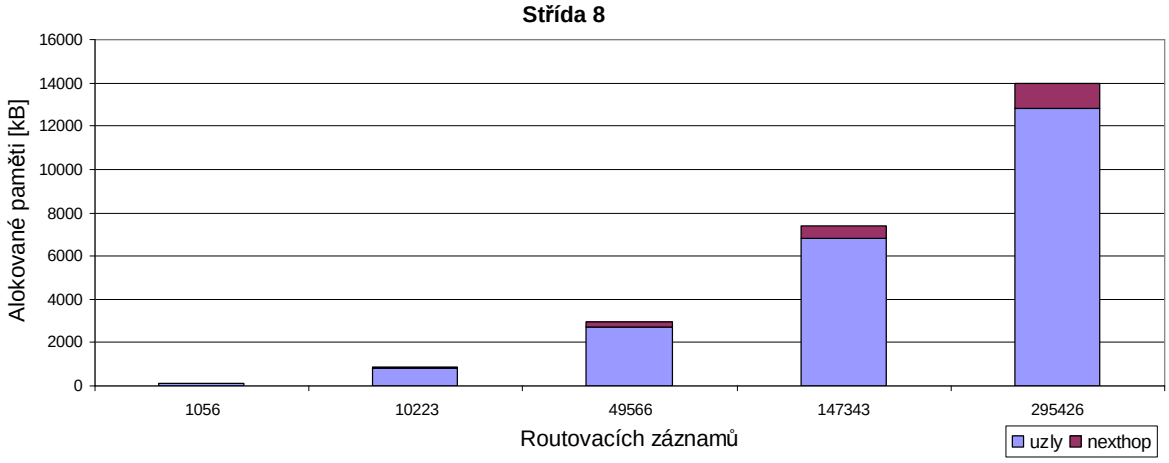
Na obrázcích 3.8 – 3.11 jsou zobrazeny velikosti alokovaných uzlů a směrovacích záznamů. Každý graf obsahuje skladbu paměti pro jednotlivou střídu a všechny testovací sady. Zajímavým ukazatelem je podíl „užitečných“ informací (směrovacích záznamů) a „neužitečných“ informací (uzlů stromu). S výjimkou střídy 4 se k routovacím informacím nabaluje čím dál více uzlů stromu a ve střídě 16 tvoří naprostou většinu alokované paměti. Co se týče střídy 4, zde nám vychází nejlepší poměr velikost_stromu/střída. S větším počtem směrovacích informací se tento poměr zvyšuje. Nejlépe je to vidět na obrázku 3.14 . Rozdíly jsou způsobeny využíváním interní bitmapy. Střída velikosti 4 obsahuje pouze 16 bitů, kdežto střída velikosti 8 už 256 bitů.



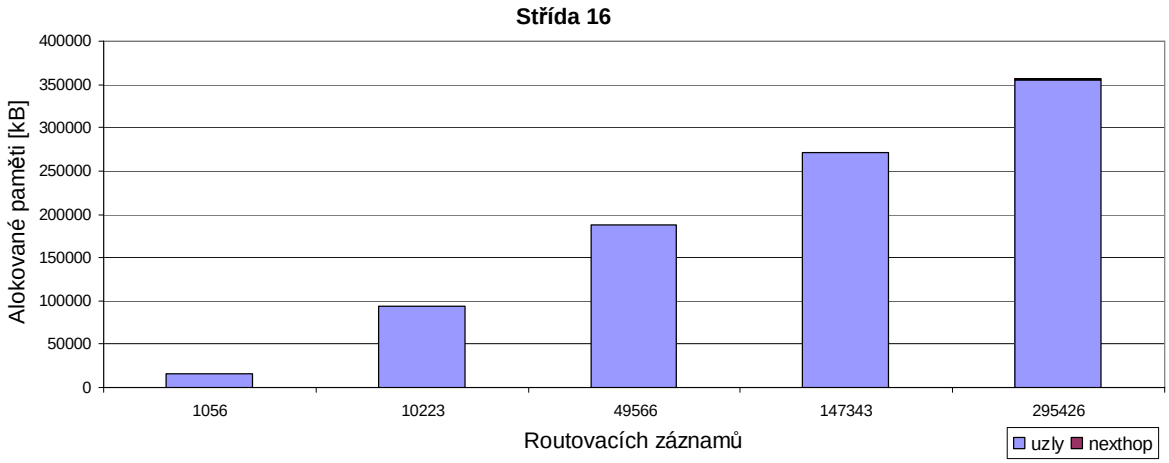
Obrázek 3. 8: Velikost alokované paměti pro střídu 2 s rozdělením na uzly a nexthop adresy.



Obrázek 3. 9: Velikost alokované paměti pro střídu 4 s rozdělením na uzly a nexthop adresy.



Obrázek 3. 10: Velikost alokované paměti pro střídu 8 s rozdělením na uzly a nexthop adresy.



Obrázek 3. 11: Velikost alokované paměti pro střídu 16 s rozdělením na uzly a nexthop adresy.

