|  |  |  |
| --- | --- | --- |
| Optimized_image_813b5ec2.png | دانشگاه تهران  پردیس دانشکده های فنی  دانشکده مهندسی کامپیوتر | Images__Logo_FE.gif |
|  | | |
|  | | |
| دسته‌بندی بسته در شبکه‌های نرم‌افزار محور  پايان‌نامه جهت دريافت درجه کارشناسی ارشد  در رشته مهندسی کامپیوتر گرايش معماری | | |
| نگارش:  **محمدرضا پیروزی** | | |
| استاد راهنما:  دکتر ناصر یزدانی | | |
| **بهمن ماه 1395** | | |



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Optimized_image_813b5ec2.png | دانشگاه تهران  پردیس دانشکده های فنی  دانشکده مهندسی کامپیوتر | | Images__Logo_FE.gif | |
|  | | | | |
|  | | | | |
| دسته‌بندی بسته در شبکه‌های نرم‌افزار محور  پايان‌نامه جهت دريافت درجه کارشناسی ارشد  در رشته مهندسی کامپیوتر گرايش معماری | | | | |
| نگارش:  **محمدرضا پیروزی** | | | | |
| استاد راهنما:  دکتر ناصر یزدانی | | | | |
|  | | | | |
| **بهمن ماه 1395** | | | | |
| tu | |  | | ArmFanni_Orginal |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **دانشگاه تهران**  **پردیس دانشکده­های فنی**  **دانشکده مهندسی برق و کامپیوتر**  **گواهی دفاع از پایان­نامه کارشناسی ارشد**  هیأت داوران پایان­نامه کارشناسی ارشد آقاي / خانم ....................................................................به شماره دانشجويي...........................در رشته ...................................... گرايش .................................. را در تاريخ ............................. با عنوان ............................................................................................   |  |  |  |  | | --- | --- | --- | --- | |  | به عدد | به حروف | | | با نمره نهايي |  |  | | |  |  |  | | | و درجه |  | | ارزيابي کرد. | | | | | | | | | |
|  | |  | |  | |  | | |
| **رديف** | **مشخصات هيئت داوران** | | **نام و نام‌ خانوادگي** | | **مرتبه دانشگاهي** | | **دانشگاه يا مؤسسه** | **امضاء** |
| **1** | **استاد راهنما**  **استاد راهنماي دوم (حسب مورد)** | |  | |  | |  |  |
| **2** | **استاد مشاور** | |  | |  | |  |  |
| **3** | **استاد داور داخلي** | |  | |  | |  |  |
| **4** | **استاد مدعو** | |  | |  | |  |  |
| **5** | **نماينده كميته تحصيلات تكميلي دانشكده / گروه** | |  | |  | |  |  |
| نام و نام خانوادگي معاون تحصيلات تكميلي و  پژوهشي دانشكده / گروه:  تاريخ و امضاء: | | | | | نام و نام خانوادگي معاون آموزشي و تحصيلات تكميلي  پرديس دانشكده‌هاي فني:  تاريخ و امضاء: | | | |

|  |
| --- |
| **تعهدنامه اصالت اثر** |
| **باسمه تعالي** |
| **اينجانب محمدرضا پیروزی تائيد مي‌كنم كه مطالب مندرج در اين پایان‌نامه حاصل كار پژوهشي اينجانب است و به دستاوردهاي پژوهشي ديگران كه در اين نوشته از آنها استفاده شده است مطابق مقررات ارجاع گرديده است. اين پایان‌نامه قبلاٌ براي احراز هيچ مدرك هم سطح يا بالاتر ارائه نشده است.**  **كليه حقوق مادي و معنوي اين اثر متعلق به دانشكده فني دانشگاه تهران مي باشد.**  **نام و نام خانوادگي دانشجو :**  **امضاي دانشجو :** |

**تقديم به پدر و مادرم**

**چکيده**

در شبکه‌های نرم‌افزار محور، یک جریان بر اساس تعداد دلخواهی از فیلدها در هر لایه از سرآیند بسته، قابل تعریف است. به عنوان مثال در نسخه 1.3.1 پروتکل OpenFlow از 15 فیلد برای تعریف یک جریان استفاده می‌شود. این امر سبب شده تا اولاً عرض مدخل در جدول جریان افزایش یابد، ثانیاً کنترل ریزتر بر روی ترافیک شبکه باعث افزایش تعداد مدخل‌ها در جدول جریان شده است. در نتیجه طراحی یک راه‌حل دسته‌بندی بسته در محیط‌هایی که به کارآیی بالا نیاز دارند، به یک چالش بزرگ تبدیل گشته است. برای حل این مسئله در سوئیچ‌های نرم‌افزاری سعی می‌شود تا حد امکان اندازه فضای نمونه برای مطابقت دادن یک چندتایی با قوانین، کوچک و کوچک‌تر شود. در الگوریتم MC-SBC یک ساختار مبتنی بر درخت تصمیم دو سطحی برای جدول‌های جستجو مطرح شده، و در آن بیت‌های موثر جهت افراز مجموعه قوانین و ایجاد درخت، با استفاده از یک روش آماری مشخص می‌گردند. در این پایان‌نامه از الگوریتم ژنتیک برای پیدا کردن مجموعه بیت‌های موثر جهت افراز مجموعه قوانین استفاده شده است. نتایج گزارش‌شده نشان می‌دهد مجموعه قوانین، با استفاده از این الگوریتم به صورت بهتری افراز می‌شوند، به طوری که هم نسبت تکرار قوانین در آن کاهش می‌یابد، و هم بیشینه تعداد قوانین در برگ‌های درخت کمتر می‌شود. بهبود کیفیت افراز مجموعه قوانین باعث می‌شود هنگامی که اندازه مجموعه قوانین بزرگ است، سرعت عمل دسته‌بندی بسته‌ها در نمونه پیاده‌سازی شده بر روی پردازنده گرافیکی تا حدود 10 درصد بهبود یابد.

**واژه‌هاي كليدي**: شبکه‌های نرم‌افزار محور، دسته‌بندی بسته، سوئیچ OpenFlow، پردازنده گرافیکی، الگوریتم ژنتیک

فهرست مطالب

[فصل 1: مقدمه 2](#_Toc473543796)

[1-1- تعریف مسئله 3](#_Toc473543797)

[1-2- روش انجام پژوهش 4](#_Toc473543798)

[1-3- ساختار پایان‌نامه 7](#_Toc473543799)

[فصل 2: زمینه‌های تحقیق 8](#_Toc473543800)

[2-1- شبکه‌های نرم‌افزار محور 8](#_Toc473543801)

[2-1-1- محدودیت‌های شبکه‌های سنتی 9](#_Toc473543802)

[2-1-2- تعریف شبکه نرم‌افزار محور 10](#_Toc473543803)

[2-1-3- معماری شبکه‌های نرم‌افزار محور 11](#_Toc473543804)

[2-1-4- چالش‌های شبکه‌های نرم‌افزار محور 13](#_Toc473543805)

[2-2- واسط OpenFlow 14](#_Toc473543806)

[2-2-1- اجزاء سوئیچ OpenFlow 15](#_Toc473543807)

[2-3- مسئله دسته‌بندی بسته‌ها 17](#_Toc473543808)

[2-3-2- تعریف رسمی مسئله دسته‌بندی بسته‌ها 19](#_Toc473543809)

[2-3-3- انواع عمل تطابق 19](#_Toc473543810)

[2-3-4- SDN و چالش دسته‌بندی بسته‌ها 20](#_Toc473543811)

[2-4- استفاده از پردازنده‌های گرافیکی برای دسته‌بندی بسته 22](#_Toc473543812)

[2-4-1- انجام محاسبات همه‌منظوره روی پردازنده‌های گرافیکی 22](#_Toc473543813)

[2-4-2- معماری پردازنده گرافیکی 23](#_Toc473543814)

[2-4-3- برنامه‌های کاربردی CUDA 26](#_Toc473543815)

[2-5- الگوریتم ژنتیک 27](#_Toc473543816)

[2-6- تولید مجموعه داده 28](#_Toc473543817)

[2-7- جمع‌بندی 31](#_Toc473543818)

[فصل3: مروری بر کارهای قبلی 33](#_Toc473543819)

[3-2- معیارهای سنجش الگوریتم‌های دسته‌بندی 35](#_Toc473543820)

[3-3- راه‌حل‌های استفاده کننده از ویژگی‌های SDN 36](#_Toc473543821)

[3-3-1- برچسب گذاری دو لایه‌ای 36](#_Toc473543822)

[3-3-2- برچسب گذاری در سوئیچ لبه 38](#_Toc473543823)

[3-3-3- پیکربندی پویای الگوریتم‎های دسته‌بندی 39](#_Toc473543824)

[3-4- حل مسئله به صورت الگوریتمی 41](#_Toc473543825)

[3-4-1- الگوریتم‌های ارائه شده بدون تاکید بر تعداد فیلد زیاد 43](#_Toc473543826)

[3-4-2- مدار پیش‌بینی کننده جریان 47](#_Toc473543827)

[3-4-3- تجزیه فیلد‌ها و ادغام نتایج جزئی 48](#_Toc473543828)

[3-4-4- GSwitch 52](#_Toc473543829)

[3-4-5- MC-SBC 57](#_Toc473543830)

[3-5- جمع‌بندی 60](#_Toc473543831)

[فصل4: استفاده از الگوریتم ژنتیک برای پیدا کردن بیت‌های موثر 61](#_Toc473543832)

[4-1- بررسی MC-SBC‌ با جزئیات بیشتر 61](#_Toc473543833)

[4-1-2- برنامه‌ریزی قانون به صورت برون‌خط 63](#_Toc473543834)

[4-1-3- فرآیند دسته‌بندی بسته به صورت برخط 64](#_Toc473543835)

[4-1-4- معیارهای انتخاب بیت‌های موثر 66](#_Toc473543836)

[4-1-5- نتایج MC-SBC 68](#_Toc473543837)

[4-2- استفاده از الگوریتم ژنتیک برای یافتن بیت‌های موثر 69](#_Toc473543838)

[4-2-2- مقداردهی اولیه 69](#_Toc473543839)

[4-2-3- انتخاب 70](#_Toc473543840)

[4-2-4- تقاطع و جهش 72](#_Toc473543841)

[4-2-5- شرط خاتمه الگوریتم 74](#_Toc473543842)

[4-3- نتایج بخش برون‌خط 74](#_Toc473543843)

[4-3-1- تعیین تعداد تکرار الگوریتم ژنتیک 74](#_Toc473543844)

[4-3-2- مقایسه دو الگوریتم از لحاظ نسبت تکرار و Binth 75](#_Toc473543845)

[4-3-3- زمان اجرای الگوریتم‌ها 81](#_Toc473543846)

[4-4- نتایج بخش بر‌خط 84](#_Toc473543847)

[4-4-2- بررسی تاثیر پارامترهای GPU در کارآیی 86](#_Toc473543848)

[4-5- جمع‌بندی 88](#_Toc473543849)

[فصل5: نتیجه‌گیری و کارهای آینده 89](#_Toc473543850)

[5-1- روش پیشنهادی و ارزیابی 90](#_Toc473543851)

[5-2- پژوهش‌های آینده 90](#_Toc473543852)

[مراجع 92](#_Toc473543853)

[واژه‌نامه انگلیسی به فارسی 95](#_Toc473543854)

[واژه‌نامه فارسی به انگلیسی 100](#_Toc473543855)

فهرست اشکال

[شکل (2-1) معماری شبکه‌های نرم‌افزار محور 12](#_Toc473543856)

[شکل (2-2) اجزاء اصلی سوئیچ OpenFlow 15](#_Toc473543857)

[شکل (2-3) یک شبکه نمونه که در آن سرویس‌های مختلفی برای بقیه فراهم می‌کند. 18](#_Toc473543858)

[شکل (2-4) معماری پردازنده‌های گرافیکی امروزی 24](#_Toc473543859)

[شکل (2-5) معماری چندپردازنده جریانی 25](#_Toc473543860)

[شکل (2-6) مثالی از زمان‌بندی wrap 25](#_Toc473543861)

[شکل (2-7) نمودار گردش کار الگوریتم ژنتیک 28](#_Toc473543862)

[شکل (2-8) نمودار بلوکی مجموعه ابزار ClassBench 30](#_Toc473543863)

[شکل (3-1) طبقه بندی روش‌های دسته‌بندی بسته بر اساس سخت‌افزاری یا نرم‌افزاری بودن 34](#_Toc473543864)

[شکل (3-2) یک گراف نمونه و مسیرهای مختلف 37](#_Toc473543865)

[شکل (3-3) نگاشت جریان به مسیر 38](#_Toc473543866)

[شکل (3-4) معماری سیستم مطرح شده در [9] 39](#_Toc473543867)

[شکل (3-5) معماری سخت‌افزار مطرح شده در [10] 40](#_Toc473543868)

[شکل (3-6) خط لوله فرآیند جستجو 41](#_Toc473543869)

[شکل (3-7) مثالی از ساختار درختواره‌های سلسله مراتبی 44](#_Toc473543870)

[شکل (3-8) مثالی از ساختار درختواره‌های با شاخه‌های هرس‌شده 45](#_Toc473543871)

[شکل (3-9) مثالی از ساختار AQT 45](#_Toc473543872)

[شکل (3-10) ساختار درخت در روش HiCut 46](#_Toc473543873)

[شکل (3-11) مدار پیش‌بینی کننده جریان به ازای هر پورت 47](#_Toc473543874)

[شکل (3-12) ذخیره مجموعه‌های RID در میزبان و استفاده از اندیس در GPU 50](#_Toc473543875)

[شکل (3-13) انتقال داده ناهمگام بین CPU، و GPU 51](#_Toc473543876)

[شکل (3-14) عملیات درج قوانین در Tuple Space Search 53](#_Toc473543877)

[شکل (3-15) نحوه نگاشت بین کلاس‌ها و چندتایی‌ها به بلوک‌ها و ریسمان‌های GPU 55](#_Toc473543878)

[شکل (3-16) معماری دو مرحله‌ای ارائه شده برای دسته‌بندی بسته در MC-SBC 58](#_Toc473543879)

[شکل (3-17) مثالی از ساختار داده MC-SBC 59](#_Toc473543880)

[شکل (4-1) معماری دو مرحله‌ای ارائه شده برای دسته‌بندی بسته در MC-SBC 62](#_Toc473543881)

[شکل (4-2) مثالی از ساختار داده MC-SBC 63](#_Toc473543882)

[شکل (4-3) نتایج بخش برون‌خط MC-SBC 68](#_Toc473543883)

[شکل (4-4) یک نمونه کروموزوم شامل بیت‌های موثر 69](#_Toc473543884)

[شکل (4-5) انجام عمل تقاطع روی دو کروموزوم منتخب و 72](#_Toc473543885)

[شکل (4-6) شماره تکرار مشخص شدن بهترین کروموزوم برای تعداد بیت‌های موثر مختلف 75](#_Toc473543886)

[شکل (4-7) نمودار نسبت تکرار و Binth برای مجموعه قوانین IPC با نسبت Wildcard مختلف 76](#_Toc473543887)

[شکل (4-8) نمودار نسبت تکرار و Binth برای مجموعه قوانین ACL با نسبت Wildcard مختلف 76](#_Toc473543888)

[شکل (4-9) نمودار نسبت تکرار و Binth برای مجموعه قوانین FW با نسبت Wildcard مختلف 77](#_Toc473543889)

[شکل (4-10) نمودار نسبت تکرار و Binth برای مجموعه قوانین IPC با تغییر تعداد زیرمجموعه‌ها 77](#_Toc473543890)

[شکل (4-11)نمودار نسبت تکرار و Binth برای مجموعه قوانین ACL با تغییر تعداد زیرمجموعه‌ها 78](#_Toc473543891)

[شکل (4-12) نمودار نسبت تکرار و Binth برای مجموعه قوانین FW با تغییر تعداد زیرمجموعه‌ها 78](#_Toc473543892)

[شکل (4-13) نمودار کمینه، میانگین، و بیشینه تعداد قوانین در برگ‌ها بر اساس نسبت Wildcard برای ACL\_100k 80](#_Toc473543893)

[شکل (4-14) نمودار کمینه، میانگین، و بیشینه تعداد قوانین در برگ‌ها بر اساس نسبت Wildcard برای FW\_100k 80](#_Toc473543894)

[شکل (4-15) نمودار کمینه، میانگین، و بیشینه تعداد قوانین در برگ‌ها بر اساس نسبت Wildcard برای IPC\_100k 81](#_Toc473543895)

[شکل (4-16) زمان اجرای الگوریتم برون‌خط MC-SBC، و ژنتیک 84](#_Toc473543896)

[شکل (4-17) تاخیر دسته‌ای بخش بر‌خط 85](#_Toc473543897)

[شکل (4-18) نمودار تاخیر دسته‌ای بر اساس تعداد بلوک 86](#_Toc473543898)

[شکل (4-19) نمودار تاخیر بر اساس تعداد ریسمان‌ها در هر بلوک 87](#_Toc473543899)

[شکل (4-20) نمودار تاخیر بر اساس اندازه دسته 87](#_Toc473543900)

فهرست جداول

[جدول (2-1) تغییرات نسخه‌های اصلی OpenFlow 16](#_Toc473543901)

[جدول (2-2) سرویس‌های ارائه شده توسط برای مشتریان 18](#_Toc473543902)

[جدول (2-3) جریان‌های مربوط به بسته‌های ورودی از پورت X 19](#_Toc473543903)

[جدول (2-4) فیلدهای سرآیند بسته که در OpenFlow پشتیبانی می‌شوند. 21](#_Toc473543904)

[جدول (2-5) نسبت تعداد مقادیر یکتا به تعداد کل قوانین در مجموعه قوانین 31](#_Toc473543905)

[جدول (3-1) نتایج به دست آمده در [12] 52](#_Toc473543906)

[جدول (3-2) گذردهی بخش برخط در MC-SBC 59](#_Toc473543907)

[جدول (4-1) گذردهی بخش برخط در MC-SBC 69](#_Toc473543908)

[جدول (4-2) کمینه، میانگین، و بیشینه تعداد قوانین در برگ‌ها برای مجموعه قوانین ACL 79](#_Toc473543909)

[جدول (4-3) کمینه، میانگین، و بیشینه تعداد قوانین در برگ‌ها برای مجموعه قوانین FW 79](#_Toc473543910)

[جدول (4-4) کمینه، میانگین، و بیشینه تعداد قوانین در برگ‌ها برای مجموعه قوانین IPC 80](#_Toc473543911)

[جدول (4-5) مشخصات پردازنده گرافیکی استفاده شده 85](#_Toc473543912)

1. فصل 1: مقدمه

شبکه‌های IP سنتی پیچیده هستند، و مدیریت آن‌ها دشوار است [1]. در این شبکه ها مدیر شبکه برای بیان سیاست‌های سطح بالای دلخواه خود، می‌بایست هر دستگاه شبکه را به صورت جداگانه و با استفاده از دستورات سطح پایین و مخصوص آن دستگاه، پیکربندی کند. علاوه بر این پیچیدگی‌ها، شبکه‌ها باید اشکالات پویا را تحمل کرده و خود را با تغییر بار سازگار سازند [2].

این شبکه‌ها همچنین به صورت عمودی تجمیع شده هستند. به این معنی که سطح کنترل[[1]](#footnote-1) - که تصمیم می‌گیرد چه رفتاری با ترافیک شبکه صورت گیرد- و سطح داده[[2]](#footnote-2) - که ترافیک را بر اساس تصمیمات گرفته شده در سطح کنترل ارسال می‌کند - با هم در دستگاه‌های شبکه قرار می‌گیرند. این امر باعث کاهش انعطاف‌پذیری، و مانع نوآوری و تکامل در زیرساختار شبکه می‌گردد. گذار از IPv4 به IPv6 که بیش از یک دهه قبل آغاز شده، و هنوز بسیار ناکامل است، یک شاهد خوب برای این چالش می‌باشد. این در حالی است که IPv6 صرفاً یک بروزرسانی پروتکل می‌باشد. به دلیل لختی[[3]](#footnote-3) شبکه‌های IP کنونی، طراحی، ارزیابی، و گسترش یک پروتکل مسیریابی جدید، ممکن است حدود 5 الی 10 سال به طول بیانجامد [2].

برای حل مشکلات و برطرف کردن محدودیت‌های شبکه‌های سنتی، یک ساختار به نام شبکه‌های نرم‌افزار محور[[4]](#footnote-4) (SDN) ارائه شده است [3]. طبق تعریف ONF‌[[5]](#footnote-5)، SDN یک معماری شبکه در حال ظهور می‌باشد، که در آن بخش‌کنترلی شبکه از بخش پیشرانی[[6]](#footnote-6) که به طور مستقیم قابل برنامه‌ریزی[[7]](#footnote-7) است، جدا شده [4]. در [2] 4 رکن اصلی SDN به صورت زیر ذکر شده است:

1. سطوح کنترل و داده از هم جدا شده‌اند.
2. تصمیمات پیشرانی مبتنی بر جریان[[8]](#footnote-8) اتخاذ می‌گردد.
3. منطق کنترلی به یک موجودیت خارجی به نام کنترلر انتقال یافته است.
4. شبکه از طریق برنامه‌های نرم‌افزاری که روی کنترلر اجرا می‌شوند، قابل برنامه‌ریزی است.

با استفاده از معماری SDN، می‌توان از محصولات شرکت‌های مختلف در یک شبکه استفاده کرد. هم‌چنین برای توسعه سرویس‌های هوشمند جدید، نیاز نیست مدیران منتظر انتشار ویژگی‌های جدید از طرف شرکت سازنده بمانند. آن‌ها می‌توانند این کار را به سرعت و مستقل انجام دهند [5].

* 1. تعریف مسئله

تفاوت اصلی بین دستگاه‌های شبکه در SDN و شبکه‌های سنتی، این است که در SDN این دستگاه‌ها صرفاً عناصر ساده پیشرانی هستند که هیچ‌گونه کنترل نهفته یا نرم‌افزاری برای اتخاذ تصمیمات خودمختار[[9]](#footnote-9) ندارند [2]. این کنترلر است که جریان‌ها را در جدول جریان، درج، حذف، و یا بروزرسانی می‌کند. بسته‌ها به محض ورود به این دستگاه‌ها با تعدادی قانون که در جدول جریان[[10]](#footnote-10) قرار دارند، مطابقت[[11]](#footnote-11) داده می‌شوند، تا جریانی که بسته به آن تعلق دارد تعیین شده و در نتیجه، نحوه رفتار با آن بسته مشخص گردد. به همین دلیل دسته‌بندی بسته یکی از کارهای اساسی[[12]](#footnote-12) در این دستگاه‌ها می‌باشد [6]. از طرفی دیگر در SDN یک جریان بر اساس تعداد دلخواهی از فیلدها در هر لایه دلخواه از سرآیند بسته، قابل تعریف است. به عنوان مثال در نسخه 1.3.1 پروتکل OpenFlow از 15 فیلد (356 بیت) برای تعریف یک جریان استفاده می‌شود. این مسئله می‌تواند سوئیچ‌های SDN را از دو جهت تحت تاثیر قرار دهد. اولاً اندازه هر مدخل جریان بزرگ می‌شود. ثانیاً، با ایجاد امکان تعریف جزئی‌تر جریان‌ها، تعداد جریان‌ها در جدول افزایش می‌یابد. نکته دیگر آن که به دلیل توسعه اینترنت، دستگاه‌های شبکه باید بتوانند از برنامه‌های کاربردی مختلف مانند دیوار آتش[[13]](#footnote-13)، و تضمین کیفیت سرویس[[14]](#footnote-14)، پشتیبانی کنند و این امر سبب شده تا دسته‌بندی بسته[[15]](#footnote-15) یک کار مهم برای مدیریت شبکه قلمداد شود [7].

به طور کلی چالش‌هایی که مسئله دسته‌بندی بسته با آن‌ها روبرو است، عبارت‌اند از [7]:

* گسترش عرض و تعداد مجموعه قوانین.
* افزایش پیچیدگی مجموعه قوانین.
* افزایش درخواست برای گذردهی[[16]](#footnote-16) بالا و توان مصرفی کم.

لذا استفاده از حافظه‌های TCAM در سوئیچ‌های سخت‌افزاری، هر چند کارآیی بسیار خوبی دارد (زمان اجرای )، اما به دلیل افزایش بیش از حد توان مصرفی و فضای سخت‌افزاری به دلیل گسترش اندازه جدول جریان، به صرفه نخواهند بود. گسترش اندازه جدول جریان در سوئیچ‌های نرم‌افزاری باعث افت کارآیی سلسله مراتب حافظه شده و کارآیی کلی سیستم را کاهش می‌دهد. بنابراین می‌بایست یک بازنگری در نحوه مدیریت جدول جریان در این سوئیچ‌ها صورت گیرد.

* 1. روش انجام پژوهش

راهکارهای ارائه شده برای بهبود کارآیی مسئله دسته‌بندی بسته را از آن جهت که از ویژگی‌های ارائه شده توسط SDN استفاده کرده‌اند یا خیر، می‌توان به دو دسته تقسیم‌بندی نمود. در دسته اول سعی شده از وجود یک کنترلر مرکزی و ارائه قابلیت برنامه‌ریزی شبکه توسط آن بهره برده شود. برای مثال در [8] و [9] ایده‌ای مشابه با ایده پروتکل MPLS‌[[17]](#footnote-17) مطرح شده، و از تکنیک برچسب‌گذاری بر روی بسته‌ها برای کاهش اندازه جدول جریان استفاده شده است. به طوری که سوئیچ‌های لبه عمل برچسب‌گذاری را انجام می‌دهند، و کنترلر این برچسب‌ها را به سوئیچ‌های هسته انتقال می‌دهد. در نتیجه اندازه جدول جریان در سوئیچ‌های هسته بسیار کاهش می‌یابد. در [10] الگوریتم‌ دسته‌بندی بسته در سوئیچ‌ها به صورت پویا توسط کنترلر، جهت سازگاری با بار کاری شبکه، پیکربندی و در هر لحظه بهترین الگوریتم برای انجام این کار انتخاب می‌شود.

در دسته دوم سعی می‌شود به صورت الگوریتمی، یا با استفاده از مکانیزم‌های جانبی کارآیی مسئله دسته‌بندی بهبود داده شود. خود این راهکارها از آن جهت که برای سوئیچ‌های سخت‌افزاری ارائه شده‌ باشند یا نرم‌افزاری، به دو دسته تقسیم‌بندی می‌گردند. در [11] یک مدار پیش‌بینی کننده جریان برای هر واسط شبکه به سیستم اضافه شده و به این ترتیب تعداد دسترسی‌ها به حافظه TCAM و توان مصرفی به مراتب کاهش یافته است. در [12] یک روش مبتنی بر تجزیه بر روی سکوهای مختلف مانند FPGA‌[[18]](#footnote-18)، پردازنده گرافیکی، و پردازنده‌های چند‌هسته‌ای پیاده‌سازی و با هم مقایسه شده است. در [13] یک نسخه از الگوریتم‌های جستجوی خطی، و جستجوی فضای چندتایی[[19]](#footnote-19)، که در سوئیچ‌های نرم‌افزاری گسترش یافته‌اند، بر روی پردازنده گرافیکی پیاده‌سازی شده و کارآیی به طور قابل توجهی بهبود یافته است.

سوئیچ‌های سخت‌افزاری به طور کلی کارآیی بهتری نسبت به سوئیچ‌های نرم‌افزاری دارند، با این حال انعطاف‌پذیری آن‌ها کمتر است و به زمان بیشتری برای توسعه نیاز دارند. این سوئیچ‌ها هم‌چنین از نظر تعداد قوانین، معمولاً مقیاس‌پذیری کمتری دارند. در مقابل زمان توسعه سوئیچ‌های نرم‌افزاری کمتر است و به سرعت می‌توان آن‌ها را با تغییر نیازمندی‌ها سازگار ساخت. از طرفی با پیشرفت سخت‌افزارهای همه‌منظوره، کارآیی این سوئیچ‌ها به قدری افزایش پیدا کرده که توانسته است توجه افراد بسیاری را در محیط‌های صنعتی و علمی به خود جلب کند.

در این پژوهش به دلیل ویژگی‌های توسعه سریع و انعطاف‌پذیری بالا که در سوئیچ‌های نرم‌افزاری وجود دارد، تمرکز خود را به مسئله دسته‌بندی بسته در این سوئیچ‌ها معطوف کرده‌ایم. ابتدا راهکارهای ارائه شده برای آن‌ها را با جزئیات بیشتری بررسی کرده و سپس از الگوریتم ژنتیک برای بهبود کیفیت افراز مجموعه قوانین در الگوریتم MC-SBC‌[[20]](#footnote-20) بهره جسته‌ایم.

در [14]، یک معماری دو مرحله‌ای به نام MC-SBC برای دسته‌بندی بسته ارائه شده است؛ به طوری که در مرحله اول یک ساختار مبتنی بر درخت تصمیم دو سطحی برای جدول‌های جستجو به صورت برون‌خط[[21]](#footnote-21) ایجاد می‌گردد. در مرحله دوم، دراین ساختار بسته‌ها به صورت برخط[[22]](#footnote-22) با قوانین مطابقت داده می‌شوند.

MC-SBC در مرحله برون‌خط از یک روش آماری برای مشخص کردن بیت‌های موثر جهت افراز مجموعه قوانین و ایجاد درخت استفاده می‌کند، و برای انجام این کار از سه معیار نسبت Wildcard‌[[23]](#footnote-23)، شاخص استقلال[[24]](#footnote-24)، و شاخص تنوع[[25]](#footnote-25) در مجموعه قوانین بهره می‌گیرد. بیت‌های موثر، بیت‌هایی هستند که با استفاده از آن‌ها می‌توان به بهترین صورت بین قوانین تمایز قایل شد؛ یعنی بیت‌هایی که تفاوت در مقدار آن‌ها، مجموعه قوانین را بهتر افراز می‌کند. در مرحله برخط، از مقدار بیت‌های موثر در سرآیند بسته به عنوان اندیس در ساختار درخت استفاده شده و به این ترتیب در نهایت تعداد اندکی از قوانین با بسته مورد نظر مطابقت داده می‌شوند. بنابر ادعای صورت گرفته در [14] این الگوریتم نسبت به دیگر روش‌های مبتنی بر درخت تصمیم مانند BC[15]، SBC[16]، HiCuts[17]، و HyperCuts[18] کارآیی بهتری دارد.

در این پایان‌نامه از الگوریتم ژنتیک برای یافتن مجموعه بیت‌های موثر در مرحله برون‌خط MC-SBC استفاده کرده و نشان داده‌ایم که وقتی نسبت Wildcard در مجموعه قوانین بیشتر می‌شود، کیفیت افراز توسط الگوریتم ژنتیک بهتر از روش آماری استفاده شده در [14] است.

برای ارزیابی روش ارائه شده ابتدا با استفاده از یک مجموعه ابزار محک‌زنی[[26]](#footnote-26) برای الگوریتم‌ها و دستگاه‌های دسته‌بندی بسته به نام ClassBench [19] مجموعه قوانین برای 5 فیلد سنتی (آدرس IP مبدا و مقصد، شماره پورت مبدا و مقصد، پروتکل) تولید شده و سپس 10 فیلد دیگر با نسبت‌‌های Wildcard 0، 0.1، 1، و 5 و تنظمیات مقادیر یکتا بر اساس [6] به مجموعه قوانین اضافه شده‌اند. هم‌چنین برای ارزیابی بخش برخط، نمونه‌ای از مرحله برخط الگوریتم بر روی یک پردازنده گرافیکی پیاده‌سازی شده است. نتایج نشان می‌دهند که وقتی نسبت Wildcard در مجموعه قوانین کم است (0 و 0.1)، کیفیت افراز مجموعه قوانین در هر دو الگوریتم تقریبا یکسان است. با این حال با افزایش نسبت Wildcard، الگوریتم ژنتیک به صورت بهتری می‌تواند آن را افراز کند. به طوری که هم نسبت تکرار در آن کاهش می‌یابد، و هم بیشینه تعداد قوانین در برگ‌های درخت کمتر می‌شود. بهبود کیفیت افراز مجموعه قوانین در مرحله برون‌خط، باعث می‌شود تا هنگامی که اندازه مجموعه قوانین بزرگ است، سرعت عمل دسته‌بندی بسته در مرحله برخط تا حدود 10 درصد بیشتر شود.

* 1. ساختار پایان‌نامه

ادامه این پایان‌نامه بدین صورت سازمان‌دهی شده است: در فصل دوم ابتدا توضیحاتی راجع به شبکه‌های نرم‌افزار محور و سوئیچ OpenFlow داده شده و به بررسی مسئله دسته‌بندی بسته و چالش آن در این زمینه پرداخته‌ شده است. در ادامه، مطالبی درباره پردازنده‌های گرافیکی همه منظوره و کاربرد آن‌ها در حل مسائل محاسباتی و به طور خاص در شبکه‌های کامپیوتری ارائه شده و همچنین الگوریتم ژنتیک به صورت مختصر شرح داده‌ شده است. در انتهای فصل، نحوه تولید مجموعه داده‌ها توضیح داده شده است. در فصل سوم به مرور پژوهش‌های صورت گرفته برای بهبود کارآیی دسته‌بندی بسته در سوئیچ‌های SDN پرداخته شده، و پس از ارائه یک دسته‌بندی از راه‌حل‌ها، تمرکز بیشتری در حل این مسئله به صورت الگوریتمی و پیاده‌سازی آن روی پردازنده‌های گرافیکی شده است. در فصل چهارم، ابتدا الگوریتم MC-SBC با جزئیات بیشتری بررسی شده و سپس نحوه استفاده از الگوریتم ژنتیک برای یافتن بیت‌های موثر توضیح داده‌ شده است. سپس کیفیت افراز مجموعه قوانین در بخش برخط، توسط الگوریتم ژنتیک و روش آماری به کار گرفته شده در MC-SBC مقایسه شده و تاثیر استفاده از هر یک از آن‌ها مورد بررسی قرار داده شده است. در پایان این فصل تاثیر پارامتر‌های مختلف در اجرای مرحله برخط الگوریتم، بر روی پردازنده گرافیکی در کارآیی سیستم را آورده‌ایم. در نهایت در فصل پنجم جمع‌بندی خود را ارائه کرده، و به ذکر کارهایی که در آینده می‌توان در این زمینه انجام داد پرداخته‌ایم.

1. فصل 2: زمینه‌های تحقیق

در این فصل مطالبی راجع به مباحث پایه‌ای مورد نیاز برای ورود به حوزه تحقیقات و روش‌های ارائه شده، مطرح شده است. ابتدا به مباحث مربوط به شبکه‌های نرم‌افزار محور، معماری SDN، چالش‌هایی که SDN با آن روبرو است و معماری سوئیچ OpenFlow، به عنوان واسطی که بسیار مورد قبول و گسترش واقع شده، پرداخته‌ایم. سپس مسئله دسته‌بندی بسته[[27]](#footnote-27) را تعریف کرده، و می‌بینیم که SDN چگونه با گسترش عرضی و عمقی جدول جریان، مجدداً این مسئله را به یک چالش تبدیل کرده است. در این پایان‌نامه از الگوریتم ژنتیک برای بهبود نحوه افراز مجموعه قوانین بهره برده و از پردازنده‌های گرافیکی همه منظوره به عنوان سکوی پردازشی استفاده و نمونه‌ای از روش ارائه شده را روی آن پیاده‌سازی و تست کرده‌ایم. بنابراین در ادامه فصل مطالبی درباره این دو مبحث مطرح شده و در نهایت نحوه تولید مجموعه داده برای ارزیابی روش‌‌های ارائه شده را توضیح داده‌ایم.

* 1. شبکه‌های نرم‌افزار محور

اخیراً SDN یکی از موضوعات بسیار محبوب در دنیای ICT‌[[28]](#footnote-28) شده است. با این حال، چون موضوع جدیدی است، اجماعی روی تعریف دقیق آن صورت نگرفته و در سال‌های اخیر تعاریف متعددی برای آن بیان شده که هر کدام در جای خود قابل تقدیر است [20]. در این بخش ابتدا محدودیت‌های شبکه‌های سنتی که منجر به ظهور SDN شده بیان گشته، و سپس تعریفی از SDN که مورد قبول عموم واقع شده است ارائه شده است. در ادامه معماری SDN و چالش‌های آن مطرح شده است.

* + 1. محدودیت‌های شبکه‌های سنتی

در روش سنتی برای شبکه‌سازی توابع شبکه عمدتاً در در دستگاه‌های[[29]](#footnote-29) اختصاصی و روی سخت‌افزارهای خاص‌منظوره، پیاده‌سازی می‌شوند. این روش سخت‌افزاری محدودیت‌هایی دارد که سازمان‌ها به طور روزافزونی با آن‌ها مواجه می‌شوند. در ادامه به برخی از این محدودیت‌ها اشاره شده است [21].

* پیاده‌سازی‌های سنتی زمان بر و مستعد خطا هستند.

در شبکه‌های سنتی، مدیر شبکه برای اضافه یا حذف کردن یک دستگاه باید گام‌های زیادی را طی کند. برای این کار او باید چندین دستگاه (سوئیچ، مسیریاب، دیوار آتش) را به صورت یکی یکی و دستی پیکربندی کند. در گام بعدی نیاز است تا تنظیمات پیکربندی متعددی مانند ACLs، VLANs، و QoS را با استفاده از ابزارهای مدیریت در سطح دستگاه، بروزرسانی گرداند. این نوع پیکربندی برای مدیری که می‌خواهد مجموعه‌ای از سیاست‌ها را در یک شبکه گسترش دهد، پیچیده‌تر می‌شود [21].

* استفاده از محصولات چند شرکت در یک جا، به تخصص بالایی نیاز دارد.

یک سازمان متوسط و نه چندان بزرگ دارای محصولات متنوعی از شرکت‌های مختلف است. برای پیکربندی موفق شبکه در آن سازمان، مدیر شبکه باید دانش زیادی درباره انواع دستگاه‌های موجود در آن جا داشته باشد [21].

* معماری قدیمی قطعه‌بندی[[30]](#footnote-30) شبکه را پیچیده می‌کند.

علاوه بر تبلت‌ها، کامپیوترهای شخصی، و گوشی‌های هوشمند، دستگاه‌های دیگر مانند سیستم‌های هشداردهنده و دوربین‌های امنیتی به زودی به شبکه اینترنت متصل خواهند شد. افزایش تعداد دستگاه‌های هوشمند، چالش‌های جدیدی برای سازمان‌ها به همراه خواهد داشت. برای مثال این که چگونه تمامی این دستگاه‌ها به صورت امن و ساخت‌یافته در شبکه جای داده شوند، یک مسئله جدی است.

خیلی از شبکه‌های سنتی تمامی دستگاه‌ها را در یک منطقه[[31]](#footnote-31) قرار می‌دهند. در این صورت اگر دستگاه‌ ناامنی در شبکه وجود داشته باشد، خطر اعطای دسترسی کل شبکه به بخش‌های خارجی وجود خواهد داشت. همچنین هکرها می‌توانند با بهره‌گیری از قابلیت اتصال به شبکه اینترنت که در دستگاه‌های هوشمند وجود دارد، به بقیه دستگاه‌ها ورود پیدا کنند. با این حال، در این شبکه‌ها انجام عمل قطعه‌بندی یک فرآیند پیچیده است و به سرعت به در هم ریختگی شبکه منجر می‌شود.

برای غلبه بر محدودیت‌های فوق و سایر محدودیت‌ها، زمان آن رسیده که چشم انداز جدیدی برای مدیریت شبکه ارائه گردد.

* + 1. تعریف شبکه نرم‌افزار محور

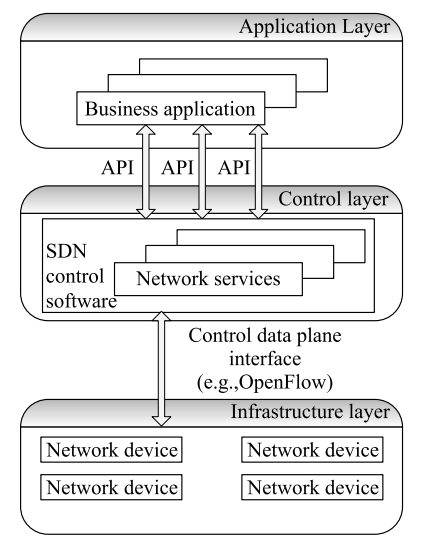
بنابر تعریف ONF‌[[32]](#footnote-32)، SDN یک معماری در حال ظهور برای شبکه می‌باشد که در آن بخش‌کنترلی از بخش پیشرانی[[33]](#footnote-33) -که به طور مستقیم قابل برنامه‌ریزی است- جدا شده است [4]. در [2] یک شبکه SDN به عنوان یک معماری شبکه با 4 رکن زیر تعریف شده است:

1. در SDN سطح کنترل از سطح داده جدا شده است. منطق کنترلی از دستگاه‌های شبکه حذف شده و این دستگاه‌ها به عناصر ساده پیشرانی تبدیل شده‌اند.
2. در SDN، تصمیمات پیشرانی، به جای آن که مبتنی بر مقصد[[34]](#footnote-34) باشد، مبتنی بر جریان[[35]](#footnote-35) است. یک جریان عموماً به صورت مجموعه‌ای از مقادیر فیلدهای سرآیند بسته به عنوان معیارهای مطابقت، و یک مجموعه از عمل‌ها[[36]](#footnote-36) (دستورات) تعریف می‌شود. تمامی بسته‌های یک جریان سرویس‌های یکسانی را در دستگاه‌های پیشران دریافت می‌کنند. این مفهوم جریان اجازه یک‌دست‌سازی[[37]](#footnote-37) نحوه رفتار با دستگاه‌های مختلف در شبکه، شامل مسیریاب‌ها، سوئیچ‌ها، دیواره‌های آتش، و ... را به ما می‌دهد.
3. در SDN منطق کنترلی به یک موجودیت خارجی انتقال داده شده، که به آن کنترلر[[38]](#footnote-38) SDN گفته می‌شود. کنترلر یک سکوی[[39]](#footnote-39) نرم‌افزاری است، که روی سرورهای موجود و در دسترس اجرا می‌شود، و منابع اصلی و لایه‌های تجرید[[40]](#footnote-40) را برای برنامه‌نویسی دستگاه‌های پیشران، مهیا می‌کند. بنابراین هدف آن مشابه سیستم عامل سنتی است.
4. در SDN شبکه از طریق برنامه‌های کاربردی[[41]](#footnote-41) نرم‌افزاری که روی کنترلر اجرا می‌شوند و در تعامل با دستگاه‌های سطح داده هستند، قابل برنامه‌ریزی است. این یک مشخصه اساسی برای SDN است.

توجه داشته باشید که مرکزی‌سازی منطقی[[42]](#footnote-42) سطح کنترل، چندین منفعت اضافی به همراه دارد: اولاً تغییر سیاست‌های شبکه از طریق زبان‌های سطح بالا و اجزاء نرم‌افزاری ساده‌تر بوده و با خطای کمتری انجام می‌شود (در مقایسه با پیکربندی‌های سطح پایین مخصوص هر دستگاه در شبکه‌های سنتی). ثانیاً یک برنامه کنترلی می‌تواند به صورت خودکار به تغییرات اشتباه در حالت شبکه عکس‌العمل نشان دهد، و سلامت سیاست‌های سطح بالا را حفظ کند. ثالثاً مرکزی‌سازی منطق کنترلی در یک کنترلر با دانش سراسری از حالت شبکه[[43]](#footnote-43)، توسعه توابع، سرویس‌ها، و برنامه‌های کاربردی بسیار پیچیده را آسان می‌سازد.

* + 1. معماری شبکه‌های نرم‌افزار محور

‏شکل (2-1) ساختار پایه‌ای SDN را نشان می‌دهد. معماری SDN به سه لایه تقسیم می‌شود. پایین‌ترین لایه، لایه زیرساختار[[44]](#footnote-44) است که شامل تمامی دستگاه‌های شبکه می‌گردد. بر خلاف شبکه‌های سنتی، دستگاه‌ها در شبکه SDN توابع کنترلی ندارند و به صورت دستگاه‌های ساده و بدون هوش هستند، که صرفاً عمل ارسال بسته‌ها را انجام می‌دهند. توابع کنترلی دستگاه‌ها در یک لایه کنترلی یکپارچه قرار می‌گیرد. لایه زیرساختار با لایه کنترلی از طریق واسط برنامه‌نویسی کاربردی (API) که تحت عنوان واسط جنوبی[[45]](#footnote-45) شناخته می‌شود، تعامل می‌کنند [22].



معماری شبکه‌های نرم‌افزار محور

در SDN تمامی هوش شبکه به صورت منطقی در یک لایه کنترلی متمرکزشده[[46]](#footnote-46) قرار می‌گیرد. این لایه شامل چندین کنترلر SDN می‌باشد. کنترلرهای SDN شبکه فیزیکی یا مجازی زیرین را مدیریت و سریس‌هایی را برای لایه بالایی (لایه کاربرد) از طریق واسط شمالی[[47]](#footnote-47) فراهم می‌کنند. لایه کنترل جزئیات مخصوص دستگاه‌ها را از طریق ایجاد تجرید، از برنامه‌های کاربردی مخفی می‌کند.

در لایه کاربرد[[48]](#footnote-48)، اپراتورهای شبکه و توسعه‌دهندگان برنامه‌های کاربردی، شبکه را از طریق واسط‌های برنامه‌نویسی به منظور تحقق نیازمندی‌های تجاری خود، مدیریت می‌کنند. این نیازمندی‌ها ممکن است شامل مدیریت کنترل دسترسی، پهنای باند، مهندسی ترافیک، کیفیت سرویس و ... باشد. به این ترتیب نیاز به پیکربندی دستگاه‌ها که درشبکه‌های سنتی صورت می‌گرفت و بسیار مستعد خطا بود، در این معماری حذف می‌شود [22].

* + 1. چالش‌های شبکه‌های نرم‌افزار محور

هر چند SDN یک راه‌حل مطلوب و مناسب برای دنیای IT است، با این حال با چالش‌هایی روبرو است که مانع از کارآیی و پیاده‌سازی آن می‌شود. مقیاس‌پذیری[[49]](#footnote-49)، کارآیی[[50]](#footnote-50)، امنیت[[51]](#footnote-51)، قابلیت اطمینان[[52]](#footnote-52) و ... برخی از این چالش‌ها می‌باشند [23].

* استانداردسازی

هر چند تعریف ONF از SDN بیشتر مورد قبول واقع شده است، با این حال OpenFlow، که توسط ONF حمایت می‌شود، به هیچ وجه تنها استاندارد SDN، و یک راه‌حل بالغ نیست. هنوز یک راه‌انداز[[53]](#footnote-53) متن-باز OpenFlow برای توسعه‌دهندگان کنترلر SDN، یا یک API شمالی و یک زبان برنامه‌نویسی سطح بالای استاندارد برای توسعه‌دهندگان برنامه‌های کاربردی SDN وجود ندارد [20].

* مقیاس‌پذیری

اگر چه مدیران شبکه، کارآیی کنترلرها را افزایش داده‌اند، اما مقدار آن هنوز به اندازه‌ای نیست که پاسخ‌گوی نیازهای یک شبکه بزرگ باشد. برای مثال یکی از کنترلرهای اولیه SDN، به نام NOX، تنها می‌تواند به 30 هزار درخواست جریان در هر ثانیه با تاخیر کمتر از 10 میلی‌ثانیه پاسخ دهد. این نارسایی در شبکه‌های بزرگ و مراکز داده، نسبت به شبکه‌های کوچک‌تر بیشتر مشاهده می‌شود. برای مثال یک خوشه از 1500 سرور، به طور میانگین 100 هزار درخواست جریان در هر ثانیه دریافت می‌کند. هم چنین یک شبکه با 100 عدد سوئیچ می‌تواند در بدترین حالت به ورود 10 میلیون جریان در ثانیه برسد. این اعداد نشان می‌دهند که سطح کنترل در معماری SDN، به دلیل ذات مرکزی آن، مستعد رنج بردن از مسئله مقیاس‌پذیری است [24].

* امنیت

حملات جاری در SDN بسیار پیچیده‌تر از قبل شده است. به طور کلی چالش‌های امنیتی SDN را می‌توان در سه دسته قرار داد: چالش‌های مرتبط با سوئیچ، چالش‌های مرتبط با کنترلر، و چالش‌های مرتبط با کانال ارتباطی [25]. چالش‌های امنیتی مرتبط به سوئیچ به طور عمده به آسیب‌پذیری‌های سیستم در سطح سوئیچ (سطح داده) که مهاجم[[54]](#footnote-54) با استفاده از آن‌ها ممکن است SDN را در معرض خطر قرار دهد، اطلاق می‌شود. سوئیچ OpenFlow و جدول جریان که حاوی اطلاعاتی درباره مدیریت شبکه، قوانین مسیریابی و ... هستند، احتمالاً اهداف اصلی در این سطح خواهند بود. منظور از چالش‌های امنیتی مرتبط با کنترلر، آسیب‌پذیری سیستم در سطح کنترلر است. در واقع خود کنترلر می‌تواند یک هدف اصلی برای انواع حمله‌ها از جمله حمله DoS[[55]](#footnote-55) باشد. یکی از چالش‌های مهم مرتبط با کانال ارتباطی این است که یک مکانیزم ارتباطی قابل اعتماد بین کنترلر و سوئیچ‌ها وجود ندارد.

* 1. واسط OpenFlow

تعامل بین سطح داده و سطح کنترلی در SDN، از طریق واسط جنوبی انجام می‌شود. در سال‌های اخیر واسط‌هایی مانند OpenFlow [26]، ForCES [27]، OVSDB [28] ، POF [29]، OpFlex [30]، OpenState [31]، برای این منظور ارائه شده‌اند. با این حال از آن‌جایی که OpenFlow به طور گسترده‌ای مورد قبول واقع شده و گسترش یافته است، در این بخش مطالبی درباره آن آورده‌ایم.

OpenFlow برای استانداردسازی ارتباط بین سوئیچ‌ها و کنترلرهای نرم‌افزاری در معماری SDN ارائه شده است. OpenFlow Specification یک پروتکل باز برای برنامه‌ریزی جدول جریان سوئیچ‌های مختلف را شرح می‌دهد. معماری OpenFlow شامل سه جزء اصلی است: یک سوئیچ OpenFlow، یک کانال ارتباطی امن، و یک کنترلر [32].

نسخه‌های مختلفی از Specification پروتکل OpenFlow در دسترس است. نسخه اول آن OpenFlow 0.2.0 بود، که در مارس 2008 منتشر شد. نسخه 1.0 که بسیار گسترش یافت، در دسامبر 2009 منتشر شده بود [32]. تغییرات اصلی در نسخه‌های مختلف OpenFlow در ‏جدول (2-1) آمده است [33].

* + 1. اجزاء سوئیچ OpenFlow

یک سوئیچ OpenFlow از یک یا چند جدول جریان، و یک جدول گروه[[56]](#footnote-56) که جستجوی بسته‌ها و پیشرانی را انجام می‌دهند، و یک یا چند کانال برای ارتباط با کنترلر خارجی تشکیل شده است (‏شکل (2-2) ).

Machine generated alternative text:
p
Protocol
_____ y
OpenFlow OpenFlow
Channel Channel :
Control Channel :
Controller
OpenFlow Switch
Figure 1: fiain components of an OpenFlow switch.

اجزاء اصلی سوئیچ OpenFlow

با استفاده از پروتکل OpenFlow، کنترلر می‌تواند مدخل‌های جریان را در جدول‌ها درج، بروزرسانی و یا حذف کند. هر جدول جریان شامل یک مجموعه از مدخل‌های جریان است، و هر مدخل شامل فیلدهای مطابقت ، شمارنده‌ها ، و مجموعه‌ای از دستورات می‌باشد.

عمل تطابق در اولین جدول جریان شروع می‌شود، و ممکن است در جدول‌های دیگر خط لوله[[57]](#footnote-57) ادامه یابد. اگر یک مدخل با بسته مطابقت داشته باشد، دستورات همراه آن مدخل، بر روی بسته اعمال می‌گردند. اگر هیچ تطابقی در جدول جریان پیدا نشود، خروجی به نحوه پیکربندی مدخل جریان پیش‌فرض، بستگی خواهد داشت. برای مثال ممکن است بسته از طریق OpenFlow به کنترلر ارسال شود، حذف گردد، و یا به جدول جریان بعدی برود.

تغییرات نسخه‌های اصلی OpenFlow

|  |  |  |  |
| --- | --- | --- | --- |
| نسخه | ویژگی‌های عمده | دلیل | مورد استفاده |
| 1.0 – 1.1 | چند جدول | جلوگیری از افزایش بیش از حد مدخل جریان |  |
| جدول گروه | فراهم کردن امکان اعمال مجموعه‌های عمل بر روی گروهی از جریان‌ها | توزیع بار، بازیابی خطا، تجمیع لینک |
| پشتیبانی کامل از VLAN، و MPLS |  |  |
| 1.1 – 1.2 | تطابق OXM | افزایش انعطاف‌پذیری تطابق |  |
| چند کنترلر | دسترسی بالا، توزیع بار، مقیاس‌پذیری | بازیابی خطا در کنترلر، توزیع بار در کنترلر |
| 1.2 – 1.3 | جدول Meter | افزودن قابلیت QoS، و DiffServ |  |
| مدخل عدم وجود در جدول | فراهم انعطاف‌پذیری |  |
| 1.3 – 1.4 | جدول همگام شده | بهبود مقیاس‌پذیری جدول | MAC Learning/Forwarding |
| Bundle | بهبود هماهنگ‌سازی سوئیچ | پیکربندی چند سوئیچ |
| 1.4 – 1.5 | جدول خروج | افزودن قابلیت انجام پردازش‌ها در پورت خروجی |  |
| زمان‌بند bundle | بهبود بیشتر هماهنگ‌سازی سوئیچ |  |

دستوراتی که در هر مدخل جریان وجود دارند، ممکن است شامل یک عمل[[58]](#footnote-58) و یا تغییر مسیر پردازشی در خط لوله باشند. عمل‌ها شامل دستوراتی هستند که نحوه ارسال یا تغییر بسته و یا نوع پردازش در جدول گروه را شرح می‌دهند. دستورات تغییر مسیر پردازشی در خط‌لوله، به بسته‌ها اجازه می‌دهند تا برای پردازش بیشتر به جدول‌های بعدی بروند. هم چنین امکان تبادل اطلاعات بین جدول‌ها از طریق فراداده[[59]](#footnote-59) وجود دارد.

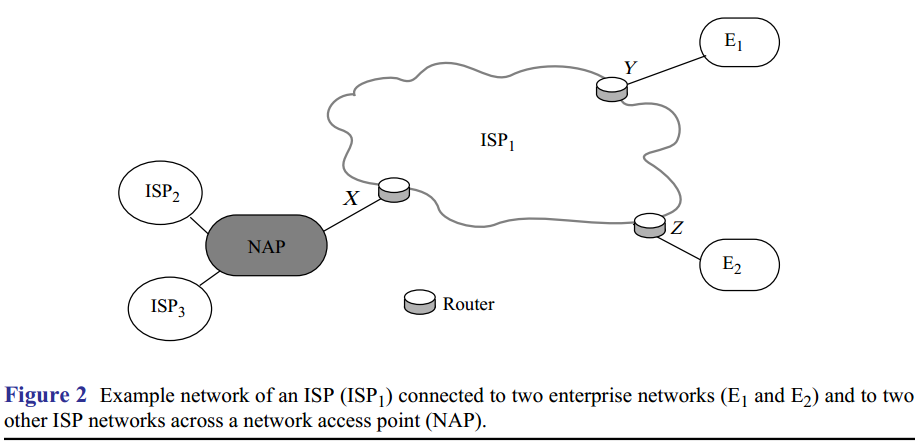
مدخل‌های جریان ممکن است یک بسته را به یک پورت ارسال کنند. این پورت معمولاً یک پورت فیزیکی است، اما ممکن است یک پورت منطقی[[60]](#footnote-60) یا یک پورت رزرو[[61]](#footnote-61) باشد. پورت‌های رزرو ممکن است مشخص کننده عمل‌هایی مانند ارسال به کنترلر، flooding، یا ارسال به صورت معمولی باشند. پورت منطقی توسط سوئیچ تعریف می‌شود و ممکن است مشخص کننده گروه‌های تجمیع شده از لینک‌ها[[62]](#footnote-62)، تونل‌ها[[63]](#footnote-63)، و یا واسط‌های loopback باشد.

عمل‌های موجود در مدخل‌های جریان ممکن است بسته را برای پردازش‌های بیشتر، به یک گروه ارسال کنند. گروه‌ها نمایانگر مجموعه‌هایی از عمل‌ها برای flooding، multipath، fast reroute، و ... هستند. جدول گروه شامل مدخل‌های گروه[[64]](#footnote-64) است، و هر مدخل گروه شامل یک لیست از عمل‌ها، که تعریف مشخصی برای هر نوع جدول دارند، می‌باشد.

طراحان سوئیچ می‌توانند معماری داخلی را به هر نحوی که راحت‌تر هستند، پیاده‌سازی کنند، فقط باید توجه داشت که معنی دستورات و عمل تطابق حفظ گردد. برای مثال، وقتی یک مدخل جریان در تمامی گروه‌ها برای ارسال به چند پورت استفاده می‌شود، طراح سوئیچ می‌تواند این را به صورت یک ماسک بیتی[[65]](#footnote-65) در جدول ارسال سخت‌افزاری پیاده‌سازی کند. مثال دیگر در این مورد عمل تطابق است، خط لوله‌ای که توسط OpenFlow ارائه شده، در عمل ممکن است توسط تعداد مختلفی جدول سخت‌افزاری پیاده‌سازی شود.

* 1. مسئله دسته‌بندی بسته‌ها

مسئله دسته‌بندی بسته در سال‌های اخیر مجدداً توجه زیادی را به خود جلب کرده است [6]. در مسیریاب اینترنت به فرآیند طبقه‌بندی بسته‌ها به جریان‌ها، دسته‌بندی بسته گفته می‌شود. تمام بسته‌هایی که به یک جریان تعلق دارند، از یک قانون از پیش تعیین شده پیروی کرده، و در مسیریاب به یک صورت پردازش می‌شوند. برای مثال ممکن است چنین تعریف شده باشد که تمام بسته‌هایی که آدرس IP مبدا و مقصد آن‌ها یکسان است، یک جریان باشند. به طور کلی عمل دسته‌بندی بسته روی چندین فیلد کار دشواری است. از این جهت محققین الگوریتم‌های متنوعی برای این کار ارائه کرده‌اند [34].



یک شبکه نمونه که در آن سرویس‌های مختلفی برای بقیه فراهم می‌کند.

سرویس‌های ارائه شده توسط برای مشتریان

|  |  |
| --- | --- |
| سرویس | مثال |
| پالایش بسته | نادیده گرفتن کل ترافیکی که از به می‌‎رود. |
| مسیریابی مبتنی بر سیاست | کل ترافیک VoIP که از به می‌رود، از طریق یک شبکه ATM جدا ارسال گردد. |
| حسابداری | دادن بالاترین اولویت به ترافیک ویدئو که به می‌رود، و انجام امور حسابرسی برای ترافیکی که به این صورت ارسال می‌شود. |
| محدودیت نرخ ترافیک | کسب اطمینان از این که در ترافیک رایانامه از 10 مگابیت بر ثانیه، و ترافیک کل از 50 مگابیت بر ثانیه بیشتر نشود. |
| شکل‌دهی ترافیک | کسب اطمینان از این که ترافیک وب وارد شده به از طریق واسط X از 50 مگابیت بر ثانیه بیشتر نشود. |

جریان‌ها توسط قوانینی که روی بسته‌های ورودی اعمال می‌شوند، مشخص می‌گردند. به مجموعه‌ای از قوانین، یک Classifier گفته می‌شود. هر قانون یک جریان را مشخص می‌کند، که بسته‌ها بر اساس برخی معیارها در فیلد‌های سرآیند خود، ممکن است متعلق به آن باشند. برای نشان دادن تنوع در Classifier ها، مثالی از نحوه استفاده یک ISP (که سرویس‌های مختلفی ارائه می‌کند) از دسته‌بندی بسته‌ها آورده‌ایم. ‏جدول (2-2) ISP1 را نشان می‌دهد که به سایت‌های مختلفی متصل شده است: شبکه‌های E1 و E2، و یک نقطه دسترسی شبکه[[66]](#footnote-66)، که خود آن به ISP2 و ISP3 وصل است. ISP1 سرویس‌های مختلفی را، طبق ‏جدول (2-2) برای مشتریان خود فراهم می‌کند [34].

‏جدول (2-3) جریان‌هایی را نشان می‌دهد، که بسته ورودی از پورت X باید در یکی از آن‌ها دسته‌بندی شود. توجه داشته باشید که جریان‌های مشخص شده ممکن است دو به دو ناسازگار[[67]](#footnote-67) باشند. برای مثال جریان اول و دوم در ‏جدول (2-3) با هم هم پوشانی دارند. این یک امر معمول است، و برای حل این مسئله معمولاً یک اولویت برای هر قانون در نظر گرفته می‌شود [34].

جریان‌های مربوط به بسته‌های ورودی از پورت X

|  |  |
| --- | --- |
| جریان | فیلدهای بسته مربوطه |
| ایمیل یا از | آدرس مبدا لایه لینک، شماره پورت مبدا لایه انتقال |
| از | آدرس مبدا لایه لینک |
| از و به | آدرس مبدا لایه لینک، آدرس مقصد لایه شبکه |
| تمامی بسته‌های دیگر | --- |

* + 1. تعریف رسمی مسئله دسته‌بندی بسته‌ها

فرض کنید هر قانون دارای d جزء[[68]](#footnote-68) باشد. جزء i ام قانون R، که به صورت R[i] نشان داده می‌شود، یک عبارت منظم[[69]](#footnote-69) روی فیلد i ام سرآیند آن بسته است (در عمل، این عبارت منظم به شکل‌های ساده address/mask یا operator/number(s) محدود می‌شود). در این صورت بسته P با قانون مشخص R تطابق می‌یابد، اگر برای هر i، i امین فیلد از سرآیند بسته P عبارت منظم R[i] را ارضا کند.

* + 1. انواع عمل تطابق

انواع عمل تطابق که ممکن است در فیلدهای مختلف صورت گیرد به صورت زیر است:

1. انطباق دقیق[[70]](#footnote-70): یا انطباق عددی، به معنی این است که دو مقدار دقیقاٌ برابر باشند. در دسته‌بندی بسته‌ها برای فیلدی مانند فیلد پروتکل، انطباق دقیق صورت می‌گیرد.
2. انطباق پیشوندی[[71]](#footnote-71): مانند انطباق در آدرس IP مبدأ یا مقصد. در این جا بررسی می‌شود که پیشوند قانون، پیشوندی از آدرس موجود در سرآیند بسته هست یا خیر.
3. انطباق بازه‌ای[[72]](#footnote-72): در این انطباق بررسی می‌شود که آیا عدد در بازه مشخص شده قرار می‌گیرد یا خیر (مانند فیلد پورت).
   * 1. SDN و چالش دسته‌بندی بسته‌ها

در بخش ‏2-1- گفته شد که یکی از مشخصه‌های اصلی SDN، جداسازی سطح کنترل و سطح داده است. از طرفی گفته شد که SDN از دانه‌بندی جریان استفاده کرده، و یک جریان را با استفاده از 15 فیلد از سرآیند بسته تعریف می‌کند. این عوامل باعث شده است تا طراحی دسته‌بندی‌کننده بسته[[73]](#footnote-73) با کارآیی بالا یک مسئله چالش برانگیز باشد [6].

* دسته‌بندی بسته یک عمل اصلی در سوئیچ‌های SDN

تفاوت اصلی بین دستگاه‌های سطح داده در شبکه‌های سنتی با شبکه‌های SDN در این است که در SDN این دستگاه‌ها عناصر ساده پیشرانی هستند که هیچ‌گونه کنترل نهفته یا نرم‌افزاری برای اتخاذ تصمیم خودمختار ندارند. بسته‌ها به محض ورود به این دستگاه‌ها با تعدادی قانون که در جدول قوانین قرار گرفته‌اند، مطابقت داده شده و در صورت تطبیق یافتن بسته با یک قانون عملی که همراه با آن قانون است، روی بسته اعمال می‌شود. بنابراین دسته‌بندی بسته به عنوان یکی از کارهای اصلی[[74]](#footnote-74) در مسیریاب‌های اینترنت نسل بعدی در نظر گرفته می‌شود [6].

* افزایش اندازه جدول جریان در SDN

در شبکه‌های سنتی غالباً از 5 فلید برای مشخص کردن جریان استفاده می‌کردند. این فیلدها عبارت بود از: پروتکل، شماره پورت مبدا و مقصد، و آدرس IP مبدا و مقصد. در شبکه‌های SDN و به طور خاص سوئیچ‌های OpenFlow تعداد این فیلد‌ها به 15 افزایش یافته است. ‏جدول (2-4) این 15 فیلد را نشان می‌دهد [6]. این امر دو تاثیر در اندازه جدول جریان می‌گذارد: اول این که اندازه مدخل جریان افزایش پیدا می‌کند (گسترش عرضی)، و دوم این که تعداد جریان‌ها در جدول جریان به طور بالقوه می‌تواند بسیار بیشتر از قبل باشد (گسترش عمقی).

فیلدهای سرآیند بسته که در OpenFlow پشتیبانی می‌شوند.

|  |  |  |  |
| --- | --- | --- | --- |
| نماد | فیلد سرآیند | تعداد بیت‌ها | نوع فیلد |
| IngPort | پورت ورودی | 32 | انطباق دقیق |
| Metadata | فراداده | 64 | انطباق دقیق |
| Eth\_src | آدرس مبدا اترنت | 48 | انطباق دقیق |
| Eth\_dst | آدرس مقصد اترنت | 48 | انطباق دقیق |
| Eth\_type | نوع اترنت | 16 | انطباق دقیق |
| VID | VLAN ID | 12 | انطباق دقیق |
| Vprty | اولویت VLAN | 3 | انطباق دقیق |
| MPLS\_lbl | برچسب MPLS | 20 | انطباق دقیق |
| MPLS\_tfc | کلاس ترافیک MPLS | 3 | انطباق دقیق |
| SA | آدرس مبدا IPv4 | 32 | انطباق پیشوندی |
| DA | آدرس مقصد IPv4 | 32 | انطباق پیشوندی |
| Prtl | پروتکل IPv4 | 8 | انطباق دقیق |
| ToS | نوع سرویس IPv4 | 6 | انطباق دقیق |
| SP | پورت مبدا | 16 | انطباق بازه‌ای |
| DP | پورت مقصد | 16 | انطباق بازه‌ای |

چالش‌هایی که مسئله دسته‌بندی بسته در شبکه‌های SDN با آن روبرو است، عبارت‌اند از [7]:

1. افزایش اندازه جدول جریان با گسترش عرض و عمق مجموعه قوانین دسته‌بندی.
2. افزایش پیچیدگی مجموعه قوانین.
3. افزایش درخواست برای گذردهی بالا و توان مصرفی کم.

بنابراین استفاده از حافظه TCAM در سوئیچ‌های سخت‌افزاری به دلیل افزایش اندازه جدول جریان که باعث افزایش بیش از حد توان مصرفی و فضای سخت‌افزاری می‌شود، هر چند زمان اجرای بسیار خوبی دارد ()، چندان به صرفه نخواهد بود. در سوئیچ‌های نرم‌افزاری این امر باعث افت کارآیی سلسله مراتب حافظه و در نتیجه کارآیی کلی سیستم می‌شود. لذا بازنگری در نحوه مدیریت جدول جریان در این سوئیچ‌ها و ارائه روش‌های جدید برای دسته‌بندی کارآمد بسته‌ها، یک امر ضروری است.

* 1. استفاده از پردازنده‌های گرافیکی برای دسته‌بندی بسته

ایده انجام محاسبات روی پردازنده‌های گرافیکی همه منظوره (GPGPU‌[[75]](#footnote-75))، به اندازه‌ای که ممکن است در نظر داشته باشید، جدید نیست. این پردازنده‌ها در کاربردهایی که به گذردهی بالا نیاز داشته و مسئله قابلیت موازی‌سازی داشته باشد، بسیار کارآمد هستند. در فصل چهارم این پایان‌نامه نمونه‌ای از روش پیشنهادی، جهت ارزیابی بر روی این پردازنده‌ها پیاده‌سازی شده است. بنابراین در این بخش ابتدا مطالبی درباره تاریخچه استفاده از این پردازنده‌ها برای کارهای غیر گرافیکی ذکر کرده‌ایم، سپس معماری این پردازنده‌ها را توضیح داده و در آخر به کاربرد‌های آن به ویژه در حوزه شبکه‌های کامپیوتری اشاره کرده‌ایم.

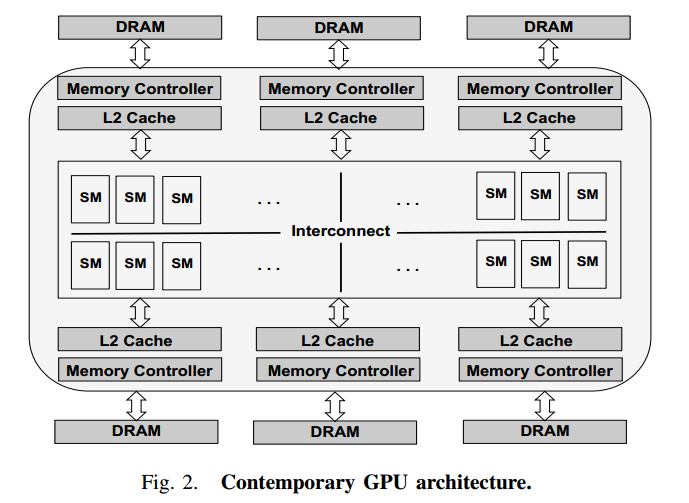
* + 1. انجام محاسبات همه‌منظوره روی پردازنده‌های گرافیکی

انتشار پردازنده‌های گرافیکی که خط لوله‌های قابل برنامه‌ریزی را ارائه می‌کردند، بسیاری از محققان را جهت بهره بردن از سخت‌افزار گرافیکی برای کارهایی جز کارهای ساده مبتنی بر OpenGL یا DirectX، مجذوب خود کرد. روش عمومی انجام محسبات روی GPU در روزهای اول بسیار سخت و پیچیده بود. چرا که واسط‌های برنامه‌نویسی استاندارد گرافیکی مانند OpenGL، و DirectX تنها راه تعامل با GPU بود، و هر تلاشی برای انجام محاسبات دلخواه روی GPU در محدودیت برنامه‌نویسی با یک API گرافیکی بود. به همین جهت، محققین برای انجام محاسبات همه منظوره از طریق واسط‌های برنامه‌نویسی گرافیکی، سعی می‌کردند مسائل خود را به گونه‌ای به شکل مسائل گرافیکی در بیاورند. انجام این کار هر چند بسیار هوشمندانه، ولی بسیار پیچیده بود [35].

در نوامبر 2006، شرکت NVIDIA از اولین پردازنده گرافیکی DirectX 10، که GeForce 8800 GTX بود، پرده‌برداری کرد. این پردازنده هم‌چنین اولین پردازنده‌ای بود که با معماری CUDA[[76]](#footnote-76) ساخته شده بود. این معماری شامل چندین جزء جدید بود که مخصوصاً برای انجام محاسبات روی GPU طراحی شده بودند، و بسیاری از محدودیت‌هایی که مانع از انجام محاسبات همه منظوره روی آن می‌شد را حذف کرده بود. NVIDIA برای آن که حداکثر توسعه‌دهندگان ممکن را جذب کند، تعداد به نسبت کمی keyword به زبان استاندارد C اضافه کرد، تا قابلیت استفاده از ویژگی‌های خاص CUDA را فراهم سازد. کمی بعد از راه‌اندازی GeForce 8800 GTX، NVIDIA یک کامپایلر به نام CUDA C، برای این زبان ارائه کرد. و به این ترتیب CUDA C اولین زبانی شد که به طور خاص توسط یک شرکت سازنده GPU، برای فراهم کردن انجام محاسبات همه منظوره روی پردازنده‌های گرافیکی طراحی گردید [35].

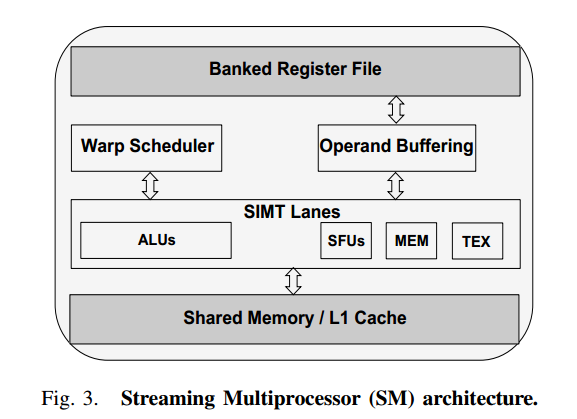
* + 1. معماری پردازنده گرافیکی

‏شکل (2-4) یک نمودار بلوکی از پردازنده‌های گرافیکی همه منظوره NVIDIA را نشان می‌دهد. GPU‌[[77]](#footnote-77) از تعدادی چندپردازنده جریانی[[78]](#footnote-78) (SM) ، شش کانال DRAM با پهنای باند بالا، و حافظه کش L2 درون‌تراشه[[79]](#footnote-79) تشکیل شده است. تعداد چندپردازنده‌ها و تعداد هسته‌ها در هر چندپردازنده با توجه به قیمت و بازار هدف GPU متفاوت است. ‏شکل (2-5) ساختار یک SM را نشان می‌دهد. یک SM از 32 خط[[80]](#footnote-80) SIMT‌[[81]](#footnote-81) تشکیل شده، که می‌تواند در هر چرخه[[82]](#footnote-82) به هر ریسمان[[83]](#footnote-83)، 1 دستور صادر کند، که در مجموع 32 دستور در هر چرخه در هر SM خواهد شد. ریسمان‌ها داخل گروه‌های 32 تایی سازماندهی می‌شوند، که به هر کدام یک "Wrap" گفته می‌شود. زمان‌بندی[[84]](#footnote-84) با دانه‌بندی[[85]](#footnote-85) wrap صورت می‌گیرد، و تمامی ریسمان‌های موجود در یک wrap از یک شمارنده برنامه[[86]](#footnote-86) استفاده می‌کنند. همان‌طور که در ‏شکل (2-5) دیده می‌شود، خطوط SIMT به یک register file سریع و کش L1 درون تراشه دسترسی دارند. داشتن register file، پهنای باند کافی برای پشتیبانی از دو ورودی و یک خروجی را به ازای هر چرخه فراهم می‌آورد [36].

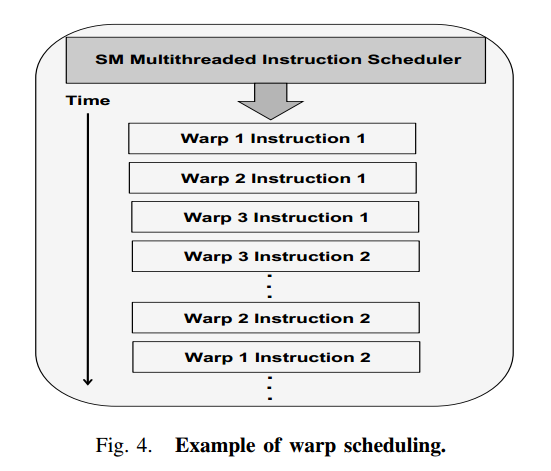


معماری پردازنده‌های گرافیکی امروزی

پردازنده‌های گرافیکی برای پر کردن واحدهای محاسباتی به multithreading سخت‌افزاری وابسته هستند. آن‌ها مخزن‌های[[87]](#footnote-87) بزرگی از ریسمان‌های فعال را که در قالب wrap سازماندهی شده‌اند، نگهداری می‌کنند. برای مثال، NVIDIA Fermi از 48 wrap فعال و به طور کلی 1536 ریسمان در هر SM پشتیبانی می‌کند. Fermi در هر SM یک register file با اندازه 128 کیلوبایت، یا به عبارت دیگر 21 رجیستر 32 بیتی در هر ریسمان دارد. هر ریسمان از رجیسترهای اختصاصی جهت راه‌گزینی[[88]](#footnote-88) سریع استفاده می‌کند. زمان‌‌بندی ریسمان با دانه‌بندی wrap صورت می‌گیرد. ‏شکل (2-6) یک مثال از زمان‌بندی wrap را نشان می‌دهد. در هر چرخه، زمان‌بند یک wrap را که آماده اجراست انتخاب، و دستور بعدی ریسمان‌های فعال از آن wrap را صادر می‌کند. در انتخاب wrap فاکتورهایی مانند نوع دستورات، و عدالت در نظر گرفته می‌شوند. پردازش دستورات در داخل یک wrap به ترتیب انجام می‌شود، در حالی که انتخاب wrap ممکن است به صورت خارج از نوبت[[89]](#footnote-89) صورت گیرد [36].



معماری چندپردازنده جریانی



مثالی از زمان‌بندی wrap

یک پردازنده SIMT در صورتی که خطوط کاملاً پر شده باشند، به طور کامل کارآمد[[90]](#footnote-90) خواهد بود. این زمانی رخ می‌دهد که تمامی 32 ریسمان یک wrap یک مسیر اجرایی را اتخاذ کنند. اگر ریسمان‌های یک wrap به دلیل کنترل جریان واگرا باشند، مسیرهای متفاوت اجرایی، به صورت سریال اجرا می‌شوند. ریسمان‌هایی که در مسیر اجرایی نباشند، غیرفعال، و در پایان تمامی ریسمان‌ها به مسیر اجرایی اصلی همگرا می‌گردند. SM از پشته همگام‌سازی شاخه[[91]](#footnote-91) برای مدیریت واگرایی و همگرایی ریسمان استفاده می‌کند [36].

* + 1. برنامه‌های کاربردی CUDA

از زمان شروع به کار CUDA C در اوایل 2007، صنایع و برنامه‌های کاربردی زیادی از تعامل موفق با آن لذت برده‌اند. بهره این تعامل غالباً شامل افزایش چندین مرتبه کارآیی نسبت به بهترین پیاده‌سازی‌های موجود بود [35]. به عنوان مثال از این تکنولوژی در تصویربرداری پزشکی[[92]](#footnote-92) [35]، علوم محیط زیست[[93]](#footnote-93) [35]، پیش‌بینی آب و هوا [37]، شبیه‌سازی دینامیک مولکولی [37]، و ... استفاده شده است.

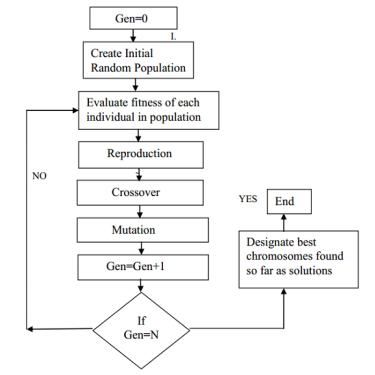
در سال‌های اخیر، در حوزه شبکه‌های کامپیوتری از پردازنده‌های گرافیکی برای کارهای مختلف پردازش بسته استفاده شده است. از آن جا که عمل اصلی در مسیریاب‌ها، تشخیص پورت خروجی برای ارسال یک بسته بر اساس آدرس مقصد آن است، می‌توان از پردازنده‌های گرافیکی که دارای صدها هسته پردازشی هستند، برای انجام عمل جستجو روی صد‌ها بسته به طور موازی استفاده کرد [38]. در این جا به برخی از پروژه‌های قابل توجه در این زمینه اشاره می‌کنیم.

1. PacketShader [39]: در این پروژه پیشرانی پروتکل‌های IPv4 و IPv6، رمزنگاری IPsec، و عمل مطابقت جریان در پروتکل OpenFlow با استفاده از معماری CUDA پیاده‌سازی شده است. این پروژه اولین پروژه‌ای بود که امکان داشتن مسیریاب نرم‌افزاری با چندین پورت 10 Gbps را نشان داد.
2. Gnort [40]: در این پروژه سیستم تشخیص نفوذ Snort به روی سکوی GPU انتقال داده شده است (با استفاده از معماری CUDA). همکاری اصلی این پروژه پیاده‌سازی تطبیق-الگوی رشته[[94]](#footnote-94) سریع روی GPU می‌باشد.
3. Hermes [41]: این پروژه که روی PacketShader ساخته شده، یک مسیریاب CPU/GPU پیاده‌سازی کرده است که به صورت پویا اندازه دسته[[95]](#footnote-95) را برای بهینه‌سازی معیارهای QoS (پهنای باند، و تاخیر) تنظیم می‌کند.
   1. الگوریتم ژنتیک

الگوریتم‌های ژنتیک[[96]](#footnote-96)، الگوریتم‌هایی مبتنی بر اصول تکامل طبیعی، برای مسائل جستجو و بهینه‌سازی هستند، که ابتدا در سال 1970 توسط John Holland معرفی گردید. الگوریتم ژنتیک به طور کلی شامل دو فرآیند است: فرآیند اول انتخاب افراد، و فرآیند دوم تغییر افراد انتخاب شده برای ایجاد نسل بعدی است، و این کار از طریق تکنیک‌های تقاطع[[97]](#footnote-97) و جهش[[98]](#footnote-98) صورت می‌گیرد. مکانیزم انتخاب، تعیین می‌کند که کدام افراد برای بازتولید انتخاب شوند، و هر فرد منتخب چندین فرزند تولید کند. اصل کلیدی در مکانیزم انتخاب این است که افراد بهتر، شانس بیشتری برای انتخاب شدن داشته باشند [42].

‏شکل (2-8) نمودار گردش کار الگوریتم ژنتیک را نشان می‌دهد. گام‌های الگوریتم ژنتیک به صورت زیر می‌باشند [42]:

1. **مقداردهی اولیه:** الگوریتم ژنتیک عموماً با یک جمعیت اولیه که به صورت تصادفی تولید شده‌ است آغاز می‌گردد. البته برخی از محققین تکنیک‌های خاصی برای تولید جمعیت اولیه‌ی با کیفیت‌تر ارائه کرده‌اند. چنین روش‌هایی برای تسریع فرآیند تکامل طراحی می‌شوند.
2. **انتخاب:** در این گام دو کروموزوم والد از جمعیت بر اساس برازش[[99]](#footnote-99) بهتر انتخاب می‌شوند. برای انجام این کار ابتدا یک امتیاز بر اساس تابع برازش[[100]](#footnote-100)، به هر فرد اختصاص داده شده و بر اساس آن امتیاز عمل انتخاب صورت می‌گیرد.
3. **بازتولید:** برای بازتولید جمعیت، عمل تقاطع روی دو کروموزومی که در گام قبل انتخاب شدند، اعمال شده و یک یا دو فرزند تولید می‌شوند. هم‌چنین ممکن است عمل جهش هم روی آن ها انجام شود. جهش یک عملگر عمومی است که برای حفظ تنوع در جمعیت نسل بعدی کروموزوم‌ها مورد استفاده قرار می‌گیرد.
4. **جایگذاری:** از جمعیت تولید شده جدید برای اجرای بعدی الگوریتم استفاده می‌شود.



نمودار گردش کار الگوریتم ژنتیک

برای خاتمه دادن به الگوریتم شرایط مختلفی ممکن است در نظر گرفته شود، برای مثال ممکن است شرط خاتمه تکرار الگوریتم به تعداد مشخصی باشد، یا الگوریتم تا زمانی ادامه پیدا کند که در کیفیت جمعیت‌های نسل بعدی بهبودی حاصل نشود. هم چنین شرط خاتمه ممکن است این باشد که کیفیت جمعیت به یک حد نصاب مشخص برسد. در فصل چهارم این پایان‌نامه از این الگوریتم برای یافتن بیت‌های موثر برای افراز یک مجموعه قوانین استفاده کرده‌ایم.

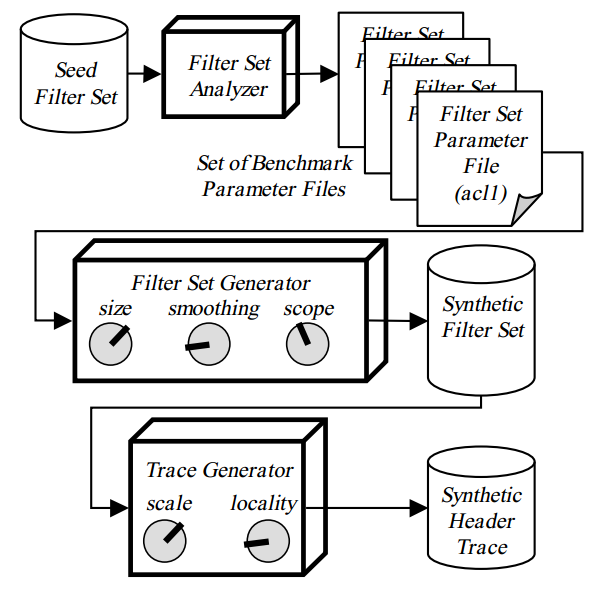
* 1. تولید مجموعه داده

ClassBench [19] مجموعه‌ای از ابزارهای محک‌زنی[[101]](#footnote-101) برای دستگاه‌ها و الگوریتم‌های دسته‌بندی بسته‌ است، که در دسترس عموم قرار داده شده است. این ابزار مجموعه قوانین را برای 5 فیلد آدرس IP مبدأ و مقصد، شماره پورت مبدأ و مقصد، و نوع پروتکل تولید می‌کند. از آن جا که در SDN مجموعه قوانین شامل 15 فیلد است، ابتدا 5 فیلد را با استفاده از ClassBench تولید کرده، و سپس 10 فیلد دیگر را با استفاده از مقادیر یکتا[[102]](#footnote-102) به صورتی که در [6] گفته شده، و با نسبت‌های Wildcard‌[[103]](#footnote-103) 0، 0.1، 1، و 5 تولید، و به مجموعه قوانین ClassBench اضافه کرده‌ایم. در ادامه ابتدا توضیحاتی راجع به ClassBench آورده و سپس نحوه افزودن 10 فیلد دیگر را شرح داده‌ایم.

* معماری ClassBench

در ClassBench ابتدا یک بررسی روی 12 مجموعه قوانین واقعی انجام شده است. این مجموعه قوانین شامل 11 تا 4557 قانون بوده و در یکی از سه قالب موجود برای مجموعه قوانین قرار می‌گرفتند: ACL‌[[104]](#footnote-104)، FW‌[[105]](#footnote-105)، و IPC‌[[106]](#footnote-106). ابزار ClassBench به این صورت عمل می‌کند که ابتدا یک سری فایل پارامتر که خصوصیات مرتبط با مجموعه قوانین واقعی در آن‌ها توصیف می‌شود، را می‌سازد. سپس با استفاده از یک فایل پارامتر، و یک سری ورودی‌های سطح بالای دیگر، یک مجموعه قوانین تولید می‌کند. علاوه بر آن، دنباله‌ای از سرآیند بسته‌ها تولید می‌کند تا با کمک آن‌ها بتوان روش مورد نظر برای دسته‌بندی را محک زد. بنابراین همان طور که در ‏شکل (2-8) نشان داده شده است این ابزار دارای سه جزء اصلی است:

1. Filter Set Analyzer: فایل‌های پارامتر شامل توزیع‌های آماری و احتمالی برای کمک به ایجاد مجموعه قوانین واقعی هستند. در این بخش توزیع‌های آماری و احتمالی از یک مجموعه قوانین اولیه استخراج می‌شود، و یک فایل پارامتر توسط آن تولید می‌گردد. این کار کمک می‌کند تا با استفاده از ساختار یک مجموعه قوانین اولیه، مجموعه قوانین واقعی بزرگ‌تر تولید شود.
2. Filter Set Generator: این بخش یک فایل پارامتر و تعدادی پارامتر سطح بالای دیگر را به عنوان ورودی دریافت کرده، و یک مجموعه قوانین تولید می‌کند. پارامترهای سطح بالا شامل اندازه مجموعه قوانین خروجی، پارامتر هموارسازی[[107]](#footnote-107)، پارامتر گستره[[108]](#footnote-108) پورت و گستره آدرس هستند.
3. Trace Generator: برای ارزیابی و اندازه‌گیری کارایی روش دسته‌بندی، نیاز داریم تا روش را با استفاده از سرآیند بسته‌ها تست کنیم. می‌خواهیم مطمئن شویم که هر سرآیند توسط حداقل یک قانون از مجموعه قوانین پوشش داده می‌شود. با وجود این که نمونه‌های زیادی از سرآیند بسته‌ها وجود دارند، ولی در هیچ کدام وابستگی آنها به مجموعه قوانین در نظر گرفته نشده است. در این جا Trace Generator مجموعه قوانین را بررسی کرده و سپس دنباله‌ای از سرآیند بسته‌ها را تولید می‌کند.



نمودار بلوکی مجموعه ابزار ClassBench

* افزودن 10 فیلد دیگر

در یک مجموعه قوانین، مقادیر مختلفی که قوانین در یک فیلد اتخاذ کرده‌اند را به عنوان مقادیر یکتا در آن فیلد خاص تعریف می‌کنیم. برای مثال در فیلد ToS مقادیر 0، 1، 2، و \*، و یا در فیلد EtherType مقدار 0x0800 مقادیر یکتایی هستند. تعداد مقادیر یکتا در یک فیلد نسبت به تعداد کل قوانین در آن مجموعه معمولاً مقدار بزرگی نیست [43]. در ‏جدول (2-5) نسبت تعداد مقادیر یکتا به تعداد کل قوانین در مجموعه را آورده‌ایم [6]. در این جا با استفاده از مقادیر ذکر شده در این جدول، و با نسبت Wildcard 0، 0.1، 1، و 5، 10 فیلد دیگر را به مجموعه قوانین ClassBench افزوده و از آن برای ارزیابی روش پیشنهادی در فصل چهارم استفاده کرده‌ایم. نسبت Wildcard برای یک فیلد در مجموعه قوانین برابر با تعداد مقادیر \* در آن فیلد تقسیم بر تعداد کل قوانین است.

نسبت تعداد مقادیر یکتا به تعداد کل قوانین در مجموعه قوانین

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| IngPort | Metadata | Eth\_src | Eth\_dst | Eth\_type | VID | Vprty | ToS |
| 0.02 | 0.02 | 0.25 | 0.25 | 0.01 | 0.2 | 1 | 1 |
| MPLS\_lbl | MPLS\_tfc | SA | DA | Ptrl | SP | DP |  |
| 0.02 | 1 | 0.25 | 0.25 | 0.02 | 0.1 | 0.1 |  |

* 1. جمع‌بندی

در این فصل شبکه‌های نرم‌افزار محور و معماری آن‌ها به طور مختصر بیان شد. سپس معماری سوئیچ OpenFlow، به عنوان یک واسط جنوبی استاندارد در معماری SDN که بسیار مورد پذیرش قرار گرفته مرور شد. سپس، مسئله دسته‌بندی بسته‌ها مطرح شده و نشان داده شد که SDN چگونه این مسئله را، مجدداً به یک موضوع چالشی تبدیل کرده است. اولاً، به دلیل جداسازی سطح کنترل از سطح داده در سوئیچ SDN، این دستگاه‌ها به عناصر ساده پیشرانی تبدیل شده و مسئله دسته‌بندی بسته یکی از کارهای اصلی در آن‌ها می‌باشد، به طوری که سرعت دسته‌بندی رابطه مستقیمی با کارآیی سوئیچ دارد. ثانیاً، تعریف عام SDN از جریان و استفاده از تعداد زیادی فیلد برای متمایز کردن جریان‌ها، از دو جهت مسئله دسته‌بندی بسته را تحت تاثیر قرار داده است: گسترش عرضی و گسترش عمقی جدول جریان.

در ادامه فصل معماری پردازنده‌های گرافیکی همه منظوره مرور و کاربرد آن‌ها به خصوص در حوزه شبکه‌های کامپیوتری ارائه شد. این پردازنده‌ها به خاطر ماهیت موازی خود، و در نتیجه گذردهی بالایی که فراهم می‌کنند، در سال‌های اخیر به شدت مورد استقبال محققین این حوزه قرار گرفته و کارهای ارزنده‌ای با استفاده از آن‌ها انجام شده است. در انتهای این فصل، الگوریتم ژنتیک معرفی شد. زیرا در روش پیشنهادی در فصل چهارم، برای یافتن بیت‌های موثر جهت افراز مجموعه قوانین از این الگوریتم استفاده شده است.

1. فصل3: مروری بر کارهای قبلی

در بخش ‏2-4- بیان شد که SDN چگونه مسئله دسته‌بندی بسته‌ها را دوباره به یک مسئله چالشی تبدیل کرده است. در این فصل برخی از پژوهش‌هایی که در این زمینه صورت گرفته مرور شده است. راه‌حل‌های مطرح شده را بر اساس معیارهای مختلفی می‌توان دسته‌بندی نمود. در این بخش، کارهای انجام شده بر اساس 2 معیار تقسیم‌بندی شده‌اند: استفاده یا عدم استفاده از ویژگی‌های SDN؛ سخت‌افزاری یا نرم‌افزاری بودن راه‌حل.

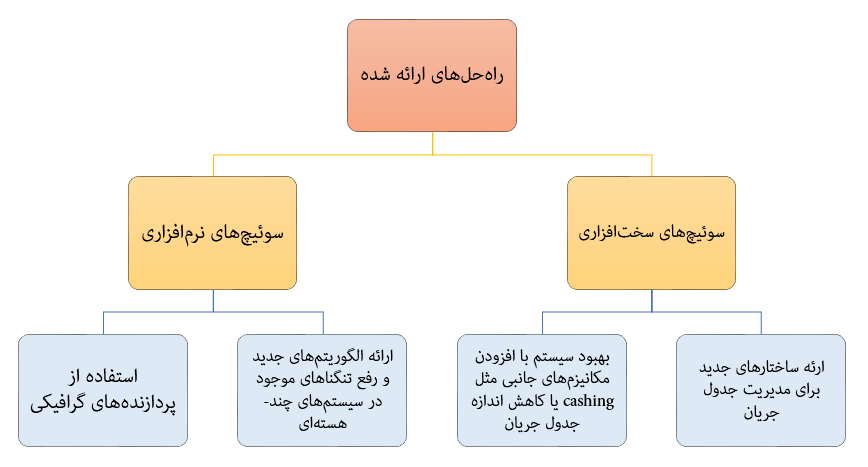
* استفاده یا عدم استفاده از ویژگی‌های SDN

راه‌حل‌های ارائه شده را می‌توان از آن جهت که از ویژگی‌های خاص شبکه‌های نرم‌افزار محور استفاده نموده‌اند یا خیر، به دو دسته تقسیم‌بندی نمود. در مراجع [8]–[10] سعی شده از ویژگی‌های انعطاف‌پذیری، قابلیت برنامه‌ریزی و وجود کنترلر مرکزی، برای حل مسئله دسته‌بندی بسته‌ها بهره‌برداری شود. در دسته دوم (مانند مراجع [11]–[14], [44]) بیشتر تلاش بر این است تا با استفاده از روش‌های الگوریتمی و یا ارائه ساختمان داده‎های جدید، کارآیی را بهبود بخشند.

* سخت‌افزاری یا نرم‌افزاری بودن راه‌حل

معیار دیگری که برای تقسیم‌بندی راه‌حل‌های ارائه شده در گذشته می‌توان استفاده کرد، سخت‌افزاری یا نرم‌افزاری بودن آن‌ها است. این تقسیم‌بندی در ‏شکل (3-1) نشان داده شده است.

در راه‌حل‌های سخت‌افزاری، هم‌چنان از حافظه TCAM استفاده کرده و با استفاده از تکنیک‌هایی همچون کاهش اندازه جدول جریان، سعی کرده‌اند سربار سخت‌افزاری و مصرف توان آن را کاهش دهند [11]. برخی نیز به جای استفاده از TCAM، ساختار جدیدی برای جدول جریان پیشنهاد داده‌اند، به طوری که معایب TCAM را نداشته باشد [44].



طبقه بندی روش‌های دسته‌بندی بسته بر اساس سخت‌افزاری یا نرم‌افزاری بودن

در گذشته، از آن جا که مسیریاب‌های نرم‌افزاری نمی‌توانستند به نرخ ارسال دستگاه‌های سخت‌افزاری برسند، تولیدکنندگان مسیریاب از راه‌حل‌های سخت‌افزاری استفاده می‌کردند. اما در سال‎های اخیر مسیریاب‌های نرم‌افزاری توجه دنیای شبکه را به دو دلیل به خود جلب کرده‌اند [38]: اول این که سخت‌افزارهای موجود به طور قابل توجهی پیشرفت کرده، و امکان این که بازدهی مسیریاب‌های نرم‌افزاری به نرخ خط[[109]](#footnote-109) برسد فراهم شده است. مورد دیگر این که محققان به طور روزافزونی توابع پردازش بسته پیچیده‌تری را معرفی می‌کنند.

در کنار توجه به مسیریاب‌های نرم‌افزاری، سخت‌افزارهایی مناسب برای پردازش نرم‌افزاری بسته‌ها کشف شده‌اند. از آن جا که پردازش بسته به طور ذاتی یک کاربرد SIMD‌[[110]](#footnote-110) است، پردازنده‌های گرافیکی یک نامزد مناسب برای مسیریاب‌های نرم‌افزاری می‌باشند [38]. راه‌حل‌های ارائه شده در مسیریاب‌های نرم‌افزاری را از نظر سکویی که بر روی آن اجرا می‌شوند می‌توان به دو دسته تقسیم‌بندی نمود: آن‌هایی که روی پردازنده‌های چند هسته‌ای اجرا می‌شود، و آن‌هایی که بر روی پردازنده‌های گرافیکی انتقال داده شده‎‌اند.

در ادامه، ابتدا مروری بر معیارهای سنجش راه‌حل‌های دسته‌بندی بسته‌ها می‌شود، سپس برخی از کارهای انجام شده در این زمینه به طور مختصر شرح داده می‌شود. در این بخش، به دلیل ارتباط با روش پیشنهادی، به الگوریتم‌های نرم‌افزاری توجه بیشتری شده است.

* 1. معیارهای سنجش الگوریتم‌های دسته‌بندی
* سرعت جستجو

با افزایش سرعت لینک‌ها، دسته‌بندی باید با سرعت بیشتری انجام شود [34].

* حافظه مصرفی

یکی دیگر از فاکتورهای ارزیابی هر روش دسته‌بندی، میزان حافظه‌ای است که مصرف می‌کند. هر چند امروزه بهای حافظه کاهش یافته است اما این پارامتر همچنان از معیارهای مهم اندازه‌گیری در هر یک از الگوریتم‌های جستجو و اندیس‌گذاری می‌باشد. به خصوص، مرتبه افزایش حافظه دارای اهمیت بسیاری است و روش‌هایی با مرتبه حافظه غیر خطی (توان 2 یا نمایی) با افزایش تعداد عناصر، میزان حافظه مصرفی آنها به سرعت رشد می‌کند. به علاوه، نیازمندی‌های حافظه اگر کم باشد، استفاده از تکنولوژی سریع و البته گران حافظه‌های SRAM ممکن می‌شود [45].

* زمان پیش‌پردازش

زمان پیش‌پردازش زمانی است که برای ایجاد ساختار داده‌های اولیه الگوریتم استفاده می‌شود. الگوریتمی که از بروزرسانی‌های افزایشی پشتیبانی کند را پویا می‌گویند. یک الگوریتم ایستا با هر بار اضافه یا کم شدن هر قانون نیاز دارد که کل ساختارش را مجدداً از اول بسازد. به طور کلی، الگوریتم‌های پویا نسبت به الگوریتم‌های ایستا تحمل زمان پیش پردازش بیشتری را دارند [45].

* بروزرسانی سریع

هنگامی که Classifier تغییر می‌کند، ساختار داده باید بروزرسانی شود. نرخ بروزرسانی در کاربردهای مختلف متفاوت است. برای مثال در دیواره‌های آتش که مدخل‌ها به صورت دستی یا به ندرت اضافه می‌شوند، نرخ بروزرسانی پایین است اما یک مسیریاب ممکن است به بروزرسانی با فرکانس خیلی بالا نیاز داشته باشد [34].

* مقیاس پذیری

تعداد فیلدهایی از سرآیند بسته که برای دسته‌بندی استفاده می‌شوند، یکی دیگر از فاکتورهای ارزیابی است [34]. با افزایش تعداد برنامه‌های شبکه، فیلدهای استفاده شده در تعریف قوانین افزایش یافته است و این فیلدها می‌توانند از سرآیند هر یک از لایه‌های اتصال داده، شبکه، انتقال و حتی در بعضی موارد لایه کاربرد باشند. روش پیشنهادی باید قابلیت گسترش به هر تعداد فیلد را داشته باشد و در صورت نیاز با افزایش تعداد فیلدها، سرعت آن کاهش پیدا نکند [45].

* توانایی پشتیبانی از جدول‌های قانون بزرگ

مسئله دسته‌بندی بسته‌ها امروزه کاربردهای زیادی پیدا کرده است و کاربران این فرآیند نیز وسیع‌تر شده‌اند، بنابراین نیازمندی‌ها افزایش یافته و این امر منجر به افزایش اندازه جدول قانون شده است. الگوریتم دسته‌بندی باید برای جداول قانون بزرگ‌تر نیز از نظر سرعت و میزان حافظه منعطف و قابل گسترش باشد [45].

* انعطاف پذیری

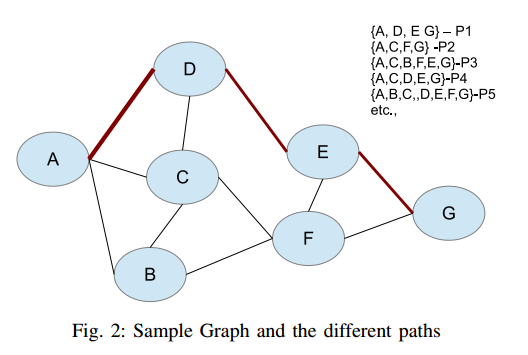
هر الگوریتم دسته‌بندی باید در نوع تعریف قانون منعطف باشد. یعنی برای هر نوع انطباق از جمله پیشوندی، عملگری (بزرگتر، کوچکتر، مساوی و غیره) و محدوده‌ای کارآمد باشد. همچنین بعضی از قوانین روی یک یا چند فیلد محدودیتی ندارند. به منظور پشتیبانی از چنین قوانینی باید از \* که به معنی همه چیز است در تمامی فیلدها پشتیبانی شود [45].

* 1. راه‌حل‌های استفاده کننده از ویژگی‌های SDN

در این بخش برخی از پژوهش‌هایی که از ویژگی‌های SDN برای حل مسئله بسته‌بندی استفاده کرده‎اند، مرور شده است.

* + 1. برچسب گذاری دو لایه‌ای

در مرجع [8]، ایده‌ای مشابه با ایده به کار رفته در پروتکل MPLS‌[[111]](#footnote-111) ارائه و سعی شده اندازه مدخل‌های جدول جریان و در نتیجه اندازه حافظه TCAM کاهش داده شود. این امر منجر به کاهش توان و فضای مصرفی می‎گردد. به این منظور از برچسب گذاری دو لایه‌ای برای کاهش تعداد بیت‌هایی که جریان را در سوئیچ توصیف می‌کنند، استفاده شده است. در این مکانیزم برچسب-در-برچسب[[112]](#footnote-112)، یک برچسب به هر مسیر داده می‌شود که به آن برچسب مسیر (PT[[113]](#footnote-113)) گفته می‌شود و برای مسیریابی بسته‌ها از آن استفاده می‌گردد. برچسب دیگری با نام برچسب جریان (FT‌[[114]](#footnote-114)) برای پیوند دادن بسته‌ها به یک جریان مورد استفاده قرار می‌گیرد. برچسب مسیر، از شباهت مسیر بین بسته‌های مختلف بهره می‌برد، و به عنوان برچسب مسیریابی، مشخص کننده عملی است که برای مسیریابی بسته‌ها در سوئیچ استفاده می‌شود. برچسب جریان برای تشخیص جریان توسط سوئیچ استفاده می‌شود تا مدیریت منبع بهتری فراهم کند.

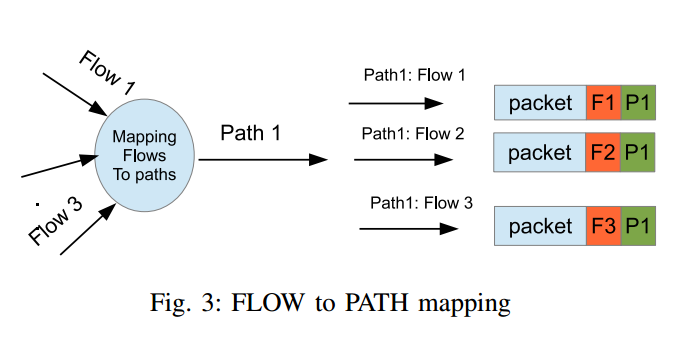


یک گراف نمونه و مسیرهای مختلف

این روش مبتنی بر رفتار معمول مشاهده شده در شبکه‌ها است، که در آن‌ها: 1- یک جریان یک مسیر اتخاذ می‌کند، 2- مسیرها مجموعه‌های قطعی[[115]](#footnote-115) هستند، و 3- چند جریان می‌توانند یک مسیر اتخاذ کنند. بنابراین برای مسیریابی بسته‌ها نیاز است تا مسیر برای یک جریان مشخص شود و این مسیر به تمامی بسته‌های آن جریان داده شود. این وابستگی بین جریان و مسیر از طریق برچسب مسیر مشخص می‌گردد. یک مثال ساده از توصیف‌گر مسیر در ‏شکل (3-2) نشان داده شده است.

با استفاده از PT می‌توان چندین جریان را به یک مسیر نگاشت کرد. در این صورت جدول جریان شامل مدخل‌هایی است که PT را دارند، و به همراه هر PT یک عمل وجود دارد. بنابراین در سوئیچ‌های هسته[[116]](#footnote-116) اندازه مدخل جریان به اندازه PT و عمل همراه با آن کاهش می‌یابد.

هنگامی که بسته‌ها به سوئیچ مقصد می‌رسند، باید به پورت صحیح که ماشین مقصد به آن متصل است ارسال شوند. سوئیچ‌های لبه باید یک جدول جستجوی مجزا که شامل (MAC + VLAN) است، برای مشخص کردن عملی که باید روی بسته انجام شود، نگه دارند. برای این کار می‌توان از یک FT که به صورت یکتا جریان بسته‌ها را مشخص می‌کند استفاده کرد. FT یک شناسه عددی است. نگاشت جریان‌ها بین PT و FT در ‏شکل (3-3) نشان داده شده است.



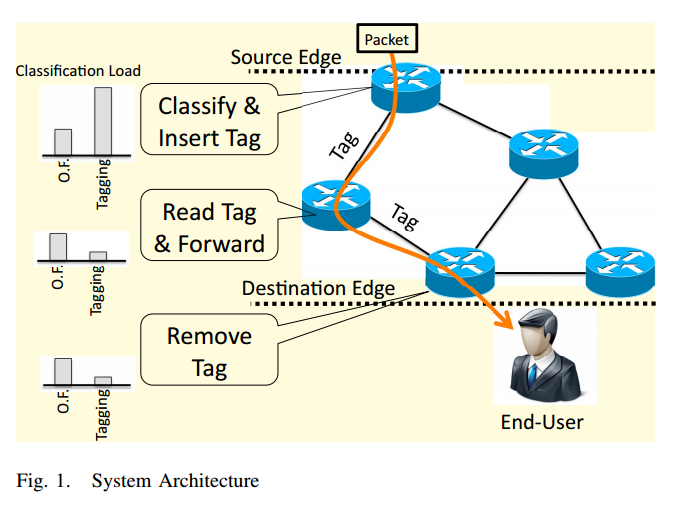
نگاشت جریان به مسیر

به منظور تحقق این معماری، PT و FT باید تولید شوند، سرآیند بسته‌ها باید این اطلاعات را در خود جای داده و جدول‌های جریان باید با PT و FT پر شوند. قابلیت برنامه‌نویسی SDN می‌تواند برای بهره بردن از این معماری به کار گرفته شود. در واقع PT، و FT توسط کنترلر تولید شده و به سوئیچ‌ها داده می‌شوند.

با استفاده از این معماری تعداد مدخل‌هایی که در یک TCAM با اندازه ثابت می‎توان جای داد، 15 برابر بیشتر شده، و توان مصرفی در سوئیچ‌های هسته حدود 80% کاهش یافته است.

* + 1. برچسب گذاری در سوئیچ لبه

فرهادی و همکاران در مرجع [9] کاری مشابه با [8] انجام داده‌اند؛ با این تفاوت که در این جا تنها از یک برچسب استفاده شده است. معماری مطرح شده در ‏شکل (3-4) نشان داده شده است.

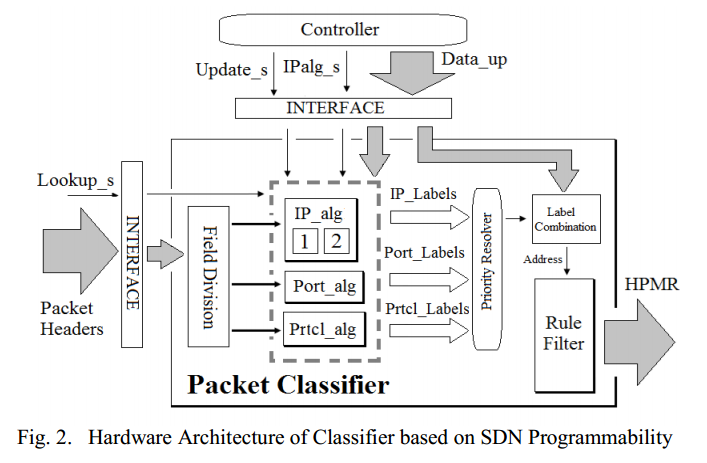


معماری سیستم مطرح شده در [9]

در این معماری سوئیچ‌های لبه مبدا[[117]](#footnote-117) اولین دستگاهی هستند، که بسته‌ها را در شبکه دریافت می‌کنند. لبه مقصد جایی است که در آن بسته شبکه را ترک می‌کند. وقتی بسته در لبه مبدا وارد می‌شود، سوئیچ لبه عمل بسته‌بندی را روی آن انجام می‌دهد، و یک برچسب را روی آن قرار داده، و آن را به گام بعدی ارسال می‌کند. بدین ترتیب هنگامی که بسته به سمت لبه مقصد ارسال می‌شود، تنها فیلدی که مشخص می‌کند بسته کجا ارسال شود، همان برچسب است. بنابراین تمامی سوئیچ‌های هسته از جدول‌های ارسال مبتنی بر برچسب استفاده می‌کنند. سوئیچ لبه مقصد برچسب را از بسته حذف کرده و آن را به بیرون می‌فرستد. مدیریت برچسب، و اختصاص دادن آن‌ها به قطعه‌ها توسط کنترلر انجام می‌شود. در این معماری عمده بار دسته‌بندی بسته در لبه مبدا انجام می‌شود.

* + 1. پیکربندی پویای الگوریتم‎های دسته‌بندی

سیستم پیشنهادی در [10] از برنامه‌نویسی و انعطاف‌پذیری ارائه شده توسط SDN، بهره برده است. در این کار یک طراحی سخت‌افزاری مبتنی بر انتخاب بهترین الگوریتم جستجو نسبت به کاربرد پیشنهاد شده، به طوری که در آن این امکان وجود دارد که فرآیند جستجوی مناسب توسط کنترلر انتخاب گردد. این ساختار در ‏شکل (3-5) نشان داده شده است.



معماری سخت‌افزار مطرح شده در [10]

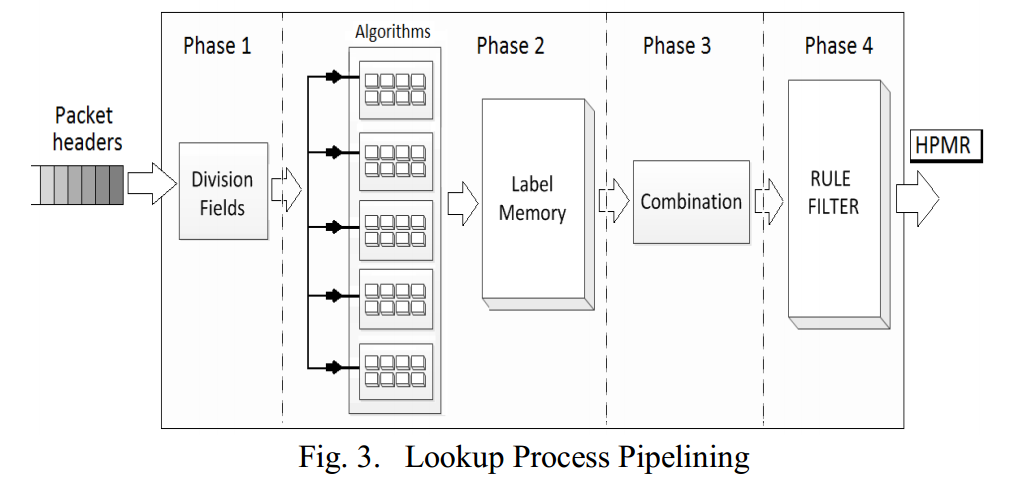
* عملکرد کنترلر

لازم است ساختار الگوریتم برای درج یا حذف قانون، به صورت افزایشی بروزرسانی شود. در این سیستم فرآیند بروزرسانی توسط کنترلر انجام می‌شود. معماری سیستم با انتخاب الگوریتم مناسب که بر اساس مشخصه‌‌های برنامه‌کاربردی انتخاب می‎گردند، قابل پیکربندی است.

انتخاب ترکیب بهینه الگوریتم‌ها بر عهده کنترلر است. در این سیستم این انتخاب بین دو الگوریتم IP (Multi-bit Trie، و Binary Search Tree) با استفاده از سیگنال IPalg\_s صورت می‌گیرد. در ادامه کنترلر اطلاعات مورد نیاز برای پیکربندی بلوک‌های حافظه سکوی سخت‌افزاری را ارسال می‌کند.

* فرآیند جستجو

فرآیند جستجو در 4 فاز انجام می‌گیرد. فاز اول توسط سیگنال lookup\_s، که آغازگر فرآیند جستجو است، برانگیخته می‌شود. همزمان سرآیند بسته به قطعاتی جدا تبدیل شده و به الگوریتم‌های مربوطه که توسط کنترلر انتخاب شده‌اند، ارسال می‌گردند. در فاز دوم، الگوریتم‌های انتخاب شده جستجو را به صورت موازی انجام می‌دهند. نتیجه هر الگوریتم یک اشاره‌گر به لیستی از برچسب‌های تطابق یافته است، که به فاز بعدی داده می‎‌شود. در فاز سوم برچسب‌های خروجی به منظور یافتن قانون تطابق یافته با اولویت بالا (HPMR‍[[118]](#footnote-118))، ادغام[[119]](#footnote-119) می‌شوند. فاز پایانی دسترسی به حافظه و گرفتن HPMR و عمل همراه با آن است. جزئیات فرآیند جستجو در ‏شکل (3-6) نشان داده شده است.



خط لوله فرآیند جستجو

یک پیکربندی نمونه از فرآیند جستجو به صورت الگوریتم Multi-bit Trie، به گذردهی 133 میلیون جستجو در ثانیه رسیده است. با در نظر گرفتن میانگین اندازه بسته 100 بایت، این معادل گذردهی بیش از 100 Gbps است. اگر سیستم با الگوریتم Binary Search Tree پیکربندی شود، اندازه مجموعه قوانین در آن می‌تواند بزرگ‌تر باشد.

* 1. حل مسئله به صورت الگوریتمی

پس از بررسی برخی از روش‌هایی که از ویژگی‌های خاص شبکه نرم‌افزار محور استفاده کرده‌اند، در این بخش، راه‌حل‌هایی بررسی می‌شوند که مسئله را بدون استفاده از ویژگی‌های خاص SDN حل کرده‌اند. به طور کلی این خانواده از روش‌های دسته‌بندی بسته، در چهار گروه زیر تقسیم می‌شوند:

1. **روش‌های ساختار داده‌ای:** از اولین ساختارهای داده‌ای که در مسئله دسته‌بندی اعمال شد، ساختار Binary-Trie [46] است که بسیاری از مقالات ارائه شده از آن استفاده می‌کنند. در اکثر ساختارهای معرفی شده، هدف بالا بردن سرعت جستجو است. از دیگر روش‌های مربوط به این دسته، روش‌های مبتنی برجدول‌های درهم‌سازی[[120]](#footnote-120) هستند. در این گونه روش‌ها با استفاده از درهم‌سازی فیلدهای سرآیند بسته، سطری را که قانون منطبق با بسته در آن قرار دارد مشخص می‌شود [47]. مشکل اصلی در این روش بدست آوردن تابع درهم‌سازی است. مشکل مهم در درهم‌سازی، مسئله تصادم[[121]](#footnote-121) است. از طرف دیگر به دلیل همپوشانی قوانین و وجود اولویت در آن‌ها، پیچیدگی درهم‌سازی افزایش می‌یابد. به همین دلیل معمولاً از درهم‌سازی برای تسریع در روش‌های دیگر استفاده می‌شود. برخی از روش‌های مبتنی بر ساختار داده عبارت‌اند از: درختواره‌های سلسله مراتبی[[122]](#footnote-122)، درختواره‌های با شاخه‌های هرس شده[[123]](#footnote-123)، درخت جستجوی دودویی سلسله مراتبی[[124]](#footnote-124) و ... .
2. **روش‌های مبتنی بر هندسه:** مسئله دسته‌بندی بسته را می‌توان به یک مسئله هندسی نگاشت کرد، به طوری که تعداد فیلدها تعیین کننده تعداد ابعاد فضای مسئله بوده و قوانین به صورت ابرمستطیل‌هایی در این فضا هستند. هر بسته ورودی به صورت نقطه‌ای در این فضای چند بعدی می‌باشد. در این روش هدف یافتن ابرمستطیلی است که در برگیرنده آن نقطه بوده و دارای بالاترین اولویت در بین تمام ابرمستطیل‌هایی باشد که این شرط را ارضاء می‌کنند. روش‌های درخت FIS[[125]](#footnote-125) و درخت GST[[126]](#footnote-126) جزو این روش‌ها هستند.
3. **روش‌های اکتشافی:** به دلیل وجود پیچیدگی و چند بعدی بودن مسئله دسته‌بندی بسته، نیاز است که در روش‌ها از ابتکاراتی استفاده شود. برخی از روش‌های اکتشافی[[127]](#footnote-127) عبارتند از روش انقطاع هوشمندانه سلسله مراتبی[[128]](#footnote-128)، جستجوی فضای چندتایی[[129]](#footnote-129) و ... .
4. **روش‌های سخت‌افزاری:** حافظهTCAM‍[[130]](#footnote-130) نوعی حافظه است که دسترسی به عناصر آن از طریق مقدار انجام می‌گیرد و به همین دلیل برای انجام عمل دسته‌بندی می‌تواند مناسب و سریع باشد. مشکل روش‌های مبتنی بر CAM گرانی و محدودیت حجم آن می‌باشد. به دلیل محدودیت‌های زیاد در پیاده‌سازی‌های سخت‌افزاری، این روش‌ها از لحاظ اندازه مجموعه قوانین به سختی قابل مقیاس هستند. برخی از روش‌های سخت‌افزاری عبارت‌اند ازTernay CAM ، اشتراک روی نقشه بیت‌ها[[131]](#footnote-131) و ... .
   * 1. الگوریتم‌های ارائه شده بدون تاکید بر تعداد فیلد زیاد

در این بخش برخی از الگوریتم‎هایی که بدون تاکید بر تعداد زیاد فیلد، برای مسئله دسته‌بندی بسته ارائه شده‌اند به طور مختصر توضیح داده شده است.

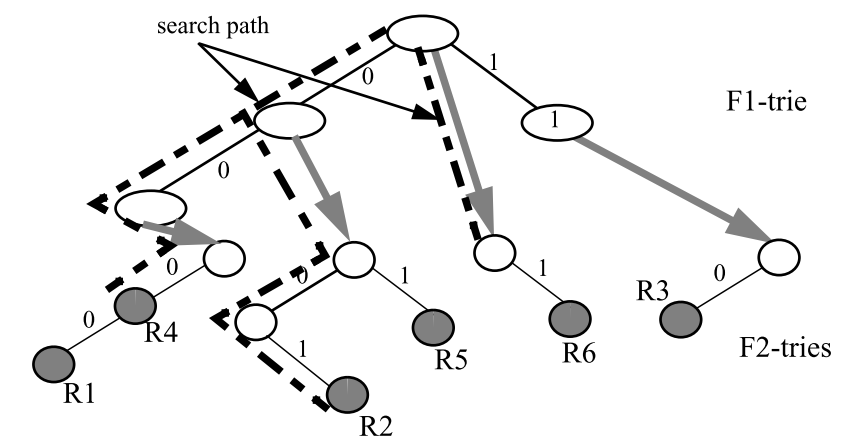
* جستجوی خطی

این روش ساده‌ترین نوع جستجو است که به طور ترتیبی قوانین را مورد جستجو قرار می‌دهد. در این روش قوانین به ترتیب اولویت در یک آرایه قرار می‌گیرند. بنابراین با اولین قانونی که با بسته منطبق می‌شود، عمل جستجو خاتمه می‌یابد. پیچیدگی زمانی و حافظه‌ای این روش است (n تعداد قوانین موجود در جدول می‌باشد). هرچند این روش ساده از نظر حافظه مصرفی به صرفه است، ولی مقیاس‌پذیری پایینی دارد و در آن زمان دسته‌بندی بسته‌ به طور خطی با افزایش تعداد قوانین افزایش می‌یابد.

* درختواره سلسله مراتبی

یک درختواره سلسله مراتبی d بعدی [34] بسطی از ساختار داده‌ی درختواره یک بعدی می‌باشد که به طور بازگشتی ساخته می‌شود. اگر d بزرگتر از 1 باشد، ابتدا یک درختواره یک بعدی می‌سازیم و آن را می‌نامیم، این درختواره روی مجموعه پیشوندهای فیلد اول ساخته می‌شود. به ازای هر پیشوند در ، به طور بازگشتی یک درختواره سلسله مراتبی d-1 بعدی می‌سازیم. اتصال بین هر دو درختواره نشانگر درختواره بعدی[[132]](#footnote-132) نامیده می‌شود. بعد سوم با توجه به تعداد پیشوندهای بعد دوم و بقیه ابعاد نیز به همین ترتیب ترسیم می‌شوند تا کل درختواره کامل گردد. این روش به نام‌های "درختواره‌های چند سطحی"، و "درختواره‌ای از درختواره‌ها " نیز شناخته می‌شود.

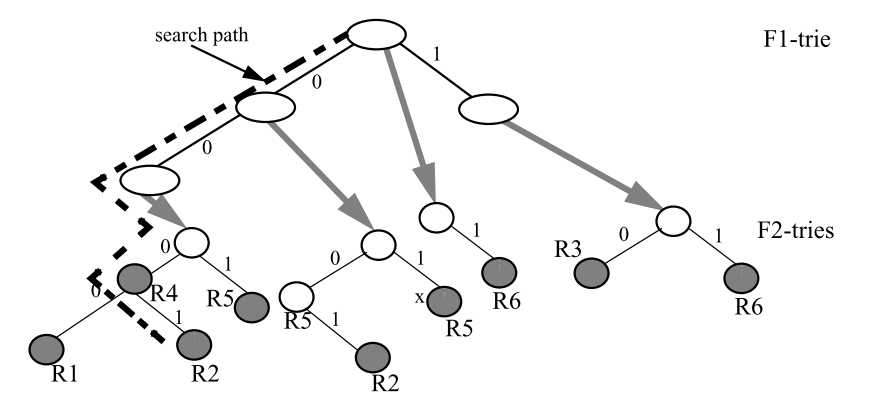
دسته‌بندی یک بسته ورودی به صورت زیر است. الگوریتم ابتدا درختواره را بر اساس بیت‌های موجود در پیمایش می‌کند. به هر برگ از این درختواره که وارد شود، اگر نشانگر درختواره بعدی وجود داشته باشد آن را پی می‌گیرد و درختواره d-1 بعدی را به طور بازگشتی پیمایش می‌کند. بنابراین این الگوریتم قانونی را در پیمایش خود نگه می‌دارد که با بسته ورودی منطبق می‌شود. ‏شکل (3-7) مثالی از این ساختار را نشان می‌دهد.



مثالی از ساختار درختواره‌های سلسله مراتبی

* درختواره‌های با شاخه‌های هرس شده

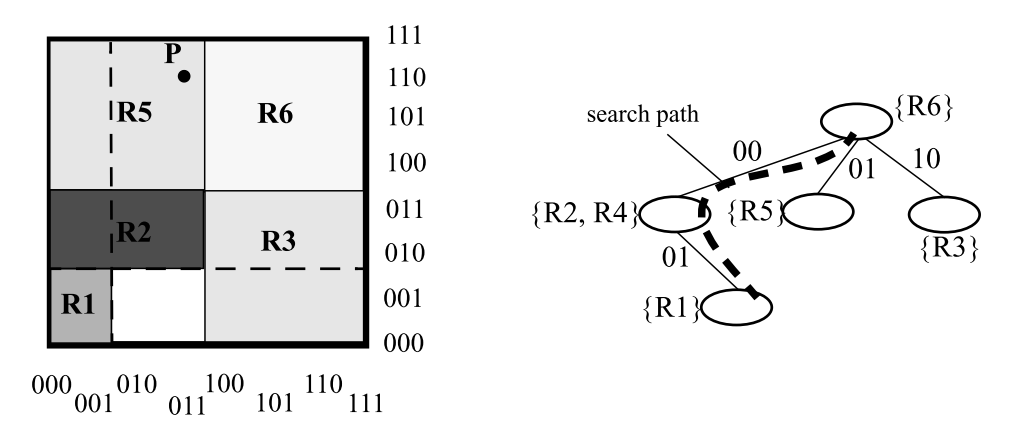
درختواره با شاخه‌های هرس شده [34] شبیه ساختار قبلی است، با این تفاوت که در آن با حذف پیمایش‌های بازگشتی، زمان جستجو کاهش داده می‌شود. این تسریع با تکرار قوانین در بعضی از گره‌‎های ساختار بدست می‌آید. یک درختواره سلسله مراتبی d بعدی را در نظر بگیرید که شامل یک درختواره و چندین درختواره d-1 بعدی دیگر است. فرض کنید S مجموعه نودهایی است که در نشان‌دهنده پیشوندهای طولانی‌تر از پیشوند p هستند. یک درختواره با شاخه‌های هرس‌شده شبیه به درختواره سلسله مراتبی است، با این تفاوت که قوانین موجود در درختواره سلسله مراتبی d-1 بعدی که مرتبط با پیشوند p هستند، در درختواره‌های d بعدی که به گره‌های S وصل هستند، تکرار می‌شوند. این تکرار باعث می‌شود هنگام جستجو به بازگشت به عقب نیاز نداشته باشیم. ‏شکل (3-8) مثالی از ساختار این روش را نشان می‌دهد.



مثالی از ساختار درختواره‌های با شاخه‌های هرس‌شده

* Area-based quadtree

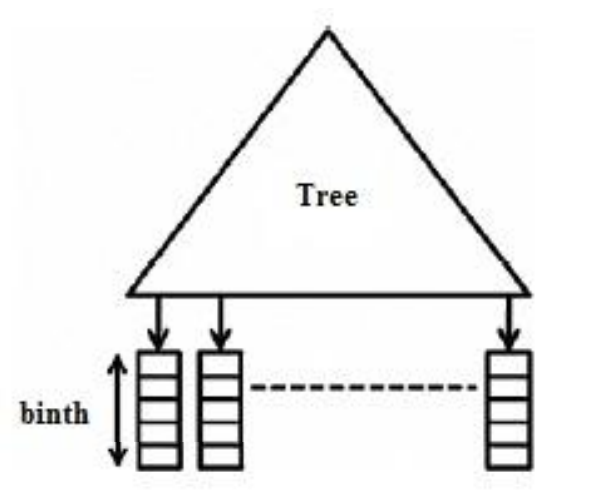
این روش در مرجع [48] برای جستجو در دو بعد ارائه شده است. در این روش پارامتر قابل تنظیمی وجود دارد که با استفاده از آن می‌توان یک تعادل بین زمان جستجو و انعطاف در تغییر قوانین ایجاد کرد. ابتدا یک درخت چهارتایی (درختی که هر گره داخلی آن دارای چهار فرزند است) ساخته می‌شود. گره پدر نمایش‌دهنده‌ی یک فضای دو بعدی است که به چهار قسمت هم اندازه تقسیم شده و هر قسمت توسط یک گره فرزند نشان داده می‌شود. به این ترتیب فضای اولیه به چهار مربع مساوی تقسیم می‌گردد. این تقسیم برای هر بخش تا جایی که در آن بیش از یک قاعده وجود داشته باشد، ادامه می‌یابد. هر بیت از دو بیت مربوط به این تقسیم‌بندی از یک بعد گرفته می‌شود.



مثالی از ساختار AQT

* انقطاع هوشمندانه سلسله مراتبی (HiCut)

در این روش که در [17] پیشنهاد شده، یک درخت تصمیم به طوری که نودهای داخلی آن حاوی اطلاعاتی برای هدایت الگوریتم دسته‌بندی باشند و برگ‌های آن تعداد کمی از قوانین را در خود ذخیره کنند، ایجاد می‌گردد. برای دسته‌بندی هر بسته ورودی، الگوریتم درخت ساخته شده را پیمایش می‌کند تا به یک برگ برسد. سپس با جستجوی خطی روی قوانین موجود در آن برگ، بهترین قانون منطبق با بسته را می‌یابد. ساختار درخت به گونه‌ای ساخته می‌شود که تعداد قوانینی که در هر برگ قرار می‌گیرند به یک عدد کوچک (به نام binth) محدود باشند. این ساختار در ‏شکل (3-10) نشان داده شده است.



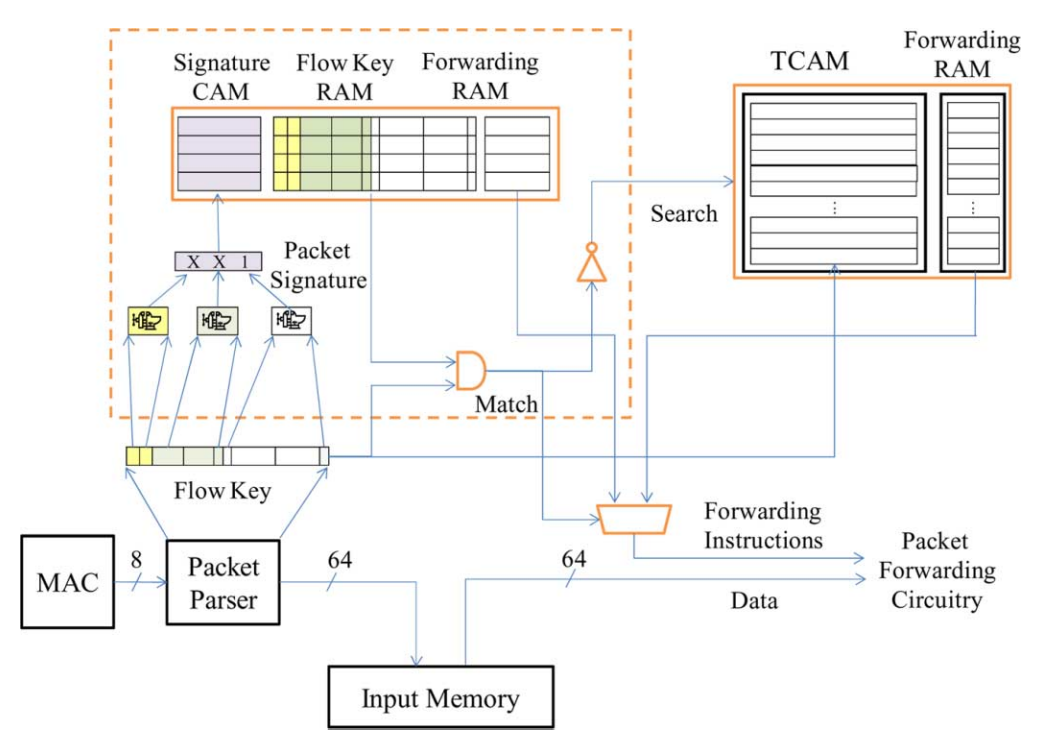
ساختار درخت در روش HiCut

الگوریتم HiCut برای d بعد به صورت زیر عمل می‌کند: هر گره داخلی نمایش دهنده‌ی یک بخشی از فضای هندسی جستجو و گره ریشه نمایش دهندهی کل فضای هندسی در d بعد است. فضای هر گره با انقطاع روی یکی از ابعاد به زیرفضاهایی شکسته می‌شود که این زیرفضاها با گره‌های فرزند آن نشان داده می‌شوند. زیرفضاها به طور بازگشتی تا جایی شکسته می‌شوند که در هر زیرفضا بیشتر از binth قانون قرار نگیرد که در این صورت این قوانین به آن گره اختصاص می‌یابند [49].

در این بخش برخی از الگوریتم‌های دسته‌بندی بسته که در آن‌ها تاکیدی بر تعداد زیاد فیلد وجود نداشت، به طور مختصر معرفی شد. در بخش‌های بعدی کارهایی که با تاکید بر تعداد زیاد فیلد به این مسئله پرداخته‌اند، بررسی شده است.

* + 1. مدار پیش‌بینی کننده جریان

معماری سوئیچ در [11] شامل یک مدار پیش‌بینی کننده جریان در هر پورت است. هدف از این مدار، پیش‌بینی هر چه سریعتر کلید جریان برای بسته دریافتی است، به طوری که دیگر نیازی به استفاده از فرآیند جستجو و حافظه TCAM نباشد. در ‏شکل (3-11) نحوه ایجاد امضا برای بسته توسط مدار نشان داده شده است.



مدار پیش‌بینی کننده جریان به ازای هر پورت

در این جا بعد از دریافت بسته، در حالی که کلید جریان در حال ساخته شدن است، بیت‌های مهمی از بسته استخراج شده و برای تولید امضا بر اساس روش پیش‌بینی استفاده می‌شوند. امضاء بسته در یک کش پیش‌بینی محلی، جستجو می‌شود. اگر دقیقا یک مورد مطابقت پیدا شود، فرض می‌شود که بسته بخشی از همان جریانی است که بسته قبلی با آن امضا، متعلق به آن بوده و عمل ارسال فوراً می‌تواند آغاز شود. این امر باعث کاهش تاخیر می‌گردد. از آن جایی که مدار پیش‌بینی به ازای هر پورت مورد نیاز است، یافتن پیاده‌سازی کوچک و کارآمد ممکن، برای آن امر ارزشمندی خواهد بود.

بخش flow-key RAM در ‏شکل (3-11) کلید جریان را برای بسته‌ای که اخیراً با آن امضا مطابقت پیدا کرده است، نگه می‌دارد. این کلید جریان در RAM با کلید جریانی که packet parser از بسته به دست آورده است، مقایسه می‌شود تا وجود مطابقت را تایید کند. این مقایسه برای جلوگیری از انجام فرآیند جستجوی کامل برای هر بسته لازم بوده و به این ترتیب توان مصرفی را کاهش می‌دهد.

پس از انجام مقایسه بین کلید‌های جریان، سه حالت ممکن است پیش بیاید:

1. پیش‌بینی صحیح: در این حالت کلید جریان بسته با کلید جریانی که در RAM هست، مطابقت دارد. در این مورد یک پیش‌بینی صحیح اتفاق افتاده و نیازی به انجام عمل اضافی دیگری نیست. در واقع نیازی به جستجو در TCAM نیست، و توان مورد نیاز برای این کار ذخیره می‌شود. تاخیر در این حالت کمترین مقدار خود را خواهد داشت، چرا که عمل ارسال قبلا آغاز شده است.
2. پیش‌بینی ناصحیح: این حالتی است که امضا در کش وجود داشته، ولی کلیدهای جریان با هم تطبیق نیافته‌اند. در این صورت عمل ارسال که قبل آغاز شده است، می‌بایست لغو شود. جستجو در حافظه TCAM باید انجام شود، و کش پیش‌بینی محلی باید بروزرسانی گردد. در این حالت توانی که برای جستجوی کش پیش‌بینی و انتقال بسته مصرف شده، هدر رفته است.
3. عدم پیش‌بینی: هیچ کلید جریانی در منطق پیش‌بینی یافته نشده است. بنابراین فرآیند جستجو بایستی به صورت کامل انجام گیرد. همانند حالت قبل کش پیش‌بینی باید بروزرسانی شود و توان مصرفی برای جستجو در کش و بروزرسانی آن هدر رفته است.

نتایج ارائه شده در این کار نشان داده که می‌توان به نرخ پیش‌بینی 97% رسید. هم‌چنین توان مصرفی فرآیند جستجویی که از TCAM استفاده می‌کند، حدود 92% و تاخیر در حدود 66% کاهش یافته است.

* + 1. تجزیه فیلد‌ها و ادغام نتایج جزئی

در [12] یک روش مبتنی بر تجزیه بر روی سکوهای مختلف (FPGA، پردازنده‌های چندهسته‌ای، و پردازنده‌های گرافیکی) پیاده‌سازی شده است. در روش‌های مبتنی بر تجزیه، فیلد‌های سرآیند بسته، به چندین فیلد تقسیم شده، و عملیات جستجو برای هر فیلد به طور مجزا و مستقل انجام می‌شود. سپس نتایج جزئی به دست آمده از جستجوها با هم ادغام[[133]](#footnote-133) و نتیجه نهایی به دست می‌آید. به طور کلی یک روش مبتنی بر تجزیه دارای سه فاز است: فاز پیش پردازش[[134]](#footnote-134)، فاز جستجو، و فاز ادغام. مزیت اصلی روش‌های مبتنی بر تجزیه این است که در فاز جستجو می توان از الگوریتم‌های موازی، و ساختمان داده‌های مختلف (مانند درخت محدوده[[135]](#footnote-135)، و جدول‌های درهم‌سازی) استفاده کرد.

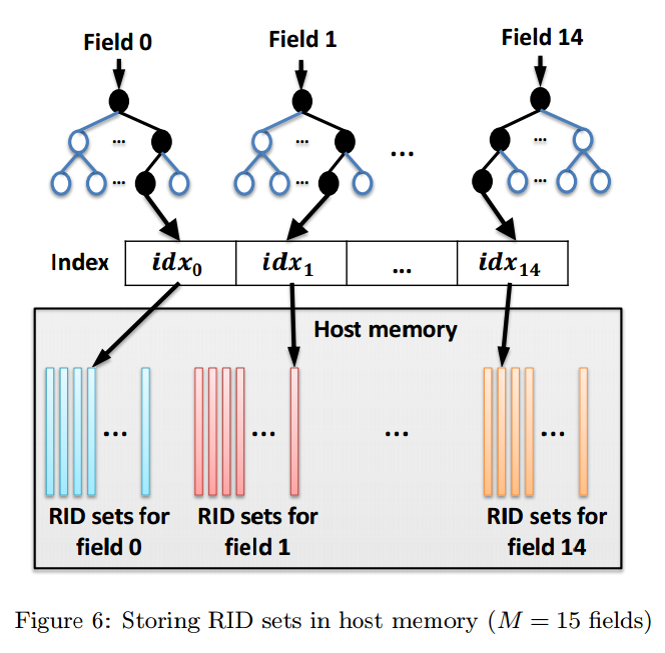
* فاز پیش‌پردازش

در [12] در فاز پیش‌پردازش ابتدا تمامی مقادیر به صورت محدوده‌ای نمایش داده می‌شوند. در واقع برای یکسان‌سازی نوع عمل تطابق در فیلدهای مختلف، مقادیر پیشوندی و مقادیر دقیق هم به یک محدوده تبدیل می‌شوند. هم چنین برای جستجوی موازی کارآمد، از ساختمان داده درخت محدوده استفاده شده است. به طوری که مقادیر فیلد‌ها به زیرمحدوده‌‌های[[136]](#footnote-136) بدون اشتراک تقسیم شده، و سپس یک درخت محدوده (range-tree) کامل با استفاده از کران‌های زیرمحدوده‌ها در هر فیلد تشکیل می‌شود.

* فاز ادغام

فرض کنید تعداد زیرمحدوده‌ها در فیلد m () با نشان داده ‌شود. بعد از فاز جستجو نتایج جزئی هر فیلد را می‌توان در یک بردار بیتی[[137]](#footnote-137) (BV) به طول N نشان داد. در این صورت مقدار حافظه کل مصرفی خواهد بود. این مقدار، به خصوص وقتی *N* بزرگ باشد، خیلی بزرگ می‌شود، و می‌تواند کارآیی کش را پایین آورد. در این جا برای بهبود کارآیی سعی شده از حافظه کوچکی برای نتایج جزئی استفاده شود. در واقع از مجموعه شناسه قوانین[[138]](#footnote-138) (RID) برای نمایش نتایج جزئی جستجو استفاده شده است. در هر فیلد، یک مجموعه RID، تمامی RIDهایی که با ورودی مطابقت یافته‌اند را، به ترتیب صعودی در خود نگه می‌دارد. این مجموعه‌ها در فاز پیش‌پردازش محاسبه شده و در برگ‌های درخت محدوده قرار می‌گیرند. فرض کنید بیشینه تعداد قوانین مطابقت یافته با تمامی ورودی‌ها در فیلد m باشد. در این صورت یک مجموعه RID به حافظه نیاز خواهد داشت. بنابراین برای تمامی *M* فیلد، حافظه کل مصرفی برای ذخیره تمامی مجموعه‌ها برابر خواهد بود با:

تجربیات نشان داده که معمولاً خیلی کوچک است. این یعنی نمایش نتایج جزئی به صورت مجموعه RID، نسبت به نمایش BV، کارآمدتر خواهد بود. استفاده از مجموعه RID، تفاوت عمده این کار با سایر کارها است.



ذخیره مجموعه‌های RID در میزبان و استفاده از اندیس در GPU

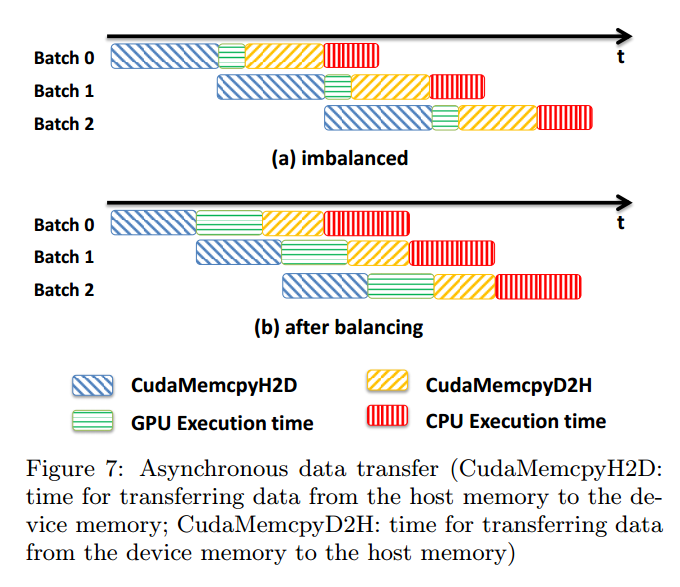
* فاز جستجو

فاز جستجو روی سکوهای مختلفی مثل FPGA، پردازنده‌های چندهسته همه منظوره، و پردازنده‌های گرافیکی پیاده‌سازی، و در هر کدام از تکنیک‌هایی برای بهبود کارآیی استفاده شده است. در این جا تکنیک‌هایی که برای بهبود کارآیی در پردازنده گرافیکی استفاده شده است، را آورده‌ایم.

دو چالش عمده هنگام دسته‌بندی بسته با استفاده از GPU عبارت‌اند از: 1- واگرایی پرش در تابع کرنل که می‌تواند باعث اتلاف منابع GPU شود، و 2- زمان انتقال داده روی گذرگاه[[139]](#footnote-139) PCIe بین CPU و GPU به نسبت زیاد است و کارآیی کلی را کاهش می‌دهد.

جهت کاهش تاثیر واگرایی پرش، پیاده‌سازی یک درخت l-سطحی با استفاده از یک آرایه به اندازه صورت گرفته است. به طوری که ریشه درخت در اولین مکان آرایه قرار می‌گیرد، و برای گره‌ای در مکان i-ام، فرزند چپ در مکان و فرزند راست در مکان قرار می‌گیرند. این نحوه ارجای ضمنی با استفاده از اندیس i، باعث می‌شود در برنامه CUDA، عملیات اشاره‌گر به عملیات محاسباتی تبدیل گردند. هم چنین از ادغام نتایج جزئی در GPU به دلایل زیرخودداری شده: 1- می‌شود از CPU در کنار GPU برای افزایش گذردهی کلی استفاده کرد، و 2- فاز ادغام واگرایی شاخه‌ای بیشتری نسبت به فاز جستجو دارد.

برای کاهش سربار ارتباطات از دو تکنیک استفاده شده است. اولاً نتایج جزئی در حافظه میزبان قرار داده شده و از اندیس برای نشان دادن آن‌ها استفاده می‌شود. یک مثال از این در ‏شکل (3-12) نشان داده شده است. این تکنیک باعث می‌شود که تنها اندیس‌ها از حافظه دستگاه به حافظه میزبان منتقل شوند. سپس مجموعه RID مربوط به اندیس در CPU استخراج می‌شود.



انتقال داده ناهمگام بین CPU، و GPU

تکنیک دیگر متعادل‌سازی تاخیر بخش‌های مختلف است. توضیح آن که کپی داده‌ها بین CPU و GPU، می‌تواند به صورت ناهمگام[[140]](#footnote-140) باشد. در این صورت همان‌طور که در ‏شکل (3-13) دیده می‌شود، می‌توان بین زمان انتقال داده با زمان اجرای کرنل و زمان اجرای CPU، هم پوشانی ایجاد کرد. این تکنیک می‌تواند گذردهی را بهبود بخشد. در GPU، گذردهی همواره توسط طولانی‌ترین بخش بین بخش‌های زیر تعیین می‌گردد: زمان کپی داده از CPU به GPU، زمان اجرای کرنل، زمان کپی داده از GPU به CPU، و زمان اجرای CPU. ایجاد تعادل بین این 4 مورد می‌تواند منجر به گذردهی بهتر گردد، چرا که هیچ کدام تنگنای دیگری نمی‌شود. بنابراین در ‏شکل (3-13) هر چند قسمت b به طور ایده‌آل متعادل نیست، ولی در کل گذردهی بهتری نسبت به قسمت a خواهد داشت. البته باید توجه داشت که تاخیر در حالت b لزوماً بهتر نمی‌شود.

نتایج این کار در ‏جدول (3-1) گزارش شده است. پیاده سازی مبتنی بر FPGA گذردهی بهتری دارد. پیاده‌سازی روی پردازنده‌های چند هسته‌ای نسبت به تعداد هسته‌ها قابل مقیاس است. پیاده‌سازی روی سکویی با شتاب‌دهنده GPU، نسبت به پیاده‌سازی روی پردازنده چند هسته‌ای گذردهی را 2 برابر بهبود بخشیده است.

نتایج به دست آمده در [12]

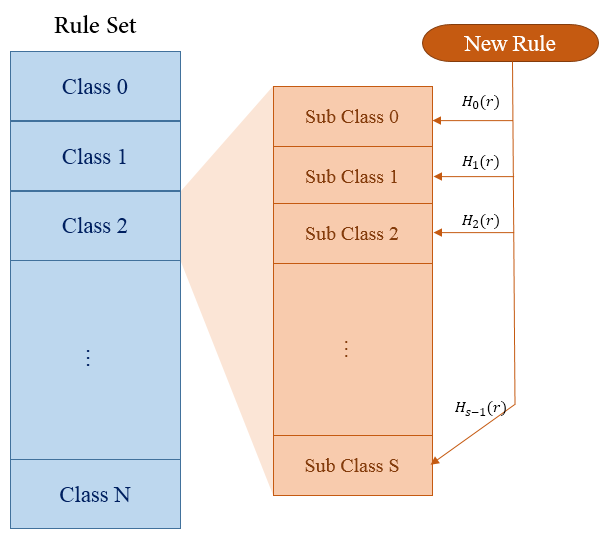
|  |  |  |  |
| --- | --- | --- | --- |
| سکو | گذردهی (MPPS) | تاخیر | اندازه مجموعه قوانین |
| FPGA | 500 | 3 میکروثانیه | 1500 |
| پردازنده‌های چندهسته‌ای | 14.7 | 22.1 میلی ثانیه | 32000 |
| پردازنده‌های گرافیکی | 30 | 22.5 میلی ثانیه | 32000 |

* + 1. GSwitch[[141]](#footnote-141)

در [13] سه الگوریتم دسته‌بندی بسته با بهره‌گیری از توجه دقیق به ویژگی‌های پردازنده‌های گرافیکی به صورت کارآمد، روی این سکو پیاده‌سازی شده است. این سه الگوریتم عبارت‌اند از: الگوریتم جستجوی خطی، الگوریتم جستجوی فضای چندتایی[[142]](#footnote-142)، و الگوریتم جستجوی Bloom. پیاده‌سازی نمونه‌هایی از این الگوریتم‌ها بر روی یک پردازنده گرافیکی توانسته به تسریع 7، 11، و 12 به ترتیب برای الگوریتم‌های جستجوی خطی، جستجوی فضای چندتایی، و جستجوی Bloom نسبت به پیاده‌سازی همین الگوریتم‌ها بر روی پردازنده‌های همه منظوره هم قیمت برسد. در ادامه توضیحاتی راجع به الگوریتم جستجوی فضای چندتایی، و الگوریتم جستجوی Bloom آورده‌ایم.

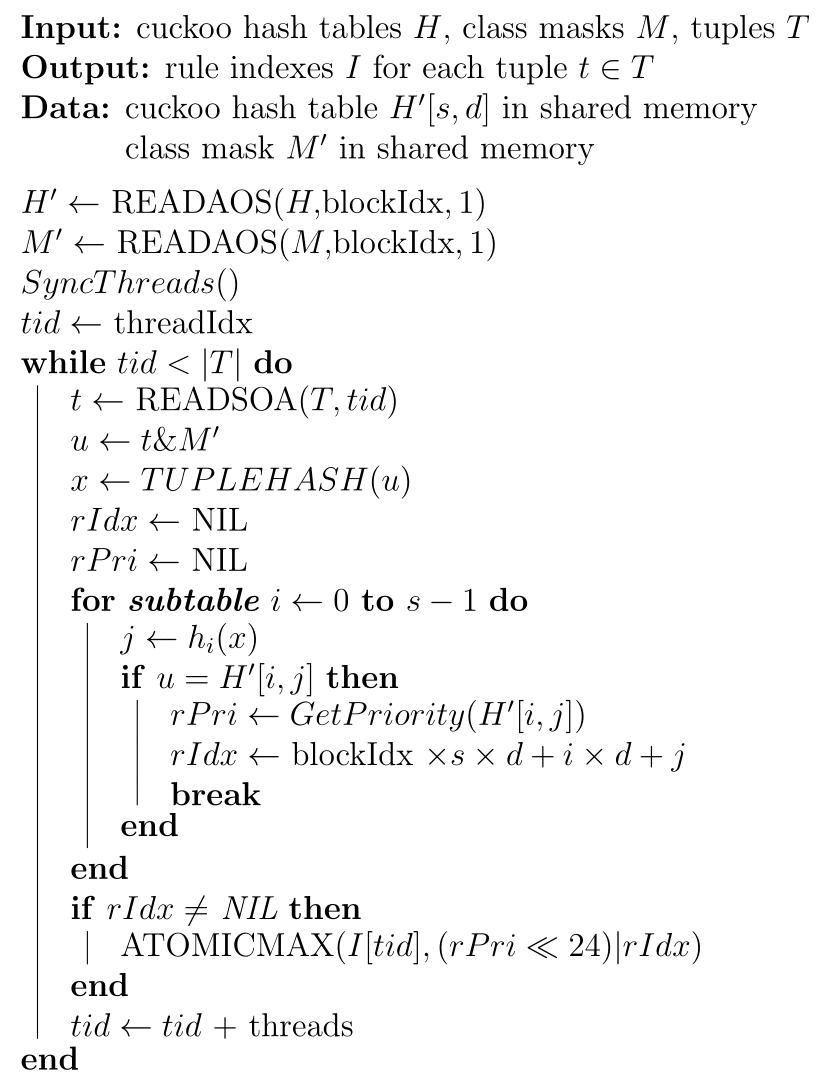
* الگوریتم جستجوی فضای چندتایی

این الگوریتم از Open vSwitch [50] گرفته شده است. Tuple Search از مفهوم کلاس (یا مجموعه‌ای از قوانین با ماسک یکسان) استفاده می‌کند. قوانینی موجود در یک کلاس، در یک جدول کلاس[[143]](#footnote-143) (که برای جستجوی سریع معمولاً به صورت یک جدول درهم‌سازی[[144]](#footnote-144) پیاده‌سازی می‌شود) قرار می‌گیرند. هر کلاس یک ماسک کلاس[[145]](#footnote-145) دارد، که برای تمامی قوانین آن کلاس یکسان است. جدول کلاس با استفاده از cuckoo hashing که زمان جستجوی ثابتی دارد، پیاده‌سازی شده. در cuckoo hashing هر جدول از s زیرجدول هم اندازه، که هر کدام d شکاف[[146]](#footnote-146) دارند، تشکیل شده است.



عملیات درج قوانین در Tuple Space Search

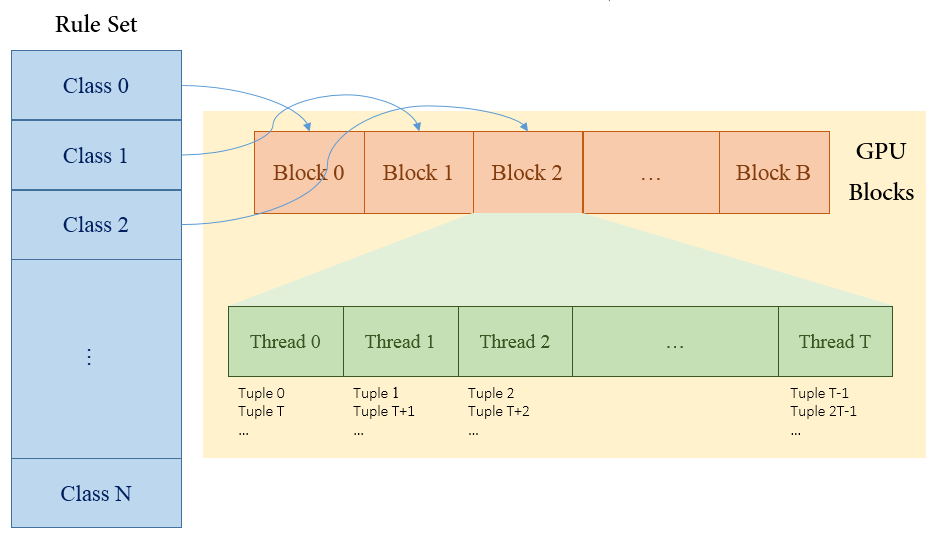
در این الگوریتم عملیات درج قانون به صورت زیر انجام می‌شود: فرض کنید ، که در آن است، مجموعه قوانینی باشد، که قرار است در جدول جریان درج شوند. برای درج قانون ، ابتدا کلاس مربوط به آن مشخص شده و در آن کلاس مقادیر ، ، …، و با استفاده از مجموعه توابع درهم‌سازی مستقل و یکنواخت[[147]](#footnote-147) محاسبه می‌شوند. این مقادیر به عنوان اندیس‌های هر یک از زیرجدول‌های مربوطه استفاده می‌شوند (‏شکل (3-14) ). اگر یکی از این شکاف‌ها خالی باشند، قانون در آن درج شده و عملیات به پایان می‌رسد. در غیر این صورت یک زیرجدول q انتخاب شده، و قانون که در شکاف مورد نیاز ذخیره شده است، خارج می‌شود تا فضایی برای ایجاد شود. سپس روند درج برای تکرار می‌گردد. اگر این روند به تعداد مشخصی تکرار شود، سپس یک مجموعه جدیدی از توابع درهم‌سازی انتخاب شده و تمامی قوانین مجدداً درج می‌شوند. اگر ابعاد جدول‌ها به خوبی انتخاب شود، عملیات درهم‌سازی مجدد به ندرت انجام می‌شود.



الگوریتم 1 شبه کد الگوریتم جستجوی tuple space

عمل تطابق در این الگوریتم به این صورت انجام می‌شود که ابتدا برای تمامی کلاس‌ها، با استفاده از ماسک کلاس، عملیات masking روی چندتایی انجام می‌شود. سپس در جدول کلاس عملیات جستجو صورت می‌گیرد. فرض کنید u خروجی ماسک‌شده چندتایی t، در یک کلاس باشد. در این جا باید مقادیر ، ، ... ، و محاسبه شده، و شکاف‌های متناظر در زیرجدول‌های مربوطه مورد بررسی قرار گیرند. این عملیات در بدترین حالت در انجام می‌شود. در نهایت قانون با اولویت‌تر به عنوان نتیجه نهایی انتخاب می‌شود.

الگوریتم 1 شبه کد کرنل مربوط به tuple space را نشان می‌دهد. کرنل آرایه‌های H، M، و T را که به ترتیب شامل جدول‌های در هم‌سازی cuckoo، ماسک کلاس، و چندتایی‌هایی که باید جستجو شوند، می‌باشند را به عنوان ورودی دریافت می‌کند. همانند الگوریتم قبل، کرنل اندیس قانون با اولویت‌تر برای هر چندتایی را، در آرایه I بر می‌گرداند. در این طرح هر بلوک مسئولیت بررسی چندتایی‌ها در یک کلاس است. برای بهبود کارآیی عمل تطابق، هر بلوک یک کپی از جدول کلاس، و mask را در H' و M' که در زمان اجرا در حافظه اشتراکی بارگذاری می‌شوند، نگه می‌دارد. نحوه نگاشت کلاس‌ها و چندتایی‌ها، بر روی بلوک‌ها و ریسمان‌های GPU در ‏شکل (3-15) نشان داده شده است.

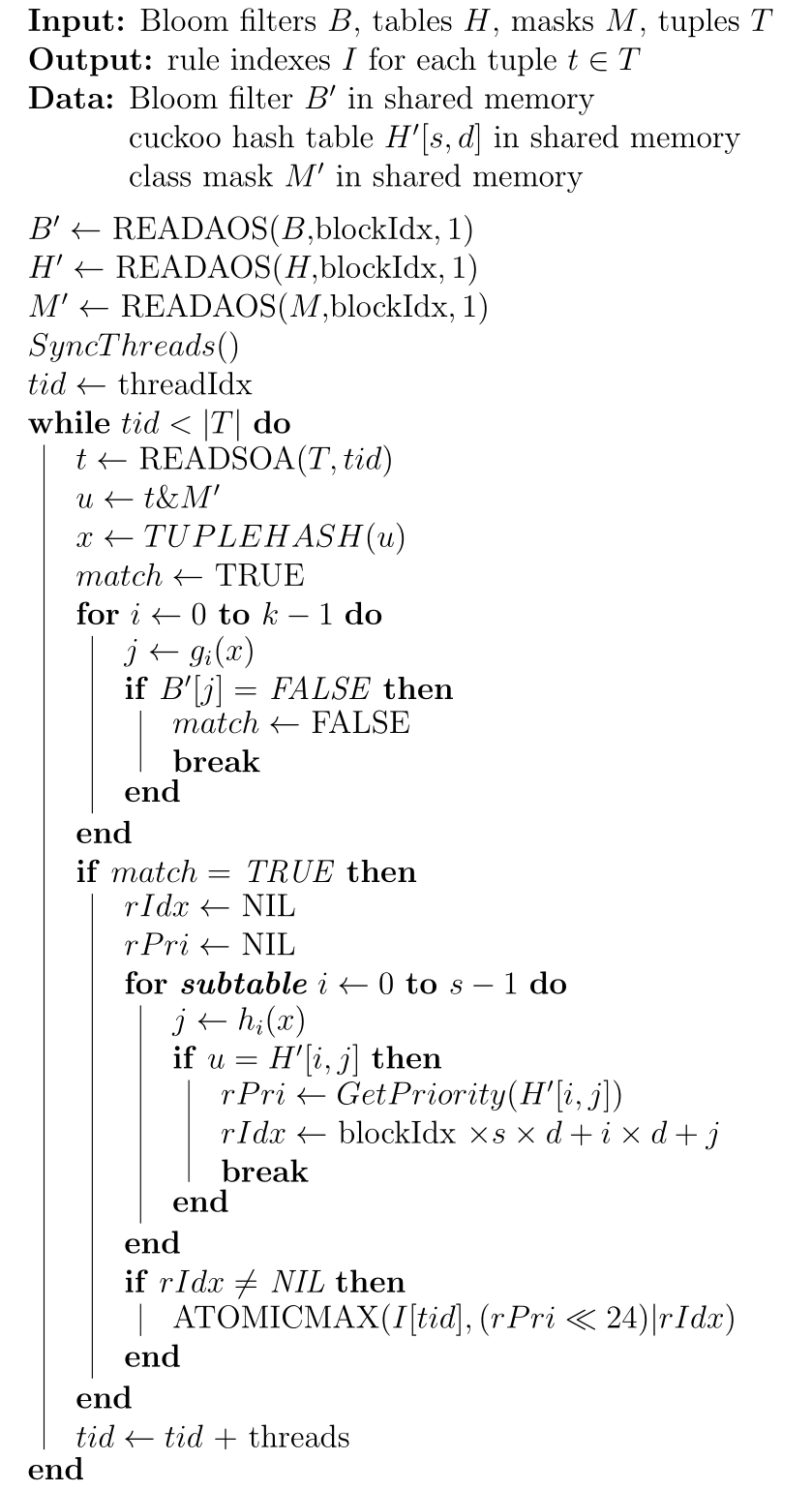


نحوه نگاشت بین کلاس‌ها و چندتایی‌ها به بلوک‌ها و ریسمان‌های GPU

* الگوریتم جستجوی Bloom

در الگوریتم tuple search نیاز است تا یک چندتایی در تمامی جدول‌های کلاس بررسی شود. بنابراین در این جا پیشنهاد شده که از فیلتر Bloom به عنوان یک نمایش جمع و جور از جدول‌های کلاس استفاده شود. در این صورت کلاس‌هایی که عمل تطابق باید در آن‌ها انجام شود، به سرعت مشخص شده و جستجو تنها در آن کلاس‌ها صورت می‌گیرد.

یک فیلتر Bloom، ساختمان داده‌ای است که از لحاظ فضای مصرفی کارآمد می‌باشد. این فیلتر از یک آرایه با m بیت و k تابع درهم‌سازی مستقل ، ، ... ، و که خروجی آن‌ها به طور یکنواخت در بازه توزیع می‌شوند، تشکیل شده است.



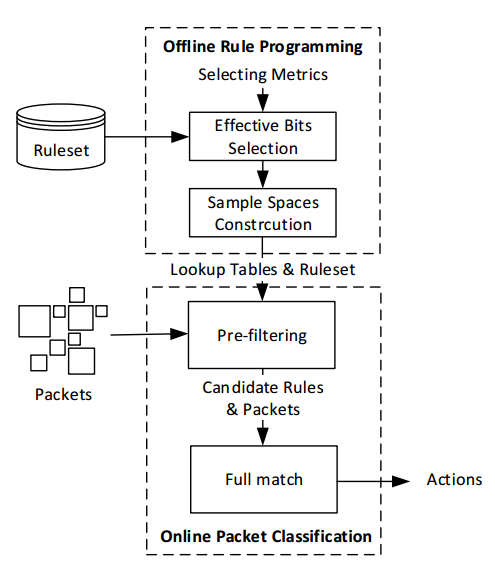
الگوریتم 2 شبه کد الگوریتم جستجوی بلوم

در این جا از فیلتر Bloom برای نمایش مجموعه در یک کلاس استفاده شده است. در ابتدا مقدار تمامی بیت‌های فیلتر صفر قرار داده می‌شود. سپس برای هر قانون بیت‌هایی که در موقعیت‌های ، ، ... و ، قرار دارند، برابر با 1 قرار می‌شوند. عملیات جستجو برای یک چندتایی t در کلاس C به صورت زیر تغییر می‌کند: ابتدا t با ماسک کلاس، ماسک شده و u به دست می‌آید. سپس بیت‌هایی که در موقعیت‌های ، ، ... و قرار دارند، بررسی می‌گردند. اگر حداقل یکی از این بیت‌ها صفر باشند، به این معنی است که قانونی که با این چندتایی مطابقت یابد، در این کلاس وجود ندارد. در غیر این صورت با احتمال زیاد یک قانون در این کلاس با این چندتایی تطابق می‌یابد (امکان ایجاد خطای مثبت کاذب[[148]](#footnote-148) با احتمال کمی وجود دارد).

الگوریتم 2 شبه کد کرنل مربوط به جستجوی Bloom را نشان می‌دهد. این کرنل مشابه کرنل tuple space search است و تفاوت اصلی بین دو کرنل در این است که در این جا باید فیلتر Bloom هم در زمان اجرا در حافظه اشتراکی بارگذاری شود، و بیت‌های آن برای هر چندتایی بررسی گردند. هم‌چنین جستجوی جدول‌ها تنها در صورتی انجام می‌شود، که یک قانون توسط فیلتر یافت شود.

* + 1. MC-SBC[[149]](#footnote-149)

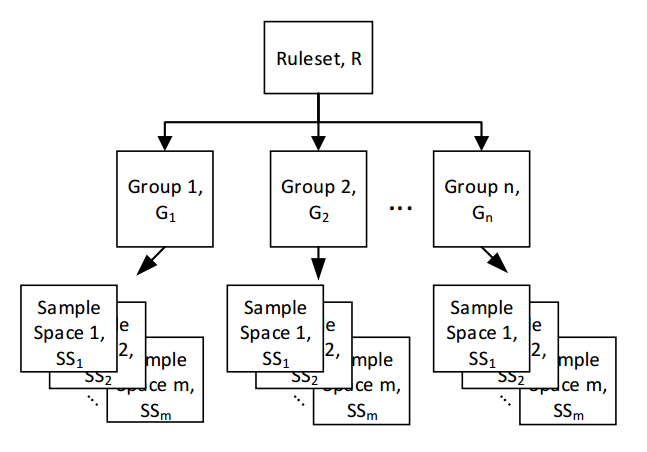
در [14] یک سیستم با معماری دو مرحله‌ای برای دسته‌بندی بسته ارائه شده است. در این معماری از جدول‌های جستجوی ساده برای جلوگیری از مسئله انشعاب محاسباتی استفاده شده، و در سکوهای محاسباتی از دستورات با کارآیی بالا برای گذردهی بیشتر سیستم، بهره برده شده است. ایده اصلی این کار از روش HiCut [17] که در آن یک درخت تصمیم به صورت هوشمندانه ایجاد می‌شود، گرفته شده است. در این جا به دلیل استفاده از پردازنده‌های گرافیکی برای جستجوی بسته‌ها یک درخت دو-سطحی ایجاد می‌شود، تا تاثیر مشکل واگرایی پرش که باعث کاهش کارآیی در این پردازنده‌ها می‌شود، کاهش یابد. هم چنین از مفهوم بیت‌های موثر[[150]](#footnote-150) جهت افراز مجموعه قوانین و ایجاد درخت استفاده شده است. بیت‌های موثر، بیت‌هایی هستند که با استفاده از آن‌ها می‌توان به بهترین صورت بین قوانین تمایز قایل شد. ‏شکل (3-16) معماری این سیستم را نشان می‌دهد.



معماری دو مرحله‌ای ارائه شده برای دسته‌بندی بسته در MC-SBC

مرحله اول در این سیستم، به صورت برون‌خط[[151]](#footnote-151) است. در این مرحله که بار محاسباتی سیستم را متحمل می‌شود، مجموعه قوانین به مجموعه‌های کوچک‌تر افراز شده، و نتایج افراز در جدول‌های جستجو که به صورت یک درخت تصمیم دو-سطحی پیاده‌سازی می‌گردند، ذخیره می‌شوند. در دسترسی به جدول‌های جستجو، از بیت‌های موثر به عنوان اندیس استفاده می‌شود. از این رو تنها تعداد کمی از بیت‌های موثر استفاده می‌شوند. MC-SBC از معیارهای نسبت wildcard[[152]](#footnote-152)، شاخص‌ استقلال[[153]](#footnote-153)، و شاخص تنوع[[154]](#footnote-154) برای تشخیص موقعیت بیت‌های موثر در یک مجموعه قوانین استفاده می‌کند. ‏شکل (3-17) مثالی از ساختار MC-SBC را نشان می‌دهد.

دسته‌بندی بسته به صورت برخط، خود شامل دو فرآیند است: فرآیند پیش-تصفیه[[155]](#footnote-155) و فرآیند تطابق کامل[[156]](#footnote-156). فرآیند پیش-تصفیه طوری طراحی شده که ID قانون‌های مرتبط با بسته ورودی را پیدا کند. در این قسمت از مقدار بیت‌های موثر در سرآیند یک بسته، به عنوان اندیس جستجو در ساختار درخت استفاده می‌شود. سپس نتایج اولیه برای به دست آوردن قانون‌های نامزد با هم ادغام می‌شوند. جهت شتاب دادن به فرآیند ادغام، شناسه قانون‌ها در جدول‌های جستجو به ترتیب ذخیره می‌شود. در نهایت ID قانون‌های نامزد به مرحله تطابق کامل ارسال می‌گردد.



مثالی از ساختار داده MC-SBC

در فرآیند تطابق کامل، بسته‌ها با قوانین نامزد تطبیق داده می‌شوند. در این گام مقادیر هر فیلد بر اساس مشخصات خاص خود، با هم مقایسه می‌شوند (برای مثال تطبیق پیشوندی، تطبیق محدوده‌ای، و یا تطبیق دقیق). در طول این فرآیند، اگر تطبیق یک فیلد با شکست روبرو شود، آن قانون کنار گذاشته شده و فوراً به سراغ قانون بعدی می‌رود. اگر بسته با قانونی با بیشترین اولویت مطابقت یابد، فرآیند خاتمه یافته و این قانون گزارش داده می‌شود. اگر هیچ تطابقی بین قوانین و بسته یافت نشود، عمل پیش‌فرض[[157]](#footnote-157) بازگردانده می‌شود.

گذردهی بخش برخط در MC-SBC

|  |  |
| --- | --- |
| گذردهی | اندازه مجموعه قوانین |
| 198 MPPS | 1K |
| 163 MPPS | 100K |

بخش برخط این معماری روی یک پردازنده گرافیکی پیاده‌سازی شده و نتایج مربوط به آن در ‏جدول (3-2) نشان داده شده است. نکته‌ای که باید به آن اشاره کرد، این است که در این کار فرض شده تاخیر دسته‌بندی بسته، تنها شامل تاخیر پردازش در تابع کرنل در GPU است و از تاخیر انتقال داده بین CPU و GPU صرف نظر شده. بنابراین نتایج نشان داده شده در ‏جدول (3-2) با این فرض محاسبه شده‌اند.

* 1. جمع‌بندی

راهکارهای ارائه شده برای حل مسئله دسته‌بندی بسته در سوئیچ SDN را از آن جهت که از ویژگی‌های ارائه شده توسط SDN استفاده کرده‌اند یا خیر، می‌توان به دو دسته تقسیم بندی نمود. در راهکارهایی که از ویژگی‌های SDN بهره برده شده است، عموماً سعی شده از وجود کنترلر و قابلیت برنامه‌ریزی شبکه توسط آن استفاده شود. در این دسته از کارها با انتقال بار پردازشی به لبه شبکه کارآیی سوئیچ‌های هسته به طور قابل توجهی بهبود می‌یابد. با این حال چالش دسته‌بندی بسته در سوئیچ‌های لبه هم‌چنان وجود دارد.

در روش‌های الگوریتمی سعی می‌شود فضای نمونه نهایی برای مطابقت دادن فیلدهای سرآیند یک بسته با قوانین به حداقل برسد. در این روش‌ها عموماً از ساختار داده‌های مبتنی بر درخت، و جدول‌های درهم‌سازی استفاده می‌شود. از طرفی برای افزایش گذردهی کلی سیستم، پردازنده‌های گرافیکی به عنوان سکوی پردازشی برای عمل دسته‌بندی، به شدت مورد توجه محققین قرار گرفته‌اند. برای استفاده از الگوریتم‌های دسته‌بندی بر روی این پردازنده‌ها باید ملاحضاتی را در نظر گرفت. برای مثال افزایش اندازه عمق درخت به دلیل ایجاد مسئله واگرایی پرش باعث افت کارآیی در این پردازنده‌ها می‌شود، و یا به دلیل محدودیت اندازه حافظه، اندازه درخت خیلی نمی‌تواند بزرگ باشد.

الگوریتم MC-SBC یک نمونه خوب از طراحی و پیاده‌سازی یک روش دسته‌بندی مبتنی بر درخت بر روی پردازنده گرافیکی است که بنابر ادعای ارائه کنندگان آن، کارآیی بهتری نسبت به دیگر روش‌های مبتنی بر درخت دارد. با این حال با افزایش نسبت Wildcard در مجموعه قوانین نسبت تکرار در ساختار ارائه شده و تعداد قوانین در برگ‌های درخت بسیار افزایش می‌یابد. در فصل بعدی با بهره‌گیری از الگوریتم ژنتیک سعی می‌کنیم ساختار داده MC-SBC را به شکل بهتری ایجاد کنیم. به طوری که نسبت تکرار، و تعداد قوانین در برگ‌ها کاهش یافته و در نتیجه استفاده از حافظه بهینه‌تر و سرعت عمل دسته‌بندی بیشتر شود.

1. فصل4: استفاده از الگوریتم ژنتیک برای پیدا کردن بیت‌های موثر

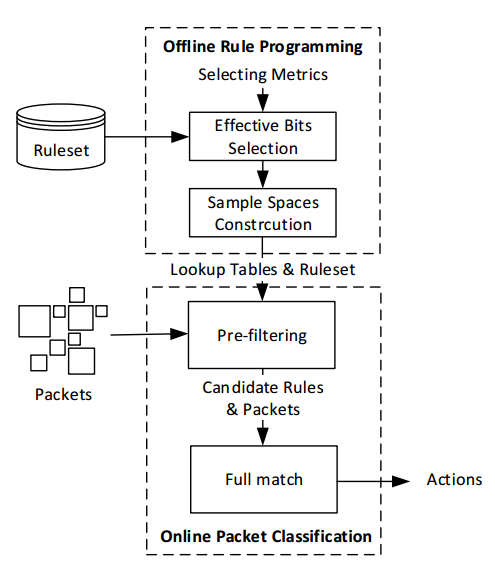
در بخش ‏3-4-5- به معماری MC-SBC اشاره کرده، و دیدیم که ارائه یک سیستم دو مرحله‌ای برای مسئله دسته‌بندی بسته، به چه صورت بار محاسباتی را به مرحله برون‌خط، برای ایجاد جدول‌های جستجو برده و بخش بر‌خط که در آن جستجوی بسته‌ها انجام می‌گیرد، تسریع یافته است.

در مرحله‌ی برون‌خط MC-SBC، از یک روش آماری جهت یافتن بیت‌های موثر برای افراز مجموعه قوانین استفاده شده است. یک افراز خوب، افرازی است که در آن اولاً اندازه زیرمجموعه‌ها در حد امکان با هم برابر، و ثانیاً نسبت تکرار قوانین پایین باشد. این امر نسبت مستقیمی با کارآیی سیستم در بخش بر‌خط خواهد داشت.

در این فصل ابتدا جزئیات MC-SBC را با دقت بررسی و سپس از الگوریتم ژنتیک برای یافتن بیت‌های موثر استفاده کرده‌ایم. نتایج گرفته شده نشان می‌دهد وقتی نسبت wildcard در یک مجموعه قوانین بالا باشد، این الگوریتم به صورت بهتری می‌تواند آن را افراز کند. هم چنین وقتی اندازه مجموعه قوانین بزرگ باشد، روش پیشنهادی تاثیر قابل توجهی در بخش برخط خواهد داشت.

* 1. بررسی MC-SBC‌[[158]](#footnote-158) با جزئیات بیشتر

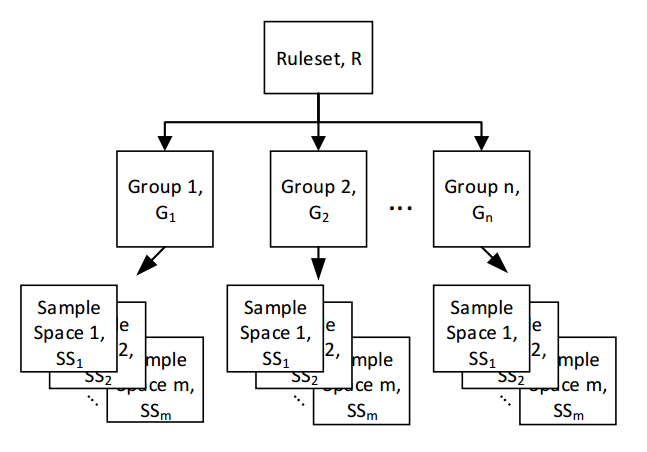
در [14] یک سیستم با معماری دو مرحله‌ای برای دسته‌بندی بسته ارائه شده است. در این معماری از جدول‌های جستجوی ساده برای جلوگیری از مسئله انشعاب محاسباتی استفاده شده، و در سکوهای محاسباتی از دستورات با کارآیی بالا برای گذردهی بالاتر سیستم، بهره برده شده است. به دلیل خلوتی[[159]](#footnote-159) و توزیع جهت‌دار[[160]](#footnote-160) قوانین در مجموعه قوانین، MC-SBC به گونه‌ای طراحی شده که با استفاده از بیت‌های موثر، به سرعت قوانین نامزد کمتری برای تطابق کامل، انتخاب کند. ‏شکل (4-1) معماری این سیستم را نشان می‌دهد.



معماری دو مرحله‌ای ارائه شده برای دسته‌بندی بسته در MC-SBC

ایده اصلی این کار از روش HiCut [17] گرفته شده که در آن یک درخت تصمیم به صورت هوشمندانه به صورتی ایجاد می‌شود که تعداد کمی قانون در برگ‌های درخت وجود داشته باشد. در این جا اولاً به دلیل استفاده از پردازنده‌های گرافیکی برای جستجوی بسته‌ها، یک درخت تصمیم دو-سطحی ایجاد می‌شود تا تاثیر مشکل واگرایی پرش در این پردازنده‌ها کاهش یابد. ثانیاً از مفهوم بیت‌های موثر جهت افراز مجموعه قوانین و ایجاد درخت استفاده می‌شود. بیت‌های موثر، بیت‌هایی هستند که با استفاده از آن‌ها می‌توان به بهترین صورت بین قوانین تمایز قایل شد. بنابر ادعای صورت گرفته در [14] این الگوریتم نسبت به دیگر روش‌های مبتنی بر درخت تصمیم مانند BC[15]، SBC[16]، HiCuts[17]، و HyperCuts[18] کارآیی بهتری دارد.

‏شکل (4-2) مثالی از ساختار داده این سیستم را نشان می‌دهد. در این ساختار به هر گره در سطح اول درخت یک گروه[[161]](#footnote-161) یا زیرمجموعه[[162]](#footnote-162) گفته می‌شود. سطح دوم درخت در هر گروه یک جدول جستجو یا فضای نمونه[[163]](#footnote-163) نامیده