روشی برای جستجو در جدول مسیریابی IP با استفاده از سخت‌افزار قابل باز پیکربندی

حمید فدیشه‌ای، مسعود صبائی و مرتضی صاحب‌الزمانی
دانشکده مهندسی کامپیوتر و فن‌آوری اطلاعات
دانشگاه صنعتی امیرکبیر (پلی‌تکنیک تهران)
{fadishei, sabaei, szamani}@ce.aut.ac.ir

چکیده

رشد سریع ترافیک در اینترنت و نیز ضرورت استفاده از کاربردهای چند رسانه‌ای جدید، منجر به لزوم استفاده از لینک‌های پر سرعت در حد چندین گیگابیت در ثانیه در مسیریاب‌های اصلی اینترنت شده است. یکی از مهم‌ترین گلوگاه‌ها در مسیریاب‌های IP، جستجوی جدول مسیریابی و یافتن پورت مناسب برای ارسال بسته است. مشکل این کار در این است که این جستجو باید به صورتی باشد که طولانی‌ترین تطبیق انتخاب شود. در سال‌های اخیر روش‌های مختلف نرم‌افزاری و سخت‌افزاری برای این کار ارائه شده است. در این مقاله یک روش جدید برای پیاده سازی جستجو در جدول مسیریابی IP با استفاده از سخت‌افزار قابل باز پیکربندی معرفی می‌شود. نتایج به دست آمده نشان می‌دهد که این روش وقتی به صورت خط لوله‌ای پیاده سازی شود، به بازدهی یک جستجو در هر پالس ساعت در جدول مسیریابی می‌انجامد، ضمن این که به علت عدم وجود دسترسی به حافظه، می‌توان از فرکانس ساعت بالاتری نسبت به سایر روش‌ها استفاده نمود.

کلمات کلیدی

جستجو در جدول مسیریابی IP (IP Address Lookup)
سخت‌افزار قابل بازپیکربندی (Reconfigurable Hardware)

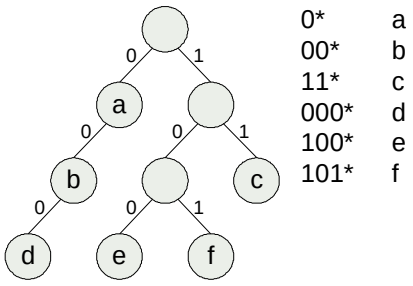
۱. مقدمه

هدف اصلی مسیریاب‌ها ارسال بسته‌ها به سمت مقصد نهایی است. یک مسیریاب برای انجام این کار باید با توجه به گره مقصد هر بسته تصمیم بگیرد که آن را از کدام پورت خروجی به سمت مقصد بعدی ارسال نماید. اطلاعات لازم برای این کار در یک جدول به نام جدول مسیریابی قرار دارد. یک آدرس IP از دو قسمت تشکیل شده است: شناسه‌ی شبکه و شناسه‌ی میزبان. در مسیریابی کلاس‌بندی شده که قبلا مورد استفاده قرار می‌گرفت سه کلاس آدرس A، B و C وجود داشت که در هر

کدام به ترتیب ۷، ۱۴ و ۲۱ بیت از آدرس IP به شناسه‌ی شبکه اختصاص داشت. این روش سبب سادگی مسیریابی می‌شد ولی از طرفی قسمت زیادی از فضای آدرس دهی تلف می‌شد. با ظهور مسیریابی فاقد کلاس‌بندی، بیت‌های مربوط به شناسه‌ی شبکه می‌تواند هر تعداد از کل بیت‌های آدرس را شامل شود. این عدد برای هر مدخل مسیریابی در جدول مسیریابی ذخیره می‌شود. بنابراین در این حالت جدول مسیریابی شامل چندین پیشوند با طول‌های متفاوت است و باید برای مسیریابی هر بسته، از بین تمام پیشوندهایی که با آدرس مقصد بسته مطابقت دارند، طولانی‌ترین آن‌ها انتخاب شود. این به معنای یک جستجوی دو بعدی در جدول مسیریابی است (طول و مقدار) که همین امر، پیاده‌سازی این جستجو را مشکل می‌سازد.

سه معیار مهم برای الگوریتم مورد استفاده در جستجوی جدول مسیریابی IP وجود دارد. اولین معیار، زمان جستجو است که باید در حدی باشد که بتوان کوچک‌ترین بسته‌های IP را با «سرعت سیم» ارسال کرد. برای لینک‌های در حد گیگابیت در ثانیه، باید بتوان چندین میلیون آدرس در هر ثانیه را در جدول جستجو کرد. دومین معیار، زمان به روز کردن جدول مسیریابی است که باید در ۱۰ میلی ثانیه یا کمتر قابل انجام باشد [1]. سومین معیار، مقیاس‌پذیر بودن روش مورد استفاده است. چه از لحاظ این که حجم جداول مسیریابی با گذشت زمان رو به افزایش است و چه از لحاظ حرکت تدریجی به سمت IPv6.

الگوریتم‌های مختلفی برای جستجو در جدول مسیریابی IP ارائه شده است [1][2]. یک ایده‌ی اولیه استفاده از ساختار Trie است. یک Trie یک درخت دودویی است که با شروع از ریشه‌ی آن، در هر سطح، یک بیت از آدرس مقصد بسته بررسی می‌شود و با توجه به مقدار آن بیت، جستجو به یکی از دو فرزند چپ یا راست منتقل می‌شود. این حرکت تا جایی که نتوان ادامه داد انجام می‌شود. اگر در مسیر جستجو یک نود حاوی پیشوند مشاهده کنیم، مقدار آن را به عنوان طولانی‌ترین تطبیق مشاهد شده تا به حال، به خاطر می‌سپاریم. به عنوان مثال، شماره‌ی پورت خروجی متناظر با آن را به عنوان نتیجه‌ی مسیریابی تا به حال، در جایی ذخیره می‌کنیم. شکل ۱ مثالی در این مورد را نشان می‌دهد. در سمت راست شکل پیشوندهای موجود در جدول مسیریابی را مشاهده می‌کنید. فرض کنید بسته‌ای با آدرس مقصد 0011 باید مسیریابی شود (برای سادگی کار، آدرس‌ها را ۴ بیتی در نظر گرفته‌ایم). در ریشه‌ی Trie، پرارزش‌ترین بیت آدرس چک می‌شود که مقدار صفر دارد و ما را به فرزند سمت چپ ریشه هدایت می‌کند. در آن‌جا با نود بیان‌گر پیشوند a مواجه می‌شویم و a را به عنوان طولانی‌ترین تطبیق تا به حال، به خاطر می‌سپاریم. سپس با توجه به بیت بعدی آدرس، باز به فرزند سمت چپ در سطح بعد حرکت می‌کنیم و با توجه به این که دوباره به یک نود بیان‌گر پیشوند رسیده‌ایم، b به عنوان طولانی‌ترین تطبیق تا به حال، جای a را می‌گیرد. بیت بعدی آدرس، یک است و با توجه به این که فرزند سمت راستی برای این نود وجود ندارد، جستجو پایان می‌پذیرد و b به عنوان طولانی‌ترین تطبیق انتخاب می‌شود.



شکل ۱. مثالی از Trie یک بیتی.

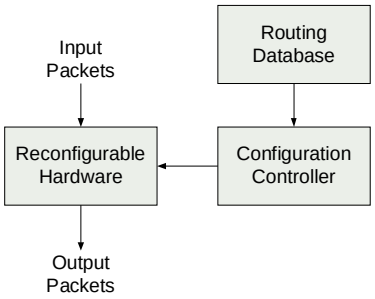
ساختار ابتدایی Trie که به Trie یک بیتی معروف است، غیر بهینه بوده و برای هر جستجو نیاز به تعداد زیادی مراجعه به حافظه (حد اکثر ۳۲ بار برای IPv4) دارد. رویکردهای مختلفی برای بهینه سازی Trie صورت گرفته است. از جمله می‌توان به فشرده‌سازی مسیر با حذف نودهای حاوی یک فرزند و فاقد پیشوند [3] و فشرده سازی در سطوح با چک کردن تعداد

بیشتری بیت در هر سطح [4] اشاره کرد. در [1] می‌توانید خلاصه‌ی مفیدی در مورد روش‌های بهینه سازی Trie بیابید. پیاده‌سازی Trie به هر دو صورت سخت‌افزاری و نرم‌افزاری انجام شده است. البته در هر دو حالت معمولا از حافظه برای ذخیره سازی Trie استفاده شده است. این امر سرعت جستجو را وابسته به پهنای باند حافظه می‌کند. البته سعی شده است تا با بهینه سازی حجم حافظه‌ی مورد استفاده، امکان پیاده سازی آن در حافظه‌های ایستا و همراه تراشه (در روش‌های سخت‌افزاری) و یا در حافظه‌ی نهان کامپیوتر (در روش‌های نرم‌افزاری) وجود داشته باشد. ضمنا سعی شده است تعداد دسترسی به حافظه برای انجام یک عمل جستجو کم شود و این دسترسی‌ها به صورت خط لوله‌ای پیاده‌سازی شود.

روش‌های دیگری غیر از استفاده از Trie هم وجود دارد. از جمله می‌توان به جستجوی دودویی اشاره نمود. البته الگوریتم جستجوی دودویی معمولی نمی‌تواند در این جا به کار رود. در [5] با ایجاد اصلاحاتی در الگوریتم جستجوی دودویی، از آن برای جستجو در جدول مسیریابی IP استفاده شده است که البته این روش هزینه‌ی به روز سازی بالایی دارد.

روش دیگر برای جستجو در جدول مسیریابی IP، استفاده از حافظه‌های قابل آدرس دهی بر اساس محتوا (TCAM) است [6]. در این روش پیشوندها بر اساس طول مرتب شده، در حافظه قرار می‌گیرند. بنابراین آخرین اصابت، طولانی‌ترین تطبیق خواهد بود. مشکل این روش، هزینه‌ی بالای تهیه‌ی TCAM و زمان‌بر بودن به روز سازی جدول مسیریابی است.

استفاده از Trie در صورتی که در حافظه پیاده‌سازی شود غیر بهینه است. اما در یک رویکرد جدید، Trie به صورت یک ماشین حالت متناهی بزرگ در نظر گرفته شده است که نودهای آن حالت‌های ماشین مذکور و یال‌های آن تغییر حالت‌های ماشین هستند. سپس این ماشین حالت متناهی بزرگ به چندین ماشین حالت متناهی کوچکتر شکسته شده است و این ماشین‌های حالت به صورت سخت‌افزاری پیاده‌سازی شده‌اند. در پیاده سازی این ماشین‌های حالت از رجیسترها استفاده شده است و به همین دلیل، با این که پیاده سازی بر اساس Trie یک بیتی بوده است، به نتایج قابل مقایسه با روش‌هایی که از حافظه به همراه فشرده‌سازی‌های پیچیده استفاده می‌کنند منجر شده است [7]. با توجه به این که این ماشین حالت متناهی با تغییر جدول مسیریابی تغییر می‌کند، باید در پیاده‌سازی آن از سخت‌افزار قابل بازپیکربندی استفاده شود. به روز سازی جدول مسیریابی، نیاز به بازپیکربندی سخت‌افزار خواهد داشت که یک کنترل کننده‌ی پیکربندی، این وظیفه را انجام خواهد داد. شکل ۲ نحوه‌ی انجام این کار را نشان می‌دهد.



شکل ۲. ساختار کلی یک جستجوکننده‌ی جدول مسیریابی IP با استفاده از سخت‌افزار قابل بازپیکربندی.

سیستم‌های قابل بازپیکربندی که مزایای انعطاف‌پذیری برنامه‌نویسی نرم‌افزاری را در کنار بازدهی بالای سخت‌افزار، با هم دارند، در سال‌های اخیر در کاربردهای مختلف مورد توجه خاص قرار گرفته‌اند. با تولید FPGA های قابل بازپیکربندی سریع توسط شرکت‌های سازنده، پیاده‌سازی این سیستم‌ها می‌تواند راحت‌تر از گذشته انجام شود.

در این مقاله روش جدیدی برای پیاده‌سازی Trie بر روی سخت‌افزار قابل بازپیکربندی ارائه شده است که در آن هر نود شامل یک بیت رجیستر، مدار منطقی و یال‌ها مسیرهای فیزیکی هستند.

۲. روش پیشنهادی

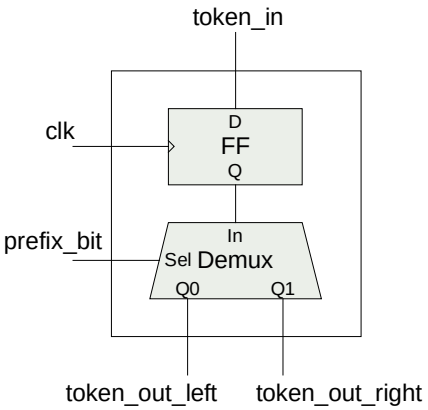
می‌توان Trie را به صورت سخت‌افزاری پیاده‌سازی نمود. به این صورت که در هر نود آن یک بیت رجیستر وجود داشته باشد. برای جستجو در Trie یک توکن (یک بیت یک) وارد رجیستر ریشه‌ی Trie می‌شود. این توکن با هر پالس ساعت، یک سطح در Trie پایین می‌رود. وقتی یک توکن یک سطح در Trie پایین می‌رود، یک توکن دیگر می‌تواند جای آن را بگیرد. به این ترتیب به راحتی می‌توان جستجو را به صورت خط لوله‌ای انجام داد و با پر شدن ارتفاع Trie از توکن، با هر پالس ساعت یک جستجو انجام خواهد شد. برای این که مشخص باشد هر توکن داخل Trie مربوط به مسیریابی کدام آدرس است، یک صف رجیستر در کنار Trie قرار می‌دهیم. با ورود هر توکن به Trie، آدرس مربوط به آن هم وارد صف رجیستر آدرس‌ها می‌شود. در هر پالس ساعت، محتویات صف، همگام با توکن، یک سطح به پایین شیفت پیدا می‌کند. تمام نودهای هر سطح از Trie باید مقدار بیت مناسب از رجیستر آن سطح از صف رجیستر آدرس‌ها را برای انتقال صحیح توکن به سطح بعد مشاهده کنند. وقتی که توکن به نود بیان‌گر پیشوند برسد، باید آن نود را به خاطر بسپاریم. مثلا باید شماره‌ی پورت مربوط به آن پیشوند را در جایی ذخیره کنیم. برای این کار می‌توان یک صف رجیستر دیگر به نام صف رجیستر پورت‌ها در نظر گرفت که شماره‌ی پورت‌های یافته شده تا به حال، در آن با هر پالس ساعت یک سطح به پایین شیفت پیدا می‌کنند. اگر در سطحی از Trie، توکن در نود بیان‌گر پیشوند باشد، به جای شیفت پیدا کردن مقدار رجیستر پورت آن سطح به رجیستر سطح بعد، مقدار پورت موجود در آن نود در در رجیستر سطح بعد نوشته می‌شود. مقدار ورودی به اولین رجیستر این صف می‌تواند پورت پیش‌فرض جدول مسیریابی باشد. البته این پورت می‌تواند در نود ریشه هم مشخص شود. در این صورت اهمیتی ندارد چه مقداری وارد اولین رجیستر صف پورت‌ها می‌شود. با استفاده از این ساختار، وقتی ارتفاع Trie از توکن پر شود، با هر پالس ساعت، یک آدرس از رجیستر آخر صف رجیستر آدرس‌ها خارج می‌شود و همزمان با آن، پورت یافته شده برای آن آدرس از صف رجیستر پورت‌ها خارج می‌شود.

شکل‌های ۳ و ۴ به ترتیب ساختار یک نود عادی و یک نود بیان‌گر پیشوند از Trie را نشان می‌دهند. در هر دو شکل، prefix_bit بیتی از آدرس است که باید در آن سطح از Trie برای هدایت توکن توسط نود، مشاهده شود. این بیت از صف رجیستر آدرس‌ها به نود داده می‌شود. بسته به مقدار این ورودی، در لبه‌ی پالس ساعت، توکن به یکی از دو فرزند چپ یا راست آن نود هدایت می‌شود. یعنی از یکی از خروجی‌های token_out_left یا token_out_right خارج می‌شود. اگر نود مورد نظر یک نود بیانگر پیشوند باشد (شکل ۴)، شماره‌ی پورت مورد نظر هم در نود ذخیره شده است که با ورود توکن به آن نود، مقدار آن بر روی خروجی port_number قرار می‌گیرد و خروجی write_port_number هم فعال می‌شود تا دستور نوشته شدن این شماره‌ی پورت به صف رجیستر پورت‌ها داده شود. خروجی port_number در حالتی که توکن در نود وجود ندارد، مقدار امپدانس بالا دارد تا بتوان این خروجی را در هر سطح از Trie به صورت یک bus پیاده‌سازی کرد.

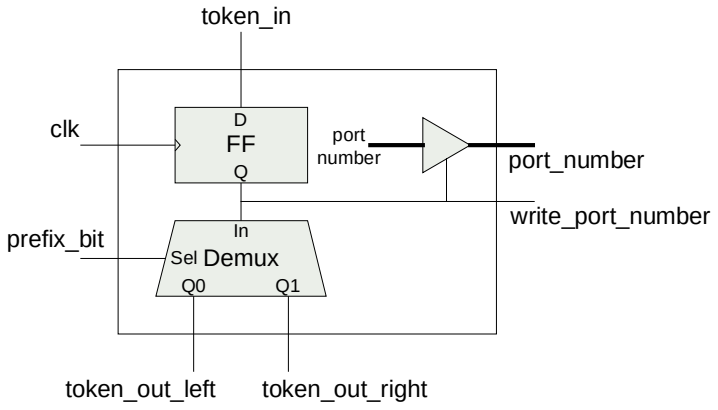
مشاهده‌ی یک مثال در مورد این روش و نحوه‌ی جستجوی خط لوله‌ای در آن می‌تواند به درک مطلب کمک کند. در شکل‌های ۵-الف تا ۵-ح نحوه‌ی عملکرد این روش در جستجو برای ۴ آدرس مختلف مشاهده می‌شود. در این مثال از همان Trie شکل ۱ استفاده شده است. در این شکل‌ها، صف رجیستر آدرس‌ها در سمت چپ Trie و صف رجیستر پورت‌ها در سمت راست آن رسم شده‌اند. همان طور که در شکل ۵-الف مشاهده می‌شود، ۴ آدرس مختلف به ترتیب منتظر ورود به صف رجیستر آدرس‌ها هستند که با هر پالس ساعت یکی وارد صف می‌شود و همزمان با ورود آن، یک بیت یک به عنوان توکن وارد Trie می‌شود و مقدار X هم وارد صف رجیستر پورت‌ها می‌شود. در هر یک از شکل‌ها، فلش‌ها بیان‌گر مسیر حرکت داده‌ها در پالس ساعت بعد است. همان طور که در شکل ۵-پ مشاهده می‌شود، چون در سطح یک از Trie، توکن در نود

بیان‌گر پیشوند a است، به جای شیفت مقدار سطح یک به سطح دو از صف رجیستر پورت‌ها با پالس ساعت بعد، مقدار پورت مشخص شده توسط نود بیان‌گر پیشوند a در رجیستر سطح دو از صف نوشته می‌شود.

همان طور که مشاهده می‌شود، از شکل ۵-ث به بعد که ارتفاع Trie از توکن پر شده است، با هر پالس ساعت نتیجه‌ی یک جستجو مشخص می‌شود.



شکل ۳. ساختار یک نود عادی.



شکل ۴. ساختار یک نود بیان‌گر پیشوند.

۳. پیاده‌سازی جزئی Trie

مسأله‌ای که روش پیشنهادی در اولین نگاه با آن مواجه می‌شود، حجم زیاد منابع سخت‌افزاری مورد نیاز برای پیاده‌سازی آن است. مشاهدات ما بر روی جدول مسیریابی FUNET [8]، نشان داد که ساختار Trie تشکیل شده برای این جدول مسیریابی دارای ۱۲۵۶۶۹ نود است که ۴۱۵۷۸ عدد از این نودها، از نوع نودهای بیان‌گر پیشوند می‌باشند. بزرگ بودن بیش از حد Trie منجر به حجم زیاد منابع سخت‌افزاری مورد نیاز می‌شود. علاوه بر آن، طولانی شدن باس‌ها و مسیرهای بحرانی (مسیرهایی که ارتباط نودهای Trie با صف رجیستر پورت‌ها و صف رجیستر آدرس‌ها را فراهم می‌کنند) پیاده‌سازی این روش را عملا غیر ممکن می‌سازد.

سیستم‌های قابل بازپیکربندی، مشخصه‌ی مهمی دارند که می‌تواند مشکل فوق را حل کند. در یک سیستم قابل بازپیکربندی، می‌توان به جای پیاده‌سازی کل سیستم، در هر لحظه فقط قسمتی از سیستم را که مشغول کار مفید است پیاده‌سازی کرد. با این کار می‌توان در منابع مورد استفاده برای پیاده‌سازی سیستم صرفه‌جویی کرد. مثلا در مورد روش پیشنهادی، در هر لحظه در هر سطح از Trie حداکثر یک توکن وجود دارد. بنابراین پیاده‌سازی کل Trie عملا اتلاف منابع است. می‌توان فقط قسمتی از Trie را که برای حرکت توکن لازم است پیاده‌سازی کرد. ما این کار را «پیاده‌سازی جزئی Trie » نامیده‌ایم. این کار بر خلاف پیاده‌سازی کل Trie، بر روی یک FPGA کوچک هم ممکن است. اما مسأله‌ای که وجود دارد این است که مسیر حرکت توکن برای ما مشخص نیست. ضمنا اگر برای هر عمل جستجو در جدول مسیریابی IP نیاز به یک بار بازپیکربندی سیستم داشته‌باشیم، بازدهی جستجو تا حد غیر قابل قبولی افت خواهد کرد. اما خوشبختانه بسته‌های IP در آدرس مقصد خود دارای خاصیت محلی بودن هستند. یعنی اگر یک جستجو برای یک آدرس مقصد در جدول مسیریابی IP انجام شود، احتمال این که در آینده‌ی نزدیک دوباره همان آدرس مقصد در جدول مسیریابی جستجو شود، زیاد است.

فرض کنید که سخت‌افزار قابل بازپیکربندی ما فقط گنجایش پیاده‌سازی درصدی از کل نودهای Trie را دارد. در شروع کار هیچ نودی بر روی این سخت‌افزار پیاده‌سازی نشده‌است. برای هر عمل جستجو بر روی این سخت‌افزار، در صورت عدم وجود نودهایی که در مسیر توکن وجود دارند، سخت‌افزار بازپیکربندی شده و این نودها در آن ایجاد می‌شوند. در صورتی که فضای خالی برای ایجاد نودهای جدید بر روی سخت‌افزار وجود نداشته‌باشد، قدیمی‌ترین نودها از روی آن حذف می‌شوند تا فضای خالی ایجاد شود. با این تمهید، روش پیشنهادی می‌تواند بر روی FPGA های قابل دسترس فعلی پیاده‌سازی شود.

٤. ارزیابی روش پیشنهادی

همان طور که در قسمت قبل گفته شد، پیاده‌سازی جزئی Trie می‌تواند به عنوان راه حلی برای بزرگ شدن آن در مسیریاب‌های اصلی اینترنت استفاده شود. برای ارزیابی این رویکرد، ساختار پیشنهادی با تعداد نود محدود شده به مقادیر مختلفی پیاده‌سازی شد و هر بار میزان منابع سخت‌افزاری مصرف شده و حداکثر فرکانس ساعت کاری سخت‌افزار و درصد اصابت جستجوها در Trie جزئی اندازه‌گیری شد. این تحلیل با استفاده از جدول مسیریاب FUNET و یک جریان واقعی بسته‌ها بر روی آن انجام شد [8] که نتایج آن در جدول ۱ مشاهده می‌شود. منابع سخت‌افزاری مورد استفاده بر اساس تعداد CLB مورد استفاده از تراشه‌ی FPGA سری Virtex-II از شرکت Xilinx است. سنتز کد به وسیله‌ی نرم‌افزار Leonardo و با بهینه‌سازی تأخیر انجام شده‌است. همان‌طور که مشاهده می‌شود، اگر فقط امکان پیاده‌سازی ۴٪ از نودهای کل Trie مربوط به جدول مسیریابی وجود داشته باشد، در نزدیک به ۹۹٪ موارد جستجو، نیاز به بازپیکربندی Trie وجود نخواهد داشت. این درصد اصابت نسبت به درصد اصابت مربوط به Cache کردن مداخل جدول مسیریابی، بهتر است. علت این امر این است که با آوردن یک مدخل از جدول مسیریابی به داخل Trie جزئی، کلیه‌ی مدخل‌هایی که پیشوندی از آن مدخل بوده‌اند نیز وارد Trie جزئی می‌شوند.

به لحاظ بازدهی جستجو، در روش پیشنهادی یک جستجو در هر پالس ساعت انجام خواهد شد که نسبت به نتیجه‌ی به دست آمده در [7] که به طور متوسط یک جستجو در ۴/۵ الی ۸/۳ پالس ساعت است، بهبود قابل ملاحظه‌ای شاهد هستیم. ضمن این که حداکثر فرکانس ساعت هم از مقدار به دست آمده در [7] که ۱۰۰ مگاهرتز است بیشتر می‌باشد.

البته باید توجه داشت که بازدهی روش پیشنهادی، علاوه بر درصد اصابت Trie جزئی، به سربار بازپیکربندی FPGA مورد استفاده نیز بستگی دارد. جهت کاهش این سربارکه بسته به میزان تغییر سخت‌افزار از چند میکروثانیه تا چند میلی‌ثانیه متغییر است، استفاده از یک الگوریتم افزایشی برای بازپیکربندی جزئی سخت‌افزار با کمترین تغییر ممکن ضروری به نظر می‌رسد. یک خاصیت مفید موجود در FPGA مورد استفاده که در این‌جا خیلی مهم است، این است که امکان بازپیکربندی بخشی از سخت‌افزار در حالی که یک بخش دیگر در حال کار است وجود دارد [9].

جدول ۱. نتایج به‌دست آمده از پیاده‌سازی Trie به صورت جزئی.

درصد اصابت جستجوها در Trie جزئی	حداکثر فرکانس ساعت (مگاهرتز)	تعداد CLB مصرف شده	محدودیت تعداد نود در Trie جزئی
٪۸۳/۷۹	۱۵۱/۵	۸۸۳	۱۰۰۰
٪۹۴/۶۶	۱۳۹/۶	۱۳۱۶	۲۰۰۰
٪۹۸/۹۸	۱۲۰/۲	۲۸۵۷	۵۰۰۰

۵. نتیجه گیری

در این مقاله یک روش برای پیاده سازی Trie بر روی سخت افزار قابل بازپیکربندی ارائه شد که کاربرد آن در جستجوی جدول مسیریابی IP است و با توجه به عملکرد خط لوله‌ای که دارد، می‌تواند در هر پالس ساعت یک جستجو انجام دهد. سپس مشکل بزرگ شدن Trie به علت حجیم بودن جدول مسیریاب‌های اصلی اینترنت در روش پیشنهادی، مطرح شد و پیاده‌سازی جزئی Trie به عنوان راه حل این مشکل ارائه شد.

نویسندگان این مقاله در ادامه‌ی کار، در حال بررسی چگونگی استفاده از ساختارهای بهینه شده‌ی Trie به جای ساختار اولیه‌ی Trie یک بیتی هستند. سپس برای پیاده‌سازی عملی این روش باید یک الگوریتم افزایشی سریع برای سنتز سطح پایین ساختار پیشنهادی بر روی FPGA با قابلیت بازپیکربندی جزئی نوشته شود.

مراجع

[1] M. Ruiz-Sanchez, E. Biersack and W. Dabbous, "Survey and Taxonomy of IP Address Lookup Algorithms", *IEEE Network Mag.*, Mar./Apr. 2001, pp. 8-23.

[2] H. Chao, C. Lam and E. Oki, *"Broadband Packet Switching Technologies"*, John Wiley, 2001, pp. 365-405.

[3] D. Morrison, "PATRICIA- Practical Algorithm to Retrieve Information Coded in Alphanumeric", *Journal of ACM*, Oct. 1968, vol. 15, no. 4, pp. 514-534.

[4] S. Nilsson and G. Karlsson, "IP-Address Lookup Using LC-Tries", *IEEE JSAC*, June 1999, vol. 17, no. 6, pp. 1083-1092.

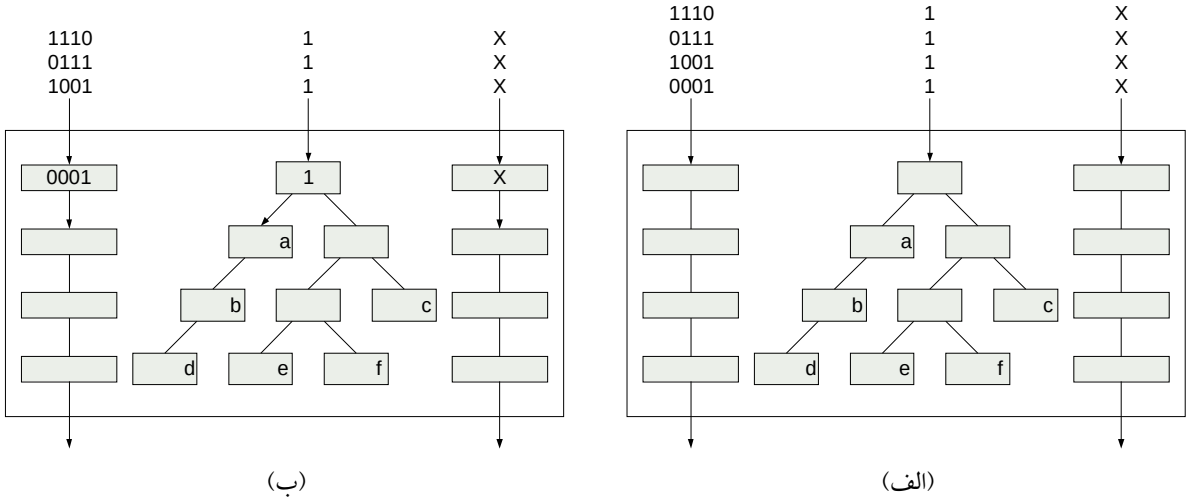
[5] B. Lampson, V. Srinivasan and G. Varghese, "IP Lookups Using Multiway and Multicolumn Search", *IEEE INFOCOM*, Apr. 1998, pp.1248-1256.

[6] A. McAulley, P. Tsuchiya and D. Wilson, "Fast Multi Level Hierarchical Routing Table Using Content -Addressable Memory", *US Patent 05386413*, Jan. 1995.

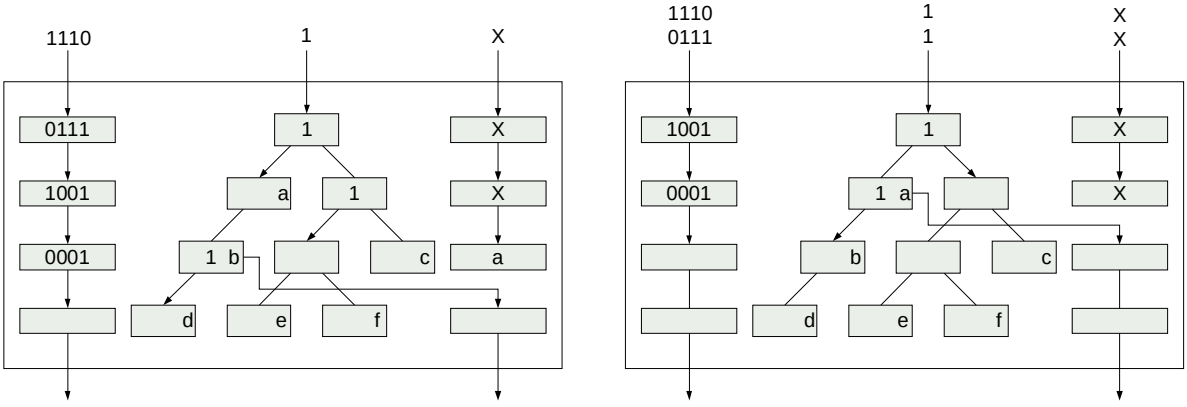
[7] M. Desai, R. Gupta, A. Karandikar, K. Saxena and V. Samant, "Reconfigurable Finite-State Machine Based IP Lookup Engine for High-Speed Router", *IEEE JSAC*, May 2003, vol. 21, no. 4, pp.501-512.

[8] FUNET Router Sample Trace and Routing Table, http://www.nada.kth.se/~snilsson/.

[9] Virtex-II FPGA Family Data Sheet and User's guide, http://www.xilinx.com/.

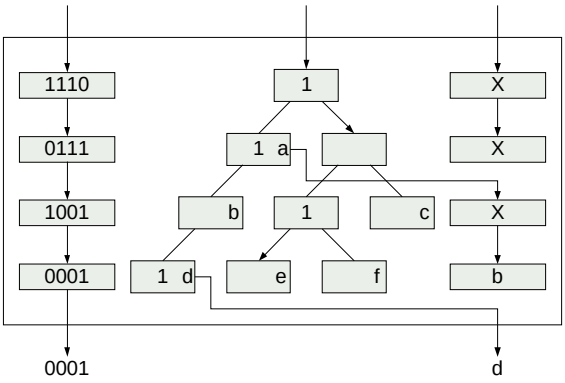


شکل ۵. مثالی در مورد نحوه‌ی عملکرد ساختار پیشنهادی و جستجوی خط لوله‌ای در آن (ادامه دارد).

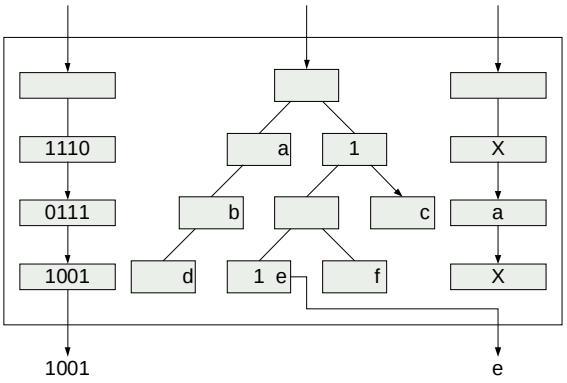


(پ)

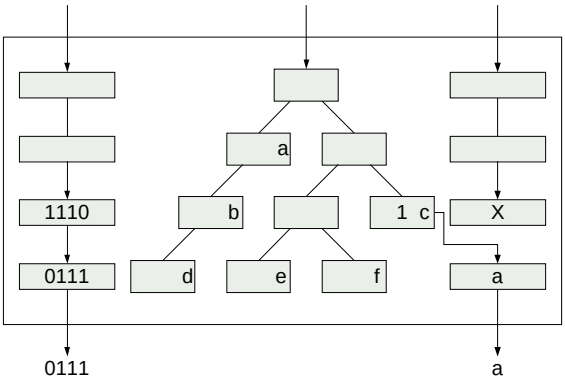
(ت)



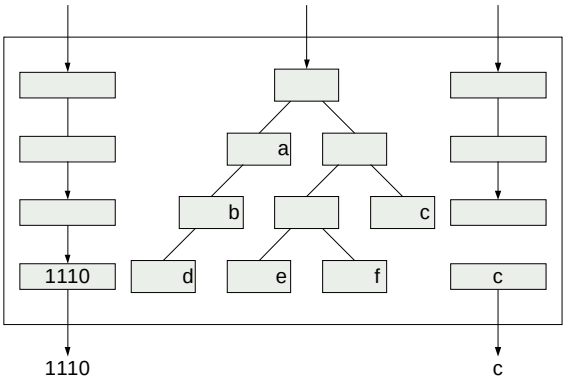
(ث)



(ج)



(چ)



(ح)

شکل ۵. مثالی در مورد نحوه‌ی عملکرد ساختار پیشنهادی و جستجوی خط لوله‌ای در آن (ادامه).