تطبیق پیشوند نام با استفاده از پیش جستجوی فیلتر بلوم برای شبکه محتوی محور

چکیده :

به عنوان یک نمونه‌ی شبکه بندی جدید برای اینترنت آینده، فناوری شبکه‌بندی محتوی محور(CCN) یک زیرساخت ارتباطی محتوی محور برای مقدار روز افزون ترافیک داده، ارائه می کند. برای پیاده‌سازی موفق CCN، لازم است که یک موتور ارسال کارآمد طراحی شود که جست و جوی نام را با سرعت بالایی اجرا کند. این مقاله، استفاده از یک پیشوند نام trie مبتنی بر هش و یک فیلتر بلوم را پیشنهاد می‌کند. در روش پیشنهادی، یک جدول هش خارج چیپ که گره های پیشوند نام trie را ذخیره می‌کند، تنها زمانی دردسترس قرار می‌گیرد که فیلتر بلوم، وجود نود trie را در صف پیشوندها تأیید نماید. در دسترسی به نود، بسته به نتایج فیلتر بلوم، ما دو الگوریتم با راهبرد های مختلف را پیشنهاد می کنیم . اولین الگوریتم به نود trie برای هر نتیجه‌ی مثبت فیلتر بلوم دسترس می یابد، در حالی که الگوریتم دوم، در ابتدا با استفاده از پرس و جو های فیلتر بلوم تلاش می‌کند تا طولانی‌ترین طول تطبیق را تعیین نماید. نودهای trie با استفاده از طولانی‌ترین مسیر ممکن مورد دسترسی قرار می‌گیرند و اگر هیچ تطبیقی وجود نداشته باشد، جستجو از سر گرفته می‌شود. نتایج شبیه‌سازی نشان می‌دهند که روش پیشنهادی می‌تواند خروجی هر نام ورودی با دسترسی تک گره ای به طور متوسط و با دسترسی دو گره ای دربدترین سناریو با استفاده از اندازه منطقی فیلتر بلوم فراهم کند.

لغات کلیدی: شبکه محتوی محور، تطبیق پیشوند نام، فیلتر بلوم، پیشوند نام trie

1. مقدمه

برنامه‌های اینترنتی نوظهور مانند شبکه‌های اجتماعی، به صورت وسیعی فایل‌های تصویری، ویدئویی و صوتی را به اشتراک می‌گذارند. این ترافیک به عنوان محتواهای حجیم و با تقاضای مکرر، به صورت کارآمدی در اینترنت امروزی که زیرساختی مبتنی بر میزبان دارد، انتقال داده نمی‌شود. شبکه‌ی محتوی محور(CCN) یک شبکه‌ی نسل آینده طراحی شده و نوید بخش برای حل این مشکلات در اینترنت امروزی است. شبکه‌ی CCN با نام

شبکه اطلاعات محور(ICN) یا شبکه‌ی داده‌ی نامگذاری شده (NDN) نیز شناخته می‌شود. شبکه‌ی CCN ارتباط داده ها را بر اساس نام های محتوی انجام می دهند، در حالی که اینترنت موجود از ارتباطات میزبان-به-میزبان مبتنی بر آدرس‌های IP استفاده می‌نماید. (Jacobson et al., 2009;Vasilakos et al., Xu et al., 2014; Qiao et al., 2015; Wang et al., 2012; Esteve ;2015;Bari et al., 2012 et al., 2008; Perino and Varvello, 2011). در مقاله‌ی ژاکوبسن و همکاران 2009، ویژگی‌های پایه‌ی CCN پیاده‌سازی شده و تاب آوری و کارآیی معماری CCN با ارتباط مبتنی بر میزبان از نظر انتقال فایل، توزیع محتوا و تماس‌های صوتی مقایسه می شود.

CCN به جای مفهوم میزبان‌های منبع یا مقصد، از مولد و مصرف‌کننده‌ی محتوا استفاده می‌کند. مولدها، محتوا را تولید و مشتریان آن‌ها را دریافت و مصرف می‌کنند. روترها با استفاده از نام‌های محتوا به جای آدرس‌های IP، مسیر یابی را انجام می‌دهند. برخلاف روترهای مرسوم، روترهای CCN قابلیت ذخیره در کش هم دارند، که محتوا را به صورت موقت ذخیره و آن را به کاربران درخواست کننده ارسال می‌کنند. بدین طریق، کاربران CCN می‌توانند به سرعت محتوای درخواستخود را به دست آورده، و لزومی ندارد که همان محتوا به صورت مکرر در طول شبکه انتقال داده شود.

CCN از دو نوع بسته استفاده می‌کند: بسته درخواست و داده. بسته درخواست توسط مصرف کننده، پخش می‌گردد. بسته‌ی داده توسط مولد محتوا تولید شده و به هر نودی که پیام درخواست را شنیده و پیام داده را در اختیار دارد، ارسال می‌شود. یک روتر CCN سه جدول مختلف را تشکیل می‌دهد: جدول محتواها (CS)، جدول تعلیق پیام‌های درخواست(PIT)، و پایه‌ی اطلاعات ارسال (FIB). CS یک کش ذخیره‌کننده‌ی بسته‌های داده است. PIT برای ارسال بسته‌های داده استفاده می‌شود، و FIB برای ارسال بسته‌های درخواستبه کار می‌رود. برای فورواردینگ بسته به صورت پرسرعت، باید الگوریتم‌های جستجوی کارآمدی در اختیار داشت که طولانی‌ترین تطبیق نام را برای هر بسته درخواستورودی اجرا می‌کنند.

trie یک ساختار داده‌ی منظم مبتنی بر درخت است که نام آن از کلمه‌ی retrieval (بازیابی) نشأت گرفته است. عبارت trie را از کلمه‌ی tree هم گرفته‌ایم. در یک ساختار trie، همه‌ی اولاد یک نود، دارای پیشوند

مشترکی مرتبط با آن نود هستند، در حالی که این مسئله همیشه در یک ساختار درختی صحیح نیست. یک پیشوند نام trie (NPT) به عنوان یک trie تعمیم داده شده برای الگوریتم جستجوی نام، پیشنهاد داده شده است (ونگ و همکاران، 2011). NPT یک روش ابتکاری برای جستجوی نام با طولانی‌ترین تطبیق است، اما مشکلی از نظر عملکرد جستجو دارد، چرا که یک جدول FIB ممکن است میلیون‌ها نام محتوا داشته و هر نام محتوا می‌تواند بسیار طولانی باشد.

هدف این مقاله، ارائه‌ی یک روش جدید برای تطبیق طولانی‌ترین نام استفاده شده در جستجوهای FIB در روترهای CCN است. روش پیشنهادی بر اساس پیشوند نام trie می‌باشد. به منظور حل مسئله‌ی عملکرد جستجو در NPT، پیشنهاد کرده‌ایم که یک فیلتر بلوم در درون تراشه ایاضافه شود که قبل از دسترسی به NPT پرس و جو را انجام دهد که این جستار ها در جدول هش برون تراشه ای، ذخیره می‌گردند. چون یک ساختار داده‌ی احتمالاتی و فضا-کارآمد برای تست اینکه آیا یک عنصر، عضوی از یک مجموعه است یا خیر، بکار می‌رود، فیلترهای بلوم عموماً به الگوریتم‌های شبکه اعمال می‌شوند ((Song et al., 2005; Tong et al., and Lim, 2015; Lim et al., 2014a, 2014b 2014;Mun). چون دسترسی به یک حافظه‌ی برون تراشه ای، 10 تا 20 برابر بیشتر از دسترسی به یک حافظه‌ی درون تراشه ایزمان نیاز دارد (پاندا و همکاران، 2000)، با پیش‌جستجوی فیلتر بلوم درون تراشه ایدر روش پیشنهادی ما، جدول هشبرون تراشه ایکه نودهای trie را ذخیره می‌کند، تنها زمانی قابل دسترسی است که احتمال بالایی برای تطبیق ورودی، وجود داشته باشد. نسخه‌ی قبلی و خلاصه‌تر این مقاله در لیم و همکاران، 2015 ارائه شده بود.

عملکرد الگوریتم‌های پیشنهادی از طریق شبیه‌سازی، ارزیابی شده است. چون فرمت نام‌های CCN هنوز تعریف نشده است، نام‌های URL که مشخصات سلسله‌مراتبی مشابهی با نام‌های CCN دارند، برای شبیه‌سازی ما مورد استفاده قرار می‌گیرند (ونگ و همکاران، 2011). حافظه‌ی مورد نیاز برای ایجاد یک فیلتر بلوم و ذخیره‌سازی NPT نیز ارزیابی شده است. با استفاده از ورودی‌هایی که 3 برابر تعداد URLهای ذخیره شده هستند، عملکرد جستجو مورد ارزیابی قرار می‌گیرد.

این مقاله به صورت زیر ساماندهی شده است. بخش 2، کارهای مرتبط از جمله پیشوند نام trie، الگوریتم‌های قبلی جستجوی نام و تئوری فیلتر بلوم را توصیف می‌کند. بخش 3، روش های ایجاد و جستجوی الگوریتم‌های پیشنهادی را معرفی می‌نماید. در بخش 4، نتایج ارزیابی عملکرد نشان داده شده است، و بخش 5 مربوط به نتیجه‌گیری‌های مقاله می‌باشد.

2. پژوهش‌های مرتبط

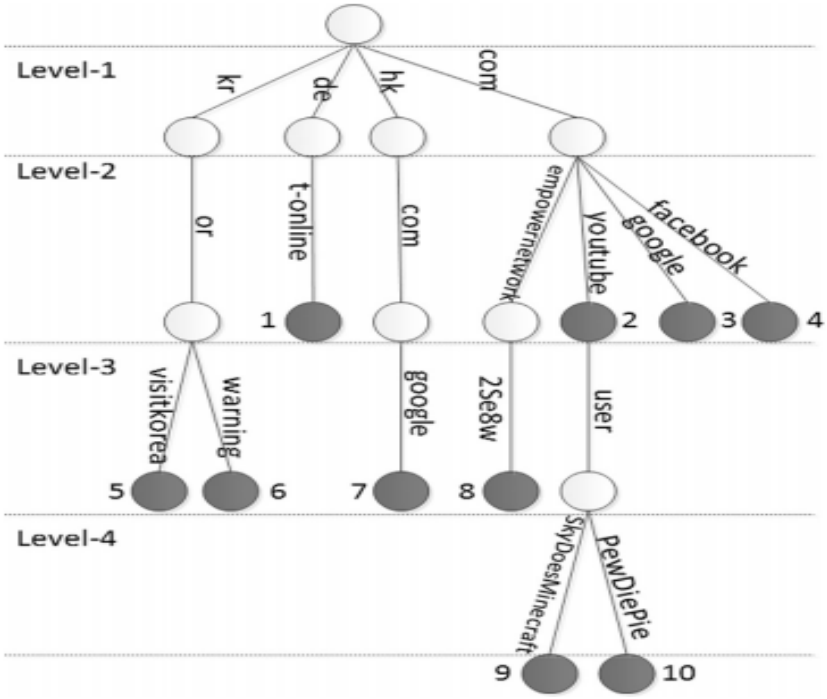
2.1پیشوند نام trie

هر جزء در یک URL از رشته های کاراکتر با طول متغیری تشکیل شده است و با نقطه یا یک اسلش از بقیه‌ی قسمت‌ها جدا می‌شود. برای مثال، URL به آدرس http://www.youtube.com/user/PewDiePie از 4 جزء تشکیل شده است: youtube، com، user، و PewDiePie. در ذخیره‌سازی URL در یک پیشوند نام trie، با منظم کردن معکوس اجزا همراه با نقطه‌ها و جدا کردن هر جزء با استفاده از یک اسلش، نام مثال مورد نظر به com/youtube/user/PewDiePie تبدیل می‌شود.

جدول 1: مثالی از یک FIB در روترهای CCN

Content name	Output face
http://t-online.de	1
http://youtube.com	2
http://google.com	3
http://facebook.com	4
http://visitKorea.or.kr	5
http://warning.or.kr	6
http://google.com.hk	7
http://empowernetwork.com/25e8w	8
http://youtube.com/user/SkydoesMincraft	9
http://youtube.com/user/PewDiePie	10

جدول 1، مثالی از یک FIB را همراه با 10 نام محتوای دلخواه و خروجی‌های متناظر با آن‌ها را نشان می‌دهد. شکل 1، ساختار نام trie (NPT) ایجاد شده برای این مثال را ارائه می‌کند.



شکل 1: پیشوند نام **trie**

هر نود سیاه، یک نام محتوا را نشان داده و یک وجه خروجی را ذخیره می‌کند، که به آن ورودی های مطابق با نام محتوی باید ارسال شوند، در حالی که نودهای سفید در مسیر منتهی به یک نود سیاه از نود ریشه ایجاد می شوند. همانطور که نشان داده شده، NPT اجزای با طول متغیر و نام‌های با طول متغیر را با هم تطبیق می‌دهد. عمق یک NPT توسط نام محتوایی که بیشترین اجزا را دارد، تعیین می‌شود.

مشابه جستجو در یک trie دودویی برای جستجوی آدرس IP، جستجو در یک NPT نیز از نود ریشه شروع می‌شود، و به صورت متوالی اشاره‌گرهای کودک، که با ورودی مفروض تطبیق دارند را دنبال می‌کند. زمانی که هیچ تطبیقی وجود نداشته باشد، روند جستجو کامل شده و بهترین نام منطبق بازگردانده می‌شود که جدیدترین تطبیق نام نیز می‌باشد. برای مثال، فرض کنید که ورودی youtube.com/user/musicbox است. نام ورودی که باید از NPT جست و جو شود، com/youtube/user/musicbox است. چون در نود com/youtube/user، هیچ مرزی برای دنبال کردن وجود ندارد، جستجو تمام شده و وجه خروجی 2 بازگردانده می‌شود، که وجه خروجی com/youtube است. فرض کنید که NPT در یک حافظه‌ی برون تراشه

ای ذخیره شده باشد، تعداد دسترسی‌های نود برون تراشه ای در این مثال برابر با 4 بوده که نود ریشه را شمارش می‌کند: com، com/youtube و com/youtube/user.

جستجوی نام با استفاده از پیشوند نام trie، مشکلی از نظر عملکرد جستجو ایجاد می‌کند. چون جستجو به صورت متوالی و با شروع از نود ریشه اجرا می‌گردد، و چون طول نام می‌تواند خیلی زیاد باشد، عملکرد جستجو در یک NPT برای جستجوی نام سیمی-سرعت کافی نیست. در مقاله‌ی ونگ و همکاران 2013، یک موتور جستجوی نام پرسرعت در پلت‌فرم GPU (واحد پردازشگر گرافیکی) با استفاده از کدینگ مؤلفه‌ی نام، پیاده‌سازی شده است.

2.2 الگوریتم‌های پیشین جستجوی نام

یک الگوریتم جستجوی موازی (ونگ و همکاران، 2011)، یک NPT را به چند ماژول تقسیم کرده و احتمال دستیابی به یک ماژول را محاسبه و جستجوی موازی را برای ماژول‌ها اجرا می‌کند. به هر حال، تخمین احتمال دسترسی برای جریان‌های ورودی تغییر داده شده به صورت دینامیک مشکل بوده و جستجوی موازی، پیچیدگی‌های سخت‌افزاری زیادی ایجاد می‌نماید.

برای بهبود عملکرد جستجوی یک NPT، در مقاله‌ی سو و همکاران (2013) پیشنهاد شده که جستجو یا از سطحی با بیشترین تعداد مؤلفه و یا از سطحی با تعداد تجمعی اجزا برابر با 75٪ تعداد کل، شروع شود. بسته به نتیجه‌ی اولین سطح دسترسی، جستجو می‌تواند کوتاه‌تر و یا طولانی‌تر شود. این الگوریتم مشکلی به نام جستجوی پیشوند دارد یعنی جستجو باید تا زمانی ادامه پیدا کند که یک تطبیق ورودی یافت شده و یا در غیر اینصورت یک نود ریشه مورد بررسی قرار گیرد.

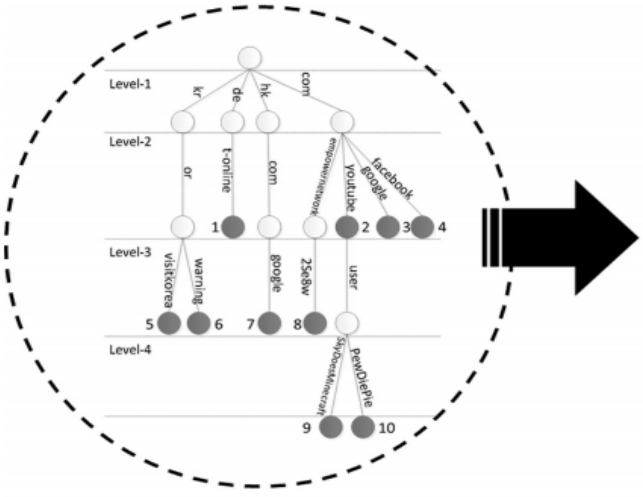
الگوریتم فیلتر نام (ونگ و همکاران 2013) از جستجوی FIB دو مرحله‌ای استفاده می‌کند که در آن هر دو مرحله از فیلترهای بلوم تشکیل شده‌اند. این الگوریتم بخاطر مشکل اشتباه مثبت یک فیلتر بلوم، که نتایج غیردقیق تولید می‌کند، با افزایش اندازه‌های فیلترهای بلوم، نرخ اشتباه مثبت آن‌ها را کمتر از 10^{-8} نگاه می‌دارد.

الگوریتم فیلتر بلوم تطبیقی (خوان و همکاران، 2014)، از یک فیلتر بلوم، یک جدول هش، و یک پیشوند نام trie استفاده می‌کند. در این الگوریتم، هر نام محتوا به بخش‌های پیشوند B و پسوند T تقسیم می‌شود. طول

بخش پیشوند B ثابت و مابقی اجزا در پسوند T قرار می‌گیرند. بخش پیشوند B با استفاده از یک فیلتر بلوم و یک جدول هش، و بخش پسوند T با استفاده از یک NPT پیاده‌سازی می‌گردد. زمانی که تطبیقی در بخش پیشوند B وجود داشته باشد، جستجوی NPT روی بخش پسوند T اجرا می‌شود. عملکرد این الگوریتم به انتخاب طول پیشوند B برای هر مجموعه‌ی مفروض از نام‌های محتوا بستگی دارد.

2.3تئوری فیلتر بلوم

یک فیلتر بلوم (بلوم، 1970) یک بردار بیت است که اطلاعات عضویت مجموعه‌ای از عناصر را نشان می دهد. فیلتر بلومی که مجموعه‌ی $S = \{x_1, x_2, \ldots, x_n\}$ متشکل از n عنصر را نشان می‌دهد، توسط آرایه‌ای از m بیت که به صورت اولیه همگی صفر هستند، نمایش داده می‌شود. فیلتر بلوم دو عملیات را پشتیبانی می‌کند: برنامه نویسی و پرس و جو. در برنامه نویسی، برای یک عنصر $x_i \ (\text{for } i = 1, \ldots, n)$ در مجموعه‌ی S، k اندیس های هشمختلف محاسبه می‌شود که برای j=1,…,k داریم: $0 \le h_j(x_i) < m$. سپس همه‌ی بیت-مکان‌های متناظر با k اندیس های هشدر فیلتر بلوم برابر با 1 تنظیم می‌شود. به عبارت دیگر، هر بیت یک فیلتر بلوم متناظر با یک اندیس های هشاست و به جای ذخیره‌سازی خود داده‌ها در اندیس هش، بیت متناظر با فیلتر بلوم برابر با یک تنظیم شده تا وجود داده را نشان دهد.



Hash table

Trie node	Output face
hkcomgoogle	7
hkcom	-
krorwarning	6
comgoogle	3
comyoutubeuser	-
com	-
det-online	1
hk	-
comyoutube	2
comempowernetwork	-
kror	-
comyoutubeuserSkydoesMincraft	9
comfacebook	4
kr	-
krorvisitKorea	5
comyoutubeuserPewDiePie	10
comempowernetwork25e8w	8
de	-

شکل 2: NPT مبتنی بر هش کردن

مرحله پرس و جو برای تست اینکه آیا ورودی y عضوی از مجموعه است یا خیر، اجرا می‌شود. برای یک ورودی y، k اندیس های هش با استفاده از یک تابع، تولید شده که برای برنامه نویسی فیلتر بکار می‌رود. بیت-مکان‌ها در فیلتر بلوم، که متناظر با شاخص‌های ترکیب هستند، بررسی می‌شوند. اگر هر کدام از مکان‌ها صفر باشد، مسلماً y عضوی از مجموعه‌ی S نیست، و عبارت آن برابر با منفی قرار داده می‌شود. اگر مکان‌های متناظر با شاخص‌های ترکیب برابر با 1 باشند، سپس ورودی ممکن است عضوی از مجموعه بوده و برابر با مثبت قرار داده می‌شود.

به هر حال، ممکن است که این مکان‌ها برابر با عناصر دیگری در مجموعه قرار داده شوند. این نوع از نتایج مثبت، یک مثبت اشتباه می‌باشد. برای یک عنصر تحت بررسی y که در S قرار ندارد، $(y \notin S)$، احتمال مثبت اشتباه بودن f را می‌توان به صورت زیر محاسبه کرد: (لیم و همکاران، a2014):

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

برای نسبت m/n مفروض، می‌دانیم زمانی احتمال مثبت اشتباه مینیمم می‌شود که تعداد توابع ترکیب k از رابطه‌ی زیر پیروی کند (دارماپوریکار و همکاران، 2006):

$$k = \frac{m}{2^{\lceil \log_2 n \rceil}} \ln 2 \quad (2)$$

به صورت کلی، یک فیلتر بلوم می‌تواند نتایج مثبت اشتباه و نه نتایج منفی اشتباه تولید کند. نرخ مثبت اشتباه یک فیلتر بلوم را می‌توان با افزایش اندازه‌ی فیلتر، طوری کنترل کرد که به اندازه‌ی کافی کوچک باشد، اما نمی‌تواند کاملاً آن را حذف نمود.

3. الگوریتم‌های پیشنهادی

3.1 NPT مبتنی بر هش (ترکیب-NPT)

هدف این مقاله، بهبود عملکرد جستجوی پیشوند نام trie (NPT) با استفاده از فیلترهای بلوم است. به عنوان اولین مرحله، باید عملیات ترکیب را در ایجاد NPT اعمال کنیم، و یک الگوریتم مبتنی بر هش NPT در این بخش، ارائه می‌کنیم. شکل 2، جدول هشتشکیل شده برای پیشوند نام trie شکل 1 را نشان می‌دهد. هر نود در پیشوند نام trie، در یک جدول هشذخیره می‌گردد. در ذخیره‌سازی یک نود در سطح iام NPT دقت کنید، از

نود ریشه تا سطح i، i مؤلفه به صورت سلسله‌ای قرار گرفته است، و این رشته به عنوان کلید ترکیب برای بدست آوردن اندیس هش، مورد استفاده قرار می‌گیرد. رشته‌ی سلسله‌ای که طول متغیری دارد بایستی به یک رشته با طول ثابت تبدیل شود چرا که هر ورودی جدول ترکیب، عرض ثابتی دارد. رشته‌ی تبدیل شده با طول ثابت، در ورودی جدول هشو طبق اندیس هش، ذخیره می‌گردد. (ورودی ترکیب نشان داده شده در شکل 2، به جای رشته‌ی تبدیل شده با طول ثابت، برای سادگی دارای یک رشته‌ی متوالی می‌باشد). در ذخیره‌سازی Lامین عنصر، که L طول یک محتوا است، خروجی وجه متناظر با نام محتوا نیز ذخیره می‌شود.

روند ذخیره‌سازی یک نام محتوا در جدول هشبرای الگوریتم NPT مبتنی بر هش، در بخش الگوریتم 1 نشان داده شده است. این روند برای همه‌ی نام‌های محتوای ذخیره شده در یک FIB تکرار می‌شود.

```
Function Store_Name(contentName, outFace)
    (c_1, c_2, ..., c_L) = Decompose(contentName);
    // L is the length of each contentName
    for (i = 1 to L) do
        conStr = Concatenate(c_1, c_2, ..., c_i);
        HT_ind = HashFunc(conStr);
        StoredStr = Convert(conStr);
        if (i == L) then
            StoreHT (HT_ind, StoredStr, outFace);
        else
            StoreHT (HT_ind, StoredStr, NULL);
        end
    end
end
```

الگوریتم 1: ذخیره‌سازی یک نام محتوا در جدول هشدر الگوریتم NPT مبتنی بر هش.

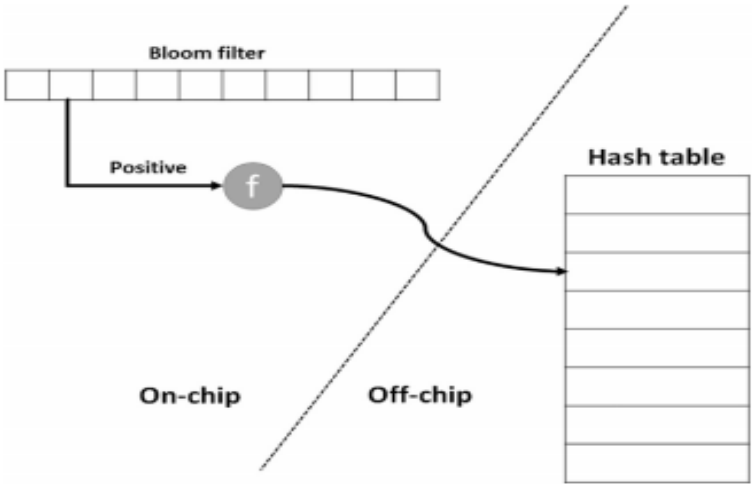
روند جستجوی به کار رفته برای تعیین بهترین نام منطبق (BMN) در الگوریتم NPT در بخش الگوریتم 2، نشان داده شده است. برای نام محتوا در هر بسته‌ی دلخواه، رشته‌ی سلسله‌ی مؤلفه‌های i (برای $1 \le i \le l$، که l طول نام ورودی است) برای بدست آوردن اندیس های هشمورد استفاده قرار می‌گیرد. همچنین رشته‌ی سلسله‌ای برای مقایسه با رشته‌ی ذخیره شده در ورودی ترکیب متناظر با شاخص، به یک رشته‌ی با طول ثابت تبدیل می‌شود. اگر یک رکورد منطبق وجود داشته باشد، روند جستجو با استفاده از رشته‌ی سلسله‌ای از ابتدا تا

مؤلفه‌ی i+1ام ادامه پیدا می‌کند. این روند تا جایی تکرار می‌شود که هیچ رکورد منطبقی وجود نداشته باشد یا جستجو به سطحی برابر با طول نام محتوا برسد. چون وجه خروجی رکورد با طولانی‌ترین تطبیق بایستی بازگردانده شود، وجه خروجی رکورد منطبق فعلی، بایستی در روند جستجو به خاطر سپرده شود. توجه کنید که در یک ساختار trie، اگر هیچ رکورد منطبقی در رشته‌ی جاری مؤلفه‌های متوالی وجود نداشته باشد، هیچ رکورد منطبقی در سطوح طولانی‌تر نیز وجود نخواهد داشت. در نتیجه، جستجو سریعاً تمام می‌شود.

```
Function Search_NPT(inName)
    BMN = default;
    (c1, c2, ..., cI) = Decompose(inName);
    // I is the length of the inName
    for (i = 1 to I) do
        conStr = Concatenate(c1, c2, ..., ci);
        HT_ind = HashFunc(conStr);
        InStr = Convert(conStr);
        match = CompString(StoredStr[HT_ind], InStr);
        if (match != 1) then
            break; // not found
        else // a matching entry found

            if (outFace[HT_ind]! = NULL) then
                BMN = outFace[HT_ind];
            end
        end
    end
    return BMN;
end
```

الگوریتم 2: جستجوی یک نام محتوا در NPT مبتنی بر هش



شکل 3: پیشوند نام trie پیشنهادی همراه با یک BF (NPT-BF)

برای مثال، ورودی مشابهی با جستجوی NPT به صورت com/youtube/user/musicbox در نظر بگیرید. روند جستجو، رکوردهای ترکیب منطبق را با com, comyoutube و comyoutubeuser می‌یابد، اما تطبیقی برای comyoutubeusermusicbox پیدا نمی‌کند. در نتیجه، جستجو پایان یافته و وجه خروجی 2 بازگردانده می‌شود که وجه خروجی comyoutube است. تعداد دسترسی‌های جدول هشدر این مثال، 4 می‌باشد.

این مقاله، برای بهبود عملکرد جستجوی الگوریتم NPT مبتنی بر هش، استفاده از یک فیلتر بلوم را پیشنهاد می‌دهد. ما دو الگوریتم را پیشنهاد کرده‌ایم: NPT-BF و NPT-BF-زنجیره‌ای. هر دو الگوریتم، ساختار مشابهی دارند که شامل یک فیلتر بلوم درون تراشه ایو یک جدول هشبرون تراشه ایهستند. به هر حال، روش‌های جستجوی متفاوتی دارند.

3.2پیشوند نام trie همراه با یک فیلتر بلوم (NPT-BF)

الگوریتم NPT-BF در شکل 3 نشان داده شده است. یک فیلتر بلوم قبل از دسترسی به جدول ترکیب، بررسی می‌شود؛ این فیلتر بلوم دسترسی‌های غیرضروری به جدول هشرا کاهش می‌دهد چرا که در زمان نبود رکورد منطبق، نتایج منفی تولید می‌نماید.

```
Function Search_NPT_BF(inName)
    BMN = default;
    (c1, c2, ..., cI) = Decompose(inName);
    // I is the length of the inName
    for (i = 1 to I) do
        conStr = Concatenate(c1, c2, ..., ci);
        BF_ind = MakeBFInd(conStr);
        if (BFQuery(BF_ind) == positive) then
            HT_ind = HashFunc(conStr);
            InStr = Convert(conStr);
            match = CompString(StoredStr[HT_ind], InStr);
            if (match != 1) then
                break; // false positive
            else
                // a matching entry found
                if (outFace[HT_ind] != NULL ) then
                    BMN = outFace[HT_ind];
                end
            end
        else
            // Search is over when BF negative
            break;
        end
    end
end
```

الگوریتم 3: جستجوی یک نام محتوا در NPT-BF.

روند جستجوی الگوریتم NPT-BF پیشنهادی، در الگوریتم 3 نشان داده شده است. در ابتدا فیلتر بلوم، با استفاده از رشته‌ی متوالی مؤلفه‌های i (برای $1 \le i \le l$، که l طول نام ورودی است) بررسی می‌شود. اگر فیلتر بلوم یک نتیجه‌ی مثبت تولید کند، دسترسی به جدول هشبرقرار می‌شود. اگر یک رکورد منطبق در جدول هشیافت شود، بررسی فیلتر بلوم با استفاده از مؤلفه‌های i+1 ادامه پیدا می‌کند. با این حال، اگر رکورد منطبقی پیدا نشود، که به معنای نتیجه‌ی مثبت اشتباه فیلتر بلوم است، جستجو خاتمه یافته و آخرین وجه خروجی ذخیره شده بازگردانده می‌شود.

برای مثال، همان ورودی youtube.com/user/musicbox را در نظر بگیرید. فیلتر بلوم نتایج مثبتی با com, comyoutube و comyoutubeuser تولید می‌کند، و در نتیجه جدول هشبرای 3 نود، قابل دسترسی خواهد بود. به هر حال، فیلتر بلوم برای comyoutubeusermusicbox، یک نتیجه‌ی منفی تولید می‌کند. تعداد دسترسی‌های جدول هشدر این مثال 3 است که یک واحد کوچکتر از الگورتیم ترکیب-NPT می‌باشد چرا که دسترسی به جدول هشبرای نتیجه‌ی منفی فیلتر بلوم، ممکن نخواهد بود.

به صورت خلاصه، تفاوت بین NPT-BF و NPT-ترکیب این است که جستجو را می‌توان سریعاً و در زمانی که فیلتر بلوم، نتیجه‌ی منفی در NPT-BF تولید می‌کند، به پایان رساند، بدون اینکه به جدول هش دسترسی ایجاد شود. در نتیجه، NPT-BF می‌تواند عملکرد جستجوی NPT-ترکیب را بهبود بخشد که به وسیله‌ی تعداد دسترسی‌های به جدول هشبرون تراشه ایاندازه‌گیری می‌شود.

3.3پیشوند نام trie همراه با اتصال زنجیره‌ای فیلتر بلوم (NPT-BF-زنجیره‌ای)

الگوریتم NPT-BF-زنجیره‌ای از این نکته استفاده می‌کند که احتمال تولید نتیجه‌ی مثبت اشتباه در یک فیلتر بلوم را می‌توان با افزایش اندازه‌ی فیلتر، تا حد زیادی پایین نگاه داشت. در این الگوریتم، اگر فیلتر بلوم یک نتیجه‌ی مثبت تولید کند، فیلتر به بررسی خود تا سطح بعدی و بدون دسترسی به جدول ترکیب، ادامه می‌دهد. اگر فیلتر بلوم یک نتیجه‌ی منفی تولید نماید، روند جستجو به یک سطح عقب‌تر، یعنی آخرین سطح مثبت، بازمی‌گردد. دسترسی به جدول هشبرون تراشه ایدر این سطح امکان‌پذیر است. اگر همراه با یک وجه خروجی، رکورد منطبقی وجود داشته باشد، جستجو کامل شده و وجه خروجی بازگردانده می‌شود. اگر رکورد منطبقی

وجود نداشت، ممکن است یکی از حالات زیر اتفاق افتاده باشد: نتیجه‌ی مثبت فیلتر بلوم اشتباه باشد، یا نتیجه مثبت بوده اما نود یک نود داخلی بدون وجه خروجی است. در این موارد، بایستی ردیابی مجدد صورت گیرد تا جایی که یک رکورد منطبق همراه با یک وجه خروجی در جدول هشیافت شود. معماری کلی NPT-BF-زنجیره‌ای در شکل 4 نشان داده شده است.

```
Function Search_NPT_BF_Chaining(inName)
    BMN = default;
    (c_1, c_2, ..., c_I) = Decompose(inName);
    // I is the length of the inName
    for (i = 1 to I) do
        conStr = Concatenate(c_1, c_2, ..., c_i);
        BF_ind = MakeBFInd(conStr);
        if (BFQuery(BF_ind)== negative) then
            break;
        end
    end
    if (i < I) then
        i = i - 1;
    end
    while (i > 0) do
        conStr = Concatenate(c_1, c_2, ..., c_i);
        HT_ind = HashFunc(conStr);
        InStr = Convert(conStr);
        match = CompString(StoredStr[HT_ind], InStr);
        if ( (match == 1) && (outFace[HT_ind]! = NULL) ) then
            BMN = outFace[HT_ind];
            break;
        else
            i = i - 1; // back-tracking
        end
    end
end
```

الگوریتم 4: جستجوی در NPT-BF-زنجیره‌ای

روند جستجوی NPT-BF-زنجیره‌ای در بخش الگوریتم 4 به صورت خلاصه آورده شده است. در حالی که روند ایجاد NPT-BF-زنجیره‌ای مشابه روند NPT-BF است، اما روند جستجو متفاوت می‌باشد، چرا که بررسی‌های فیلتر بلوم به صورت متوالی تا جایی انجام می‌شود که نتیجه‌ای منفی اتفاق بیافتد. روند جستجوی الگوریتم NPT-BF-زنجیره‌ای دو مرحله دارد. مرحله‌ی اول شامل بررسی فیلتر بلوم به صورت متوالی برای نتایج مثبت

است. مرحله‌ی اول تا جایی ادامه پیدا می‌کند که نتیجه‌ی منفی فیلتر بلوم اتفاق بیفتد یا همه‌ی اجزای نام ورودی استفاده شود.

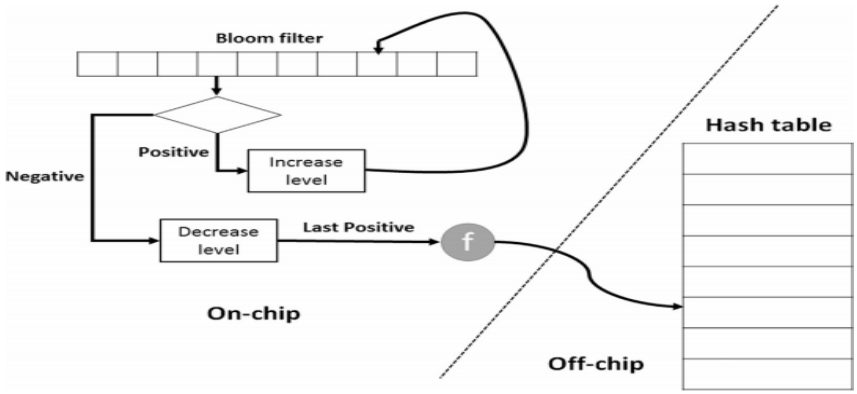
دسترسی به جدول هشبرون تراشه ای، در مرحله‌ی دوم اتفاق می‌افتد. نتیجه‌ی منفی یک فیلتر بلوم در مرحله‌ی i به این معنی است که هیچ رشته‌ای در جدول هشبا رشته‌ی متوالی مؤلفه‌های i، منطبق نیست. چون یک نتیجه‌ی مثبت در سطح i-1 وجود دارد، دسترسی به جدول هشبرای رشته‌ی سلسله‌ای i-1 مؤلفه برقرار می‌شود. (زمانی که اولین مرحله بخاطر استفاده شدن تمام مؤلفه‌های ورودی تمام می‌شود، دسترسی به جدول هشبایستی از سطح i اتفاق بیفتد).

در زمان دسترسی به جدول هشمی‌توان سه مورد را برشمرد: نود منطبق با یک وجه خروجی، هیچ نود منطبقی وجود نداشته و یا یک نود داخلی بدون وجه خروجی وجود داشته باشد. اگر یک رکورد منطبق همراه با وجه خروجی در جدول هشبه دست آید، روند جستجو کامل شده و وجه خروجی رکورد متناظر بازگردانده می‌شود. در غیر اینصورت، اگر هیچ نودی وجود نداشته باشد، نتیجه‌ی مثبت فیلتر بلوم یک نتیجه‌ی اشتباه بوده که موجب ردیابی مجدد می‌گردد. در غیر اینصورت، اگر یک نود داخلی بدون وجه خروجی حاصل شود، سپس نتیجه‌ی مثبت فیلتر بلوم درست بوده اما ردیابی مجدد بایستی اجرا شود. این مسئله با پیش‌محاسبه‌ی وجه خروجی بهترین نام منطبق در هر نود داخلی، رفع می‌گردد. بهترین نام منطبق، همان نام در والد مستقیم هر نود داخلی است. در بخش ارزیابی عملکرد، نتایج شبیه‌سازی را برای هر دو مورد نشان خواهیم داد: با و بدون پیش‌محاسبه.

برای مثال، همان ورودی youtube.com/user/musicbox را در نظر بگیرید. فیلتر بلوم، همراه با com, comyoutube، و comyoutubeuser نتایج مثبتی تولید می‌کند، اما نتیجه برای comyoutubeusermusicbox منفی است. زمانی که فیلتر بلوم، نتیجه‌ی منفی تولید می‌نماید، دسترسی به جدول هشبرای آخرین نتیجه‌ی مثبت که comyoutubeuser است، ممکن خواهد بود. چون این مورد، یک نود خالی بدون وجه خروجی است، ردیابی مجدد اتفاق می‌افتد. دسترسی به رکورد ترکیبی comyoutube انجام شده و با برگرداندن وجه خروجی 2، جستجو تمام می‌شود. تعداد دسترسی به جدول هشدر این مثال 2

است. اگر خروجی وجه comyoutube در نود comyoutubeuser از قبل محاسبه شود، تعداد دسترسی به جدول هشرا می‌توان برابر با 1 نمود.

به صورت خلاصه، الگوریتم NPT-BF-زنجیره‌ای دسترسی به جدول هشرا تا زمانی به تعویق می‌اندازد که فیلتر بلوم، یک نتیجه‌ی منفی تولید کند، در حالی که الگوریتم NPT-BF برای هر نتیجه‌ی مثبت فیلتر، دسترسی به جدول هشبرون تراشه ایرا ممکن می‌سازد. به عبارت دیگر، الگوریتم NPT-BF-زنجیره‌ای در ابتدا، طولانی‌ترین تطبیق ممکن در یک رشته‌ی ورودی را با استفاده از بررسی‌های فیلتر بلوم تعیین می‌کند. اگر نرخ مثبت اشتباه از طریق تعیین اندازه‌ی مناسب فیلتر بلوم، پایین نگاه داشته شود، می‌توان عملکرد جستجوی سریعی را به دست آورد. این مسئله از طریق شبیه‌سازی نشان داده شده است که الگوریتم NPT-BF-زنجیره‌ای، با استفاده از میانگین دسترسی به جدول هشبرون تراشه ایبرابر با 1، جستجوی نام کارآمدی را ارائه می‌دهد.



شکل 4: پیشوند نام trie پیشنهادی همراه با یک BF زنجیره‌ای (NPT0BF-زنجیره‌ای)

جدول 2: تعداد نام‌های محتوا در هر سطح

Level	10 k	50 k	100 k	300 k
1	0	0	0	0
2	501	3943	9557	55,473
3	1429	8325	18,471	65,379
4	2059	11,284	23,579	62,189
5	2392	11,860	22,562	54,417
6	1693	7274	13,501	32,642
7	975	3964	6962	17,080
8	533	1868	3000	7151
9	216	783	1267	2970
10	100	338	554	1351
Longer	102	361	545	1337
Total	10,000	50,000	100,000	300,000

جدول 3: ساختارهای داده‌ی ورودی ترکیبی

Field	Length (bits)
Stored string	64
Output face	8

Stored string: رشته‌ی ذخیره شده، output face: وجه خروجی

1. ارزیابی عملکرد

ارزیابی عملکرد با استفاده از نام‌های URL انجام شده چرا که ساختار سلسله‌مراتبی مشابهی با نام‌های محتوا دارند. در بین 1 میلیون نام ارائه شده در الکسا (alexa)، چهار مجموعه با اندازه‌های 10000، 50000، 100000 و 300000 برای شبیه‌سازی ما استخراج شده است. برای این مجموعه‌ها، الگوریتم‌های NPT، NPT-ترکیبی، NPT-BF و NPT-BF-زنجیره‌ای با استفاده از زبان C++ تشکیل شده‌اند.

شکل 2، تعداد نام‌های محتوا را طبق تعداد مؤلفه‌ها در هر نام نشان می‌دهد. تعداد مؤلفه‌ها در هر نام، سطحی که نام محتوا در یک پیشوند نام trie قرار دارد را تعیین می‌کند. هر مجموعه با استفاده از تعداد اجزای آن، مانند 10k، 50k، 100k و 300k نامگذاری شده است.

جدول 3، ساختار داده‌ی جدول هشبرون تراشه ایکه نودهای پیشوند نام trie را ذخیره می‌کند، نشان می‌دهد. فرض می‌کنیم که یک تابع ترکیب کامل، در ذخیره‌سازی و دسترسی به نودهای trie در جدول هشداده شده است. هر رشته‌ی ذخیره شده در یک نود NPT با استفاده از تابع ترکیب، به یک رشته‌ی 64 بیتی تبدیل شده و در نتیجه هر رکورد ترکیب، همانطور که در جدول 3 نشان داده شده، عرضی 9 بایتی دارد.

جدول 4، مشخصات پیشوند نام trie ایجاد شده برای هر مجموعه را نشان می‌دهد. تعداد ورودی‌ها در جدول ترکیب، N' ثابت و برابر با $2^{\lceil \log_2 N \rceil}$ است، که N تعداد نودها در یک پیشوند نام trie می‌باشد. با استفاده از ساختار داده‌ی نشان داده شده در جدول 3، حافظه‌ی برون تراشه ایمورد نیاز تخمین زده شده است. حافظه‌ی مورد نیاز با استفاده از تعداد ورودی‌های N' ضربدر عرض ورودی که برابر با 9 بایت است، محاسبه می‌گردد. حافظه‌ی مورد نیاز نشان داده شده در جدول 4 که برحسب کیلوبایت است بر 1024 تقسیم می‌شود. توجه کنید

که حافظه‌ی مورد نیاز برای جداول ترکیب برون تراشه ایبرای هر سه الگوریتم یکسان است، چرا که ساختارهای trie مشابهی را تشکیل می‌دهند. همچنین به یاد داشته باشید، حتی اگر بهترین تابع ترکیب برای ذخیره‌سازی هر نود NPT در جدول هشفرض شود، اندازه‌ی جدول هشدر یک حافظه‌ی درون تراشه ای، برای مجموعه‌های نام بزرگ در فناوری امروزی مناسب نیست.

جدول 4: مشخصات trie.

Characteristic	10 k	50 k	100 k	300 k
No. of names	10,000	50,000	100,000	300,000
Trie depth	21	21	30	32
No. of nodes (N)	39,388	178,030	336,314	794,866
No. of entries (N')	2^{16}	2^{18}	2^{19}	2^{20}
Hash table size (KB)	576	2304	4608	9216

جدول 5: عملکرد جستجو: 10k

Metric	NPT	Hashing-NPT	NPT-BF			NPT-BF-Chaining (w/o pre-comp)			NPT-BF-Chaining (w pre-comp)		
BF size	-	-	$4N'$	$8N'$	$16N'$	$4N'$	$8N'$	$16N'$	$4N'$	$8N'$	$16N'$
BF mem (KB)	-	-	32	64	128	32	64	128	32	64	128
No. of false pos	-	-	308	19	0	318	19	0	318	19	0
Wst false pos	-	-	-	-	-	2	1	0	2	1	0
Avg node acc	4.84	4.05	3.85	3.84	3.84	1.68	1.67	1.67	0.98	0.97	0.97

جدول 6: عملکرد جستجو: 50k

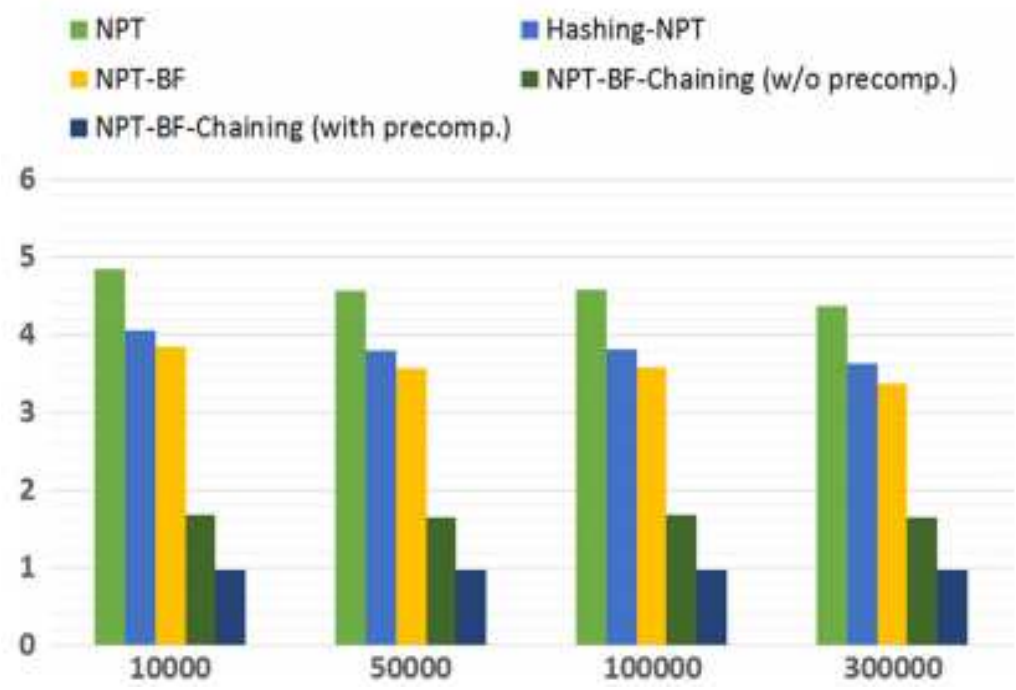
Metric	NPT	Hashing-NPT	NPT-BF			NPT-BF-Chaining (w/o pre-comp)			NPT-BF-Chaining (w pre-comp)		
BF size	-	-	$4N'$	$8N'$	$16N'$	$4N'$	$8N'$	$16N'$	$4N'$	$8N'$	$16N'$
BF mem (KB)	-	-	128	256	512	128	256	512	128	256	512
No. of false pos	-	-	2242	153	4	2406	153	4	2406	153	4
Wst false pos	-	-	-	-	-	3	1	1	3	1	1
Avg node acc	4.56	3.80	3.58	3.56	3.56	1.67	1.65	1.65	0.99	0.97	0.97

جدول 7: عملکرد جستجو: 100k

Metric	NPT	Hashing-NPT	NPT-BF			NPT-BF-Chaining (w/o pre-comp)			NPT-BF-Chaining w pre-comp)		
BF size	-	-	$4N'$	$8N'$	$16N'$	$4N'$	$8N'$	$16N'$	$4N'$	$8N'$	$16N'$
BF mem (KB)	-	-	256	512	1024	256	512	1024	256	512	1024
No. of false pos	-	-	3667	213	1	3873	213	1	3873	213	1
Wst false pos	-	-	-	-	-	3	1	1	3	1	1
Avg node acc	4.58	3.80	3.59	3.58	3.58	1.69	1.68	1.67	0.99	0.97	0.97

جدول 8: عملکرد جستجو: 300k.

Metric	NPT	Hash-NPT	NPT-BF			NPT-BF-Chaining (w/o pre-comp)			NPT-BF-Chaining (w pre-comp)		
BF size	-	-	$4N'$	$8N'$	$16N'$	$4N'$	$8N'$	$16N'$	$4N'$	$8N'$	$16N'$
BF mem (KB)	-	-	512	1024	2048	512	1024	2048	512	1024	2048
No. of false pos	-	-	18,392	1590	33	19,832	1599	33	19,832	1599	33
Wst false pos	-	-	-	-	-	4	2	1	4	1	1
Avg node acc	4.37	3.62	3.39	3.37	3.37	1.67	1.65	1.65	1.00	0.97	0.97



شکل 5: تعداد میانگین دسترسی‌های نود.

تعداد ورودی‌ها برای ارزیابی عملکرد جستجو در هر الگوریتم، 3 برابر تعداد عناصر در هر مجموعه می‌باشد. یک سوم مسیر ورودی شامل نام‌های محتوا می‌شود که برای ایجاد نام محتوای trie استفاده می‌شود دو سوم باقیمانده، برای نام‌های محتوای استفاده نشده جهت تشکیل پیشوند نام trie می‌باشد.

برای تعداد نودهای برابر با N در یک پیشوند نام trie که بایستی در فیلتر بلوم برنامه نویسی شود، اندازه‌ی اصلی فیلتر را می‌توان برابر با $N' = 2^{\lceil \log_2 N \rceil}$ قرار داد. فرض کنید m اندازه‌ی فیلتر بلوم باشد که مضربی از اندازه‌ی اصلی است. تعداد شاخص‌های ترکیب $k = (m/N') \ln 2$ خواهد بود (سانگ و همکاران، 2005). در این مقاله، شبیه‌سازی‌ها را برای اندازه‌های مختلف فیلترهای بلوم انجام می‌دهیم که عبارتند از $m = 4N', 8N'$ و $16N'$، و ارزیابی عملکرد جستجو از نظر اندازه‌ی فیلتر بلوم انجام می‌گیرد.

یک مولد چک افزونگی سیکلی (CRC)، اسکرمبلر بیت بسیار مناسبی است که عموماً در تولید شاخص‌های ترکیب به کار می‌رود. یک مولد CRC 64 بیت برای بدست آوردن شاخص‌های ترکیب در فیلتر بلوم استفاده شده است. رجیسترها در مولد CRC به صورت اولیه برابر با صفر تنظیم می‌شوند. هر بیت ورودی در مولد CRC شیفت داده می‌شود و بعد از اینکه همه‌ی بیت‌های ورودی در آن شیفت داده شدند، مقدار رجیستر به عنوان یک کد CRC بازگردانده می‌شود. چند اندیس های هشبا اندازه‌های متغیر را می‌توان به سادگی و از طریق ترکیب بیت‌ها در یک کد CRC، از آن استخراج نمود. (الاگو پریا و لیم، 2010).

جدول 5-8 نتایج شبیه‌سازی را نشان می‌دهد. همانطور که در بخش قبلی بیان شد، حتی اگر نتیجه‌ی مثبت فیلتر بلوم صحیح هم باشد، در الگوریتم NPT-BF-زنجیره‌ای، یک ردیابی مجدد را می‌توان انجام داد. به همین دلیل، برای الگوریتم NPT-BF-زنجیره‌ای شبیه‌سازی را برای دو حالت اجرا می‌کنیم: بدون پیش محاسبه و با پیش محاسبه‌ی وجه خروجی بهترین تطبیق نام. در الگوریتم NPT-BF-زنجیره‌ای بدون پیش‌محاسبه، زمانی که نود داخلی در نظر گرفته شود، تعداد دسترسی‌های نود افزایش یافته تا جایی که یک نود با نام محتوای مورد نظر بدست آید. از طرف دیگر، اگر نود داخلی در نظر گرفته شود، در الگوریتم NPT-BF-زنجیره‌ای با پیش‌محاسبه، نام محتوای والد مستقیم، در نودهای داخلی ذخیره می‌شود و در نتیجه تعداد دسترسی‌های به نود

افزایش نمی‌یابد. در نتیجه، تفاوت در تعداد دسترسی‌های به نود بین الگوریتم‌های بدون پیش‌محاسبه و با پیش‌محاسبه، تعداد دسترسی‌های به نود داخلی را نشان می‌دهد.

جدول 5، عملکرد جستجو را برای مجموعه‌ی 10k نشان می‌دهد. الگوریتم‌های NPT و NPT ترکیبی، وجه خروجی را به ترتیب و به صورت میانگین، 4.84 و 4.05 نشان می‌دهند. چون این الگوریتم‌ها براساس ساختار trie هستند، اگر نودی در یک سطح وجود نداشته باشد، روند جستجو سریعاً خاتمه می‌یابد. برخی از ورودی‌ها در سطح اول، نام منطبقی ندارند و در نتیجه تعداد دسترسی‌های نود برای این ورودی‌ها، برای هر دو الگوریتم 1 است. الگوریتم NPT-ترکیب به صورت میانگین، عملکرد کمی بهتری نسبت به الگوریتم NPT از خود نشان می‌دهد، چرا که در زمان استفاده از همه‌ی نام‌های ورودی در روند جستجو، NPT به یک دسترسی بیشتر از NPT-ترکیبی نیاز دارد. برای مثال، ورودی youtube.com/user را فرض کنید. تعداد دسترسی‌های نود برای NPT برابر با 4 است، یعنی نود ریشه، com، com/youtube، و com/youtube/user، در حالی که برای الگوریتم NPT-ترکیبی این عدد برابر با 3 می‌باشد و عبارتست از com، comyoutube، و comyoutubeuser.

برای الگوریتم‌های همراه با فیلتر بلوم، حافظه‌ی مورد نیاز فیلتر نسبت به اندازه‌ی اصلی 'N محاسبه می‌گردد. comyoutubeuser بدترین حالت برای تعداد نتایج مثبت اشتباه است، که اگر اندازه‌ی فیلتر بلوم برابر با '16N باشد به صفر میل می‌کند.

در حالی که دسترسی به یک نود trie ذخیره شده در جدول هشبرون تراشه ای، برای هر نتیجه‌ی مثبت فیلتر بلوم در الگوریتم NPT-BF میسر می‌شود، الگوریتم NPT-BF-زنجیره‌ای به صورت پیوسته، اجرای بررسی‌های فیلتر بلوم را بدون دستیابی به نود ادامه می‌دهد تا زمانی که فیلتر، یک نتیجه‌ی منفی تولید نماید (یا تا زمانی که همه‌ی مؤلفه‌های نام ورودی استفاده شوند).

برای اندازه‌ی '16N در فیلتر بلوم، میانگین تعداد دسترسی‌های نود برای الگوریتم NPT-BF برابر با 3.84 است که در الگوریتم NPT-BF-زنجیره‌ای بدون پیش‌محاسبه 1.67، و برای همین الگوریتم همراه با پیش‌محاسبه، 0.97 بهبود می‌یابد. تفاوت عملکرد در الگوریتم‌های NPT-BF-زنجیره‌ای با و بدون

پیش‌محاسبه، از عدم ردیابی مجدد در نودهای داخلی خالی ناشی می‌شود. توجه کنید که ورودی‌های دارای نتیجه‌ی منفی در سطح اول، نام‌های منطبقی ندارند و در نتیجه برای هیچ کدام از ورودی‌ها در الگوریتم‌های مبتنی بر فیلتر بلوم، دسترسی به جدول هش ممکن نخواهد بود.

جدول 6، 7 و 8، عملکرد جستجو را به ترتیب برای 50k، 100k و 300k نشان می‌دهد. تعداد میانگین دسترسی نود برای الگوریتم‌های NPT-BF-زنجیره‌ای همراه با پیش‌محاسبه در این مجموعه‌ها، 1 یا کمتر است. بدترین حالت نتایج مثبت اشتباه برای اندازه‌ی فیلتر بلوم '16N برابر با 1 می‌باشد. در نتیجه، بدترین حالت تعداد دسترسی به نود در این مجموعه‌ها، 2 است. توجه کنید که بدترین تعداد دسترسی به نود برای الگوریتم‌های NPT و NPT ترکیبی برابر با عمق trie بود و همانطور که در جدول 4 نشان داده شده، عدد بزرگی می‌باشد. برای الگوریتم NPT-BF، بدترین تعداد دسترسی نود یک واحد کمتر از عمق trie است، چرا که فیلتر بلوم از یک بار دسترسی به جدول هشبخاطر تولید یک نتیجه‌ی منفی، بعد از اینکه روند جستجو به انتهای trie می‌رسد، جلوگیری می‌کند.

شکل 5، تعداد میانگین دسترسی نود را در هر الگوریتم برای اندازه‌ی فیلتر بلوم '16N مقایسه می‌کند.

این شکل نشان می‌دهد که عملکرد جستجوی هر الگوریتم، نسبت به تعداد نام‌های محتوای ذخیره شده در پیشوند نام trie ارتباطی ندارد. الگوریتم NPT-BF-زنجیره‌ای همراه با پیش‌محاسبه، بهترین عملکرد را از خود نشان می‌دهد و می‌تواند وجه خروجی را از طریق یک دسترسی نود به صورت میانگین و در بدترین حالت، با دو دسترسی به نود، تعیین کند.

نتیجه‌گیری

این مقاله، الگوریتم‌های جدیدی برای بهبود عملکرد جستجوی FIB برای ارسال بسته در یک شبکه‌ی محتوی محور پیشنهاد می‌کند. روش پیشنهادی ما براساس این حقیقت است که دسترسی به یک حافظه‌ی برون تراشه ای زمان بیشتری نسبت به حافظه‌ی درون تراشه ای، نیاز دارد. ما برای کاهش دسترسی به جدول هش برون تراشه ای، یک فیلتر بلوم درون تراشه ای پیشنهاد کرده‌ایم. در نتیجه بین مقدار حافظه و تعداد بررسی‌ها برای فیلتر بلوم بده‌بستان وجود داشته و در روش پیشنهادی ما، تعداد دسترسی‌ها به جدول هشکاهش یافته است.

ما در ابتدا، پیشوند نام trie را توضیح داده‌ایم که برای جستجوی نام محتوا به کار رفته و می‌تواند با استفاده از معماری مبتنی بر هش؛ NPT -هش پیاده‌سازی شود. به منظور بهبود عملکرد جستجوی الگوریتم NPT ترکیبی، دو الگوریتم با استفاده از فیلترهای بلوم پیشنهاد می‌کنیم: NPT-BF و NPT-BF-زنجیره‌ای. هر دو الگوریتم روند تشکیل مشابهی دارند، اما روش‌های جستجو در آن‌ها متفاوت است. دسترسی‌های الگوریتم NPT-BF به جدول هشبرون تراشه ایبرای هر نتیجه‌ی مثبت فیلتر بلوم، صورت می‌گیرد. الگوریتم NPT-BF-زنجیره‌ای از این مسئله استفاده می‌کند که نرخ مثبت اشتباه فیلتر را می‌توان با افزایش اندازه‌ی آن، به صورت کافی کاهش داد. در نتیجه، الگوریتم NPT-BF-زنجیره‌ای، جستجو را تا سطح بعدی برای یک نتیجه‌ی مثبت در سطح فعلی ادامه می‌دهد تا جایی که فیلتر بلوم یک نتیجه‌ی منفی تولید نماید. چون نتایج منفی فیلتر بلوم همیشه صحیح هستند، الگوریتم NPT-BF-زنجیره‌ای در سطح قبلی، دسترسی به نود را ایجاد می‌کند که سطحی با جدیدترین نتیجه‌ی مثبت است. در این روش، الگوریتم NPT-BF-زنجیره‌ای می‌تواند طولانی‌ترین سطحی را که نود منطبق دارد، تعیین نماید. عملکرد جستجوی الگوریتم NPT-BF-زنجیره‌ای با پیش‌محاسبه‌ی وجه خروجی بهترین تطبیق نام برای نودهای داخلی، بهبود می‌یابد.

نتایج شبیه‌سازی نشان می‌دهد که الگوریتم NPT-BF-زنجیره‌ای همراه با پیش‌محاسبه، می‌تواند وجه خروجی هر نام ورودی را با کمتر از دسترسی تک گره ای به صورت میانگین و دسترسی دو گره ای در بدترین حالت با استفاده از اندازه‌ی قابل قبولی فیلتر بلوم، ارائه دهد.