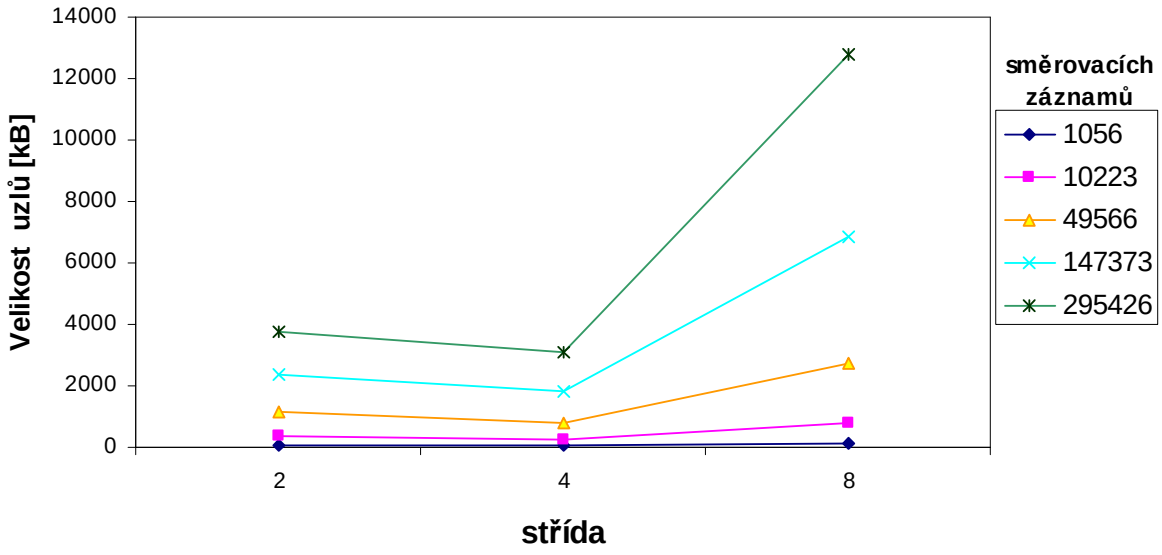
Tabulky 3.12 a 3.13 obsahují statistické informace o průměrném počtu bitů 1 v interních a externích bitmapách. Velikosti bitmap jsou rovny $2^{\text{střída}}$, přičemž interní bitmapa má o jeden bit méně.

záznamů	1056	10223	49566	147343	295426
střída 2	0,17	0,26	0,39	0,56	0,71
střída 4	0,33	0,49	0,73	0,98	1,15
střída 8	0,63	0,9	1,3	1,55	1,66
střída 16	1,1	1,78	4,3	8,87	13,53

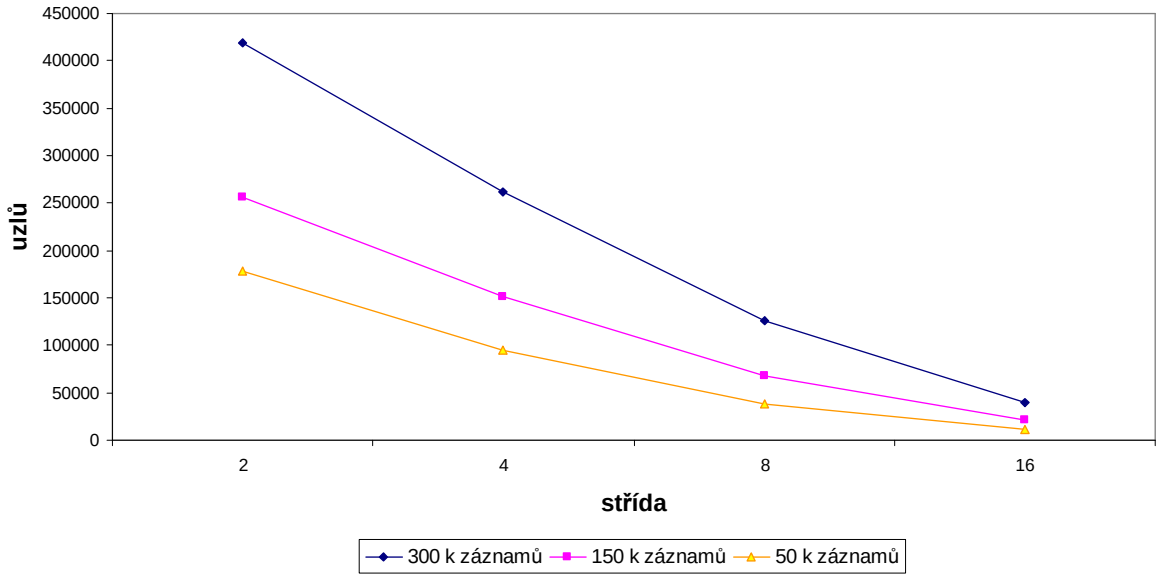
Tabulka 3.12: průměrný počet bitů 1 v interní bitmapě.

záznamů	1056	10223	49566	147343	295426
střída 2	1	1	1	1	1
střída 4	1	1	1	1	1
střída 8	0,92	1	1	1	1
střída 16	0	0	0	0,01	0,01

Tabulka 3.13: Průměrný počet bitů 1 v externí bitmapě.

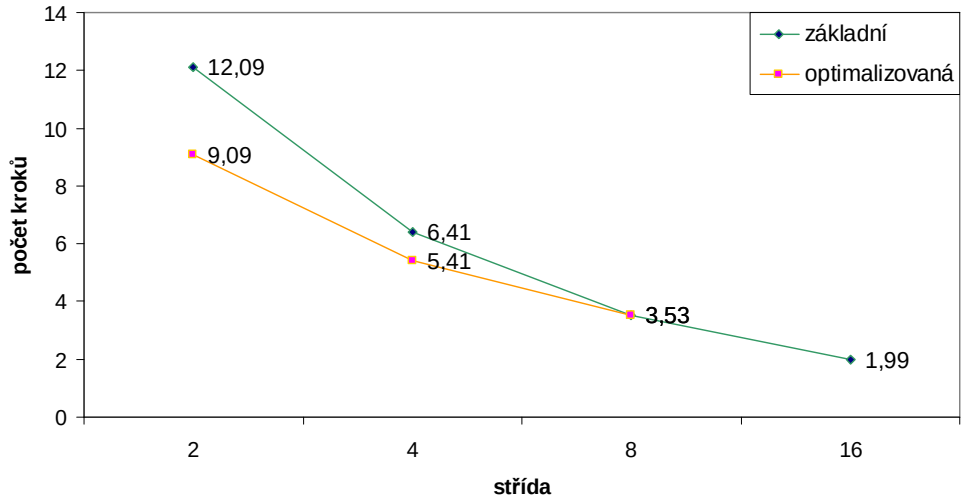


Obrázek 3. 14: Velikost alokované paměti pro uzly v závislosti na zvolené střídě. (Střída 16 není pro svoje obrovské paměťové nároky do grafu zavedena).



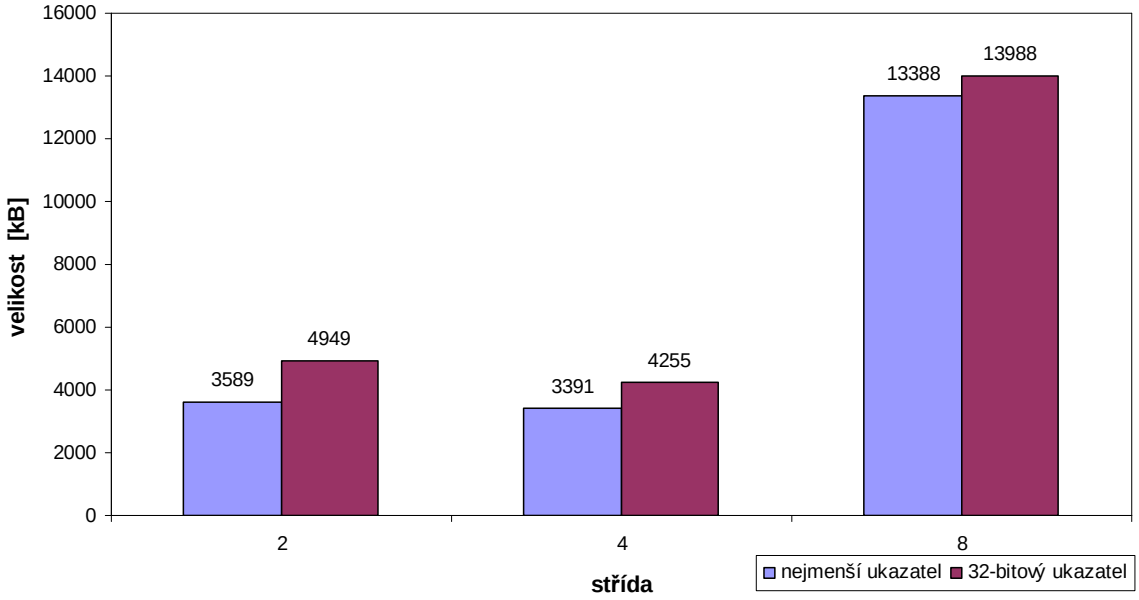
Obrázek 3. 15: Graf sledující počet alokovaných uzlů v závislosti na zvolené střídě a počtu záznamů.

Obrázek 3.16 znázorňuje graf počtu kroků pro základní a optimalizovanou verzi. Rozšířená verze nemá pro střídu 16 žádný smysl, proto není v grafu zobrazena. Střída 8 je logicky stejně výpočetně náročná. Ovšem v hardwaru bude rychlejší. Adresu následujícího uzlu získáme přímo z paměti na čipu a nemusíme provádět celý jeden krok výpočtu.



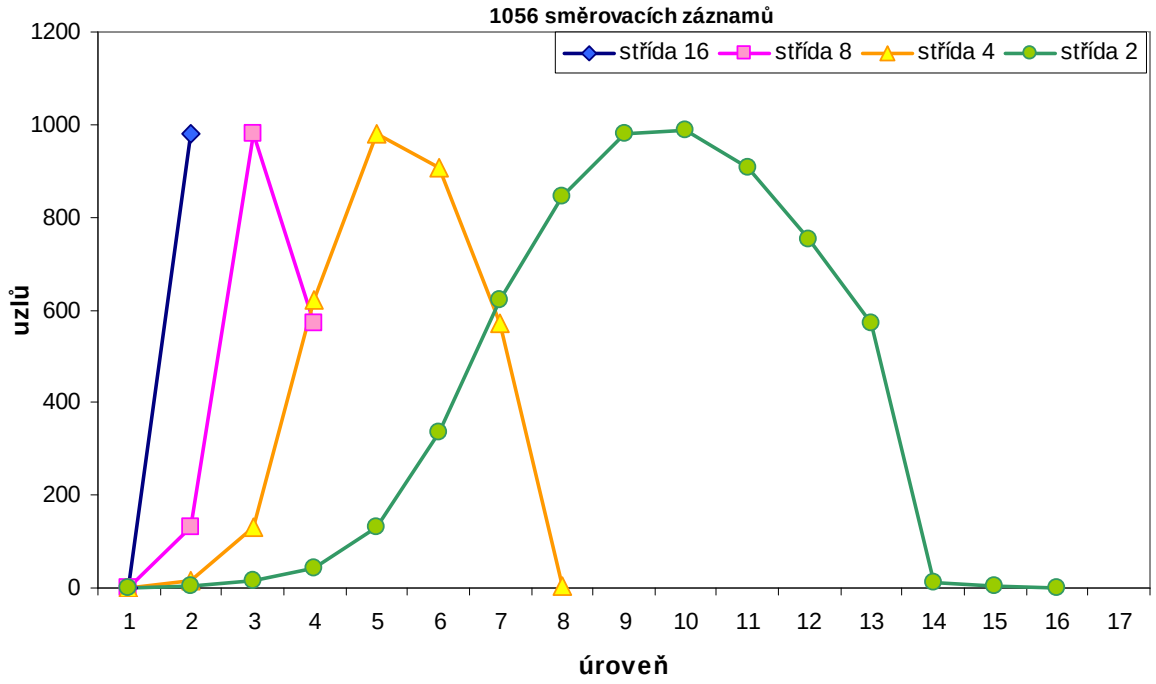
Obrázek 3. 16: Průměrné počty kroků algoritmu při vyhledávání nejdelšího shodného prefixu.

Obrázek 3.17 zobrazuje alokovanou paměť pro co nejmenší ukazatel a pro standardní 32-bitový ukazatel. Z grafu jednoznačně vyplívá, že největší úspory dosáhneme při malé střídě a tedy větším počtu uzlů (kde můžeme ušetřit).

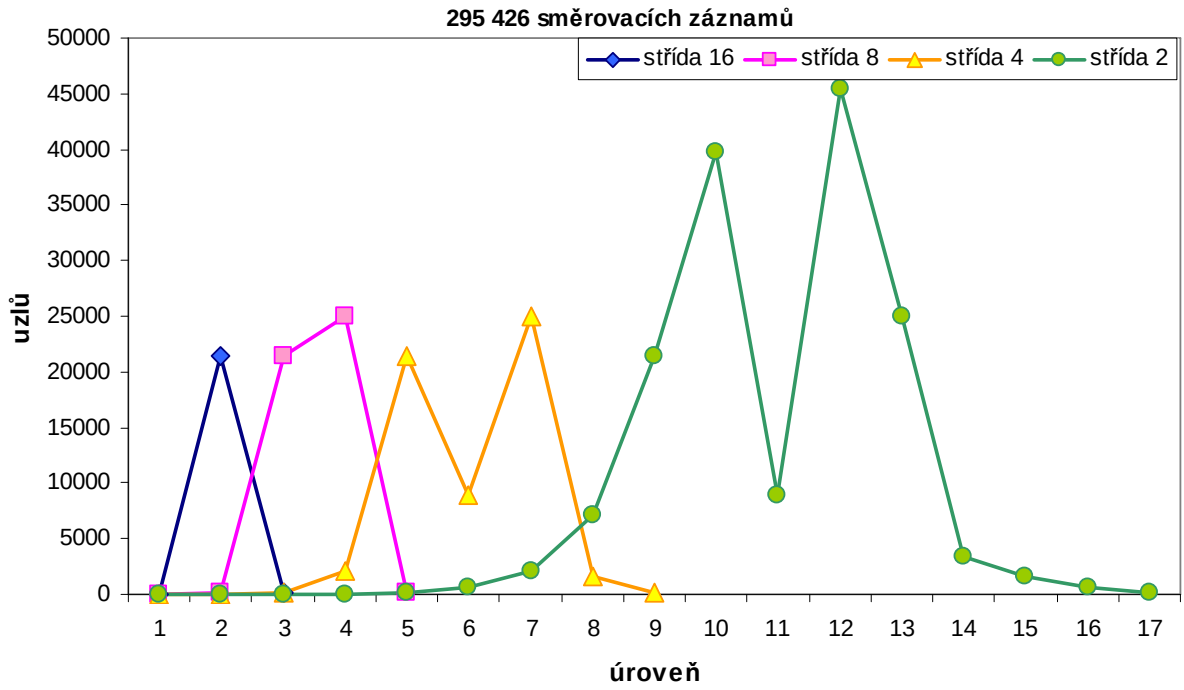


Obrázek 3. 17: Graf alokované paměti pro různě velké ukazatele.

Grafy na obrázcích 3.18 a 3.19 znázorňují počet uzlů v jednotlivých úrovních stromu. Protože není použita optimalizace Koncového uzlu, obsahuje strom v několika málo případech také „uzel navíc“ neboli uzel, ve kterém je využit jediný bit interní bitmapy a záznam je uložen v kořeni. Byly vybrány okrajové testovací sady co se týče počtu záznamů.



3. 18: Počty uzlů v úrovni stromu (nejmenší sada routovacích záznamů).

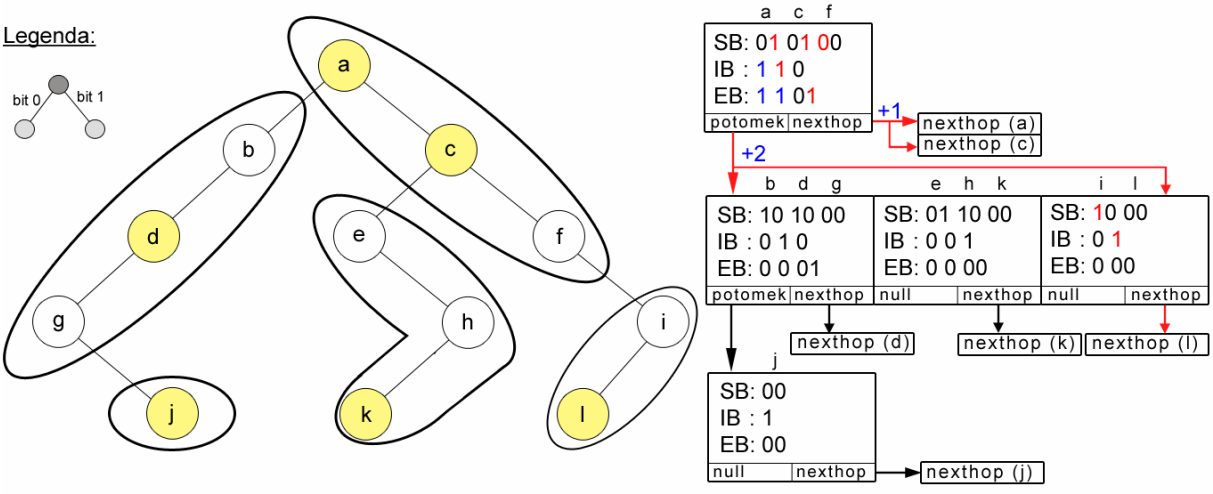


3. 19: Počty uzlů v úrovni stromu (největší sada routovacích záznamů).

3.3 Shape shifting trie

S rozvojem IPv6 přichází nové výzvy k vylepšení efektivnosti algoritmů vyhledávajících nejdelší shodný prefix. Tree Bitmap algoritmus s 8 kB inicializačním pole přímo na čipu (prvních 13 bitů adresy) a střídou velikosti 5 potřebuje v nejhorším případě 4 průchody stromem s IPv4. Ovšem čas roste s délkou adresy lineárně a pro IPv6 už není Tree Bitmap atraktivní. Jednou z cest je implementace Shape Shifting trie algoritmu [2]. Vychází z Trie a z binárních stromů vytváří podstromy různých tvarů. To umožňuje SST algoritmu upravit si strukturu binárního podstromu tak, aby se zredukoval počet uzlů, kterými algoritmus prochází.

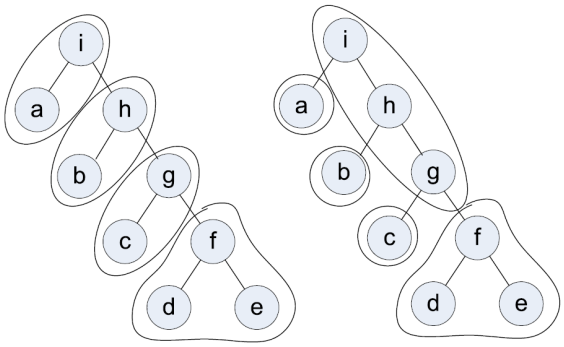
SST je v mnohém podobný Tree Bitmap algoritmu. Kromě interní a externí bitmapy používá i bitmapu tvarů. Uzly i směrovací informace jsou v paměti uloženy za sebou tak, aby se mohly adresovat jediným ukazatelem. SST však nevytváří uzly podle hloubky, ale podle již zmíněných tvarů. Binární strom se rozdělí na tvary obsahující maximálně K uzlů, kde K je parametr tvaru(vytvořeného nového multibitového uzlu z binárního stromu). Na obrázku 3.20 se nachází ukázka takového rozdělní s K = 3. Bitmapa tvarů pro každý uzel má velikost vždy 2K a tvoří se tak, že procházíme stromem zvoleným tvarem a v každém uzlu (na obr. označenými písmeny a-l) zjistíme jestli tvar pokračuje bitem 0 a bitem 1 v tomto pořadí. Pokud tímto bitem pokračuje, zapíšeme do bitmapy tvarů 1, v opačném případě 0. K interní bitmapě přistupujeme jen podle pořadí bitu na vstupu. Externí bitmapa obsahuje bit 1 tam, kde leží potomek uzlu. Můžeme ji procházet pouze s konzultací bitmapy tvarů. Bity značící následníka uzlu jsou zapsaný v pořadí shora dolů a zleva doprava.



Obrázek 3.20: Skladba stromu v algoritmu Shape Shifting Trie s ukázkou průchodu stromem pro K = 3 a vstupní bity „1101“. Barevně označené uzly obsahují směrovací záznamy

Vyhledávání je velmi podobné Tree Bitmap. Vyhledávací proces sestupuje od kořene rekurzivně stromem. V každém kroku používá bity ze vstupní adresy a pokud po cestě narazí na bit 1 v interní bitmapě, uloží si ukazatel do tabulky výsledků. Pokud již strom nepokračuje a nebo jsme na konci, algoritmus teprve tehdy přistoupí do tabulky výsledků (technika líného vyhledávání).

Nabízí se otázka jaké tvary použít, pokud máme více možností. Postupovat můžeme několika způsoby. Na obrázku 3.21 jsou vyobrazeny 2 extrémy. Nalevo se snažíme vytvořit velké uzly a napravo co nejmenší hloubku stromu. Verze s nejmenší hloubkou se používá v praxi, konkrétně v modifikaci SST – Breadth-First Prunning (BFP).



Obrázek 3.21: Maximální velikost uzlu a minimální hloubka stromu v SST algoritmu [2].

Shape Shifting Trie ztrácí svoji výhodu ve stromech (Trie), které obsahují ve svých uzlech více než polovinu možných záznamů. Zde se uplatňuje spíše Tree Bitmap. Kombinací obou algoritmů, kdy se využívají podle potřeby uzly SST nebo Tree Bitmap, využijeme lépe paměť a u adres IPv4 i zvýšíme propustnost. V tabulkách 3.22 – 3.24 jsou výsledky testů se zmíněnými variantami algoritmů. SST nebyl předmětem implementace bakalářské práce a výsledky jsou převzaty z publikace[2] . Testovací sada IPv4 obsahovala řádově 184 000 záznamů z protokolu BGP. Sada pro IPv6 pouze asi 900 prefixů, protože tak velké reálné tabulky neexistovaly.

	výška trie	počet uzlů	propustnost v nejhorším případě	nexthop [Kb]
Trie	32	487696	-	-
Tree Bitmap	6	64245	22,2M pkt/s	845
BFP SST	5	49177	25M pkt/s	648,3
Hybrid SST	4	33094	28,6M pkt/s	436,3

Tabulka 3.22: Výsledky vyhledávání pro IPv4 adresy [2].

	výška trie	počet uzlů	propustnost v nejhorším případě	nexthop [Kb]
Trie	128	5415	-	-
Tree Bitmap	25	1013	7,14M pkt/s	13,4
BFP SST	8	530	18,2M pkt/s	7,0
Hybrid SST	8	491	18,2M pkt/s	6,5

Tabulka 3.23: Výsledky vyhledávání pro IPv6 adresy s maskou sítě až 128 [2].

	výška trie	počet uzlů	propustnost v nejhorším případě	nexthop [Kb]
Trie	64	5015	-	-
Tree Bitmap	12	934	13,3M pkt/s	12,3
BFP SST	5	498	25M pkt/s	6,6
Hybrid SST	5	455	25M pkt/s	6,0

Tabulka 3.24: Výsledky vyhledávání pro IPv6 adresy s maskou sítě až 64. Toto vyhledávání se blíže více realitě [2].

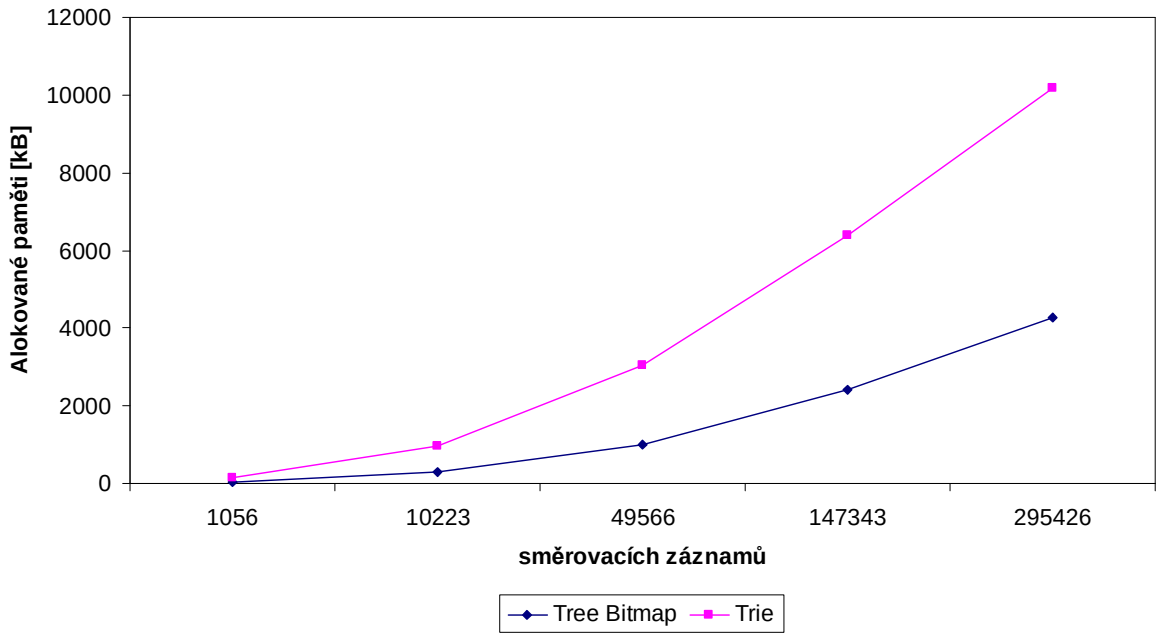
Tento algoritmus může být používán v nejvýkonnějších směrovačích na páteřních sítích a má potenciál pro větší rychlosti linek. Je vyhovující pro rychle rostoucí směrovací tabulky a nejvíce vhodný pro IPv6 adresování. S paměťmi QDR II je tento algoritmus schopný zpracovávat datový tok rychlostí linky (tzv. wire speed) pro IPv4 i IPv6 adresy. Konkrétně se jedná 10 Gbps linky. Teoreticky by jsme se mohli dostat replikováním datových struktur do více GDR pamětí až na 40 Gbps [2].

4 Zhodnocení výsledků

Tato kapitola se zaobírá zhodnocením výsledků, které jsem získal ze softwarové implementace algoritmu Trie a Tree Bitmap. Oficiální převzaté výsledky hardwarové implementace a srovnání jsou uvedeny v kapitole 3.3.

Na obrázku 4.1 je jasně vidět, že algoritmus Trie zaostává za Tree bitmap co se týče alokované paměti více než dvojnásobně. Co se týče počtu kroků vyhledávání, je Tree Bitmap téměř 5x rychlejší. Pro porovnání byla použita střída 4. Jak jsme viděli v grafu 3.14, má nejlepší poměr využití paměti. Ve srovnání se střídou 8 alokuje 4x méně paměti za cenu jediného vyhledávácího kroku navíc (s použitím Inicializačního pole). V práci nejsou záměrně uvedeny velikosti alokované paměti s optimalizací Inicializačního pole, protože rozdíly jsou zanedbatelné – cca 1 kB.

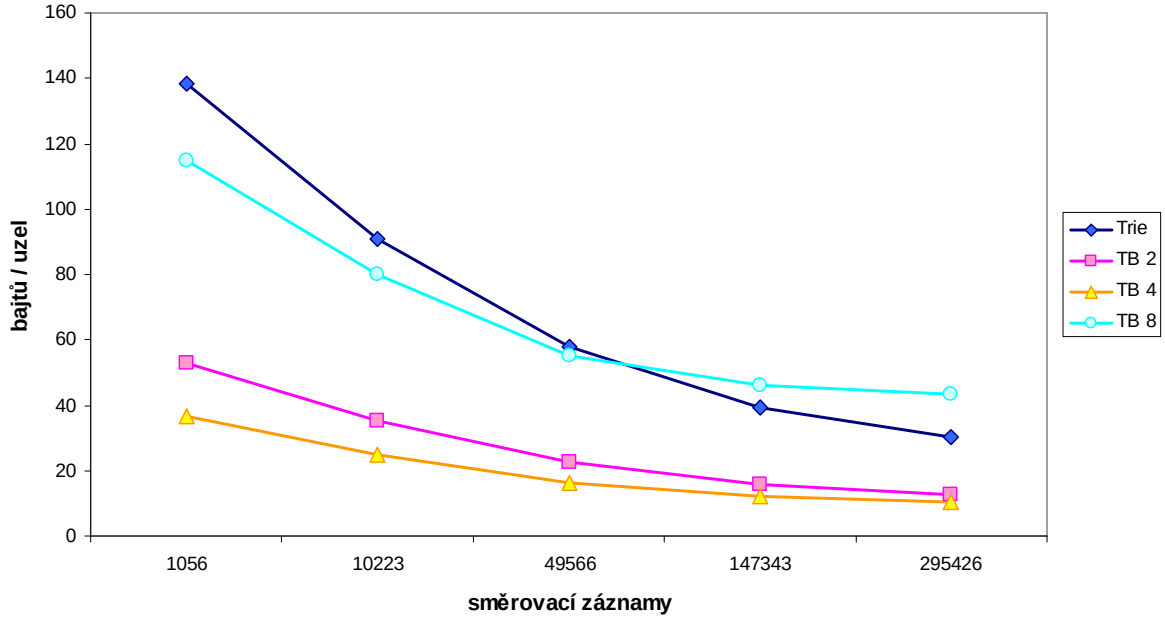
Absolutním vítězem v rychlosti vyhledávání s průměrným počtem kroků 1,99 je Tree Bitmap se střídou 16. Bohužel by u této verze vznikaly velké komplikace při hardwarové implementaci tak složité kombinační logiky. Závažným problémem je i obrovská paměťová náročnost (> 355 MB).



Obrázek 4.1: Graf alokované paměti v závislosti na počtu směrovacích informací. Tree Bitmap používá střídu 4.

Zajímavým ukazatelem efektivity algoritmu je mimo jiné i průměrná velikost paměti na jeden směrovací záznam. Graf 4.2 znázorňuje vývoj alokované paměti pro různé střídy Tree Bitmap ve srovnání s Trie. Zde se objevuje zajímavý stav, kdy střída 8 pro více než cca 70 000 nexthop adres má menší paměťovou efektivitu než základní Trie. V tomto případě při řádově tisíci záznamech vykazuje

Trie o 20% vyšší nároky než Tree Bitmap, zatímco při řádově tři sta tisících záznamech o 30% nižší. Střída 16 nebyla pro svoji náročnost zahrnuta do grafu.



Obrázek 4.2: Graf závislosti velikosti paměti na jeden směrovací záznam v různých testovacích sadách. Graf porovnává Trie a Tree Bitmap se střídou 2(TB 2), 4(TB 4) a 8(TB 8).

5 Závěr

Práce shrnuje paměťové a výpočetní nároky vybraných směrovacích algoritmů. Algoritmy Trie a Tree Bitmap byly odsimulovány se sadami skutečných směrovacích záznamů z protokolu BGP a dosažené výsledky shrnuty.

V práci jsem se věnoval také otázce implementace v hardware. Každý parametr algoritmu má jiné dopady na výkon a některé kombinace nejen, že by nebyly proveditelné po ekonomické stránce, ale přinesly by velké implementační potíže v kombinační logice. Softwarová implementace sice dodržuje principy algoritmů, ale protože je implementována pomocí vyššího programovacího jazyka, trvají jednotlivé operace časově několikanásobně déle. V počtech několika set tisíc směrovacích záznamů probíhá simulace u Tree Bitmap v řádu hodin. Toto ovšem v žádném případě neovlivňuje získané výsledky.

Ačkoliv nebyl Shape Shifting Trie implementován a testován, z dostupných informací je jednoznačně nejefektivnější a s nástupem IPv6 je jeho rozšíření nevyhnutelné.

Řešený problém lze dále rozšiřovat. Na všechny zmíněné algoritmy existuje celá řada rozšíření upravující paměťové a výpočetní nároky. Kromě vylepšování jednotlivých algoritmů můžeme dále zkoumat vlastnosti vyplývající z používání uzlů Tree Bitmap v různých částech Shape Shifting Trie algoritmu.

Literatura

[1] EATHERTON, William. Hardware-based internet protocol prefix lookups. Pages 28-51, Washington University : [s.n.], 1999. 108 s.

[2] SONG, Haoyu, TURNER, Jonathan, LOCKWOOD, John. Shape Shifting Tries for Faster IP Route Lookup. Washington University in St. Luois : [s.n.], c2005. 10 s.

[3] PUŠ, Viktor. Klasifikace paketů s využitím technologie FPGA. Pages 14-15, 31, Brno, 2008. 39 s. FIT VUT. Diplomová práce.

[4] PEARSON, Michael. QDR Consorcium : QDR III preview [online]. c2006 , 6/10/04 [cit. 2009-04-20]. Pdf. Dostupný z WWW: <http://www.qdrconsortium.org/presentation/QDR-III-SRAM.pdf>.

[5] KLAŠKA, Luboš . Svět sítí [online]. 2008 , 25. září 2008 [cit. 2008-12-15]. Dostupný z WWW: <http://www.svetsiti.cz/view.asp?rubrika=Technologie&clanekID=337>.

[6] PUŽMANOVÁ, Rita. Moderní komunikační sítě od A do Z. 2. aktualiz. vyd. Pages 244-258, Brno : Computer Press, a.s., c2006. 430 s. ISBN 80-251-1278-0.

[7] Wikipedia : Směrování [online]. 2008 [cit. 2008-12-10]. CS. Dostupný z WWW: <http://cs.wikipedia.org/wiki/Směrování>.

[8] Wikipedia : IP protokol [online]. 2009 [cit. 2009-04-10]. CS. Dostupný z WWW: <http://cs.wikipedia.org/wiki/Internet_Protocol>.

[9] Potaroo : bgp table [online]. 2009 [cit. 2009-01-30]. EN. Dostupný z WWW: <http://bgp.potaroo.net/as6447/bgptable.txt>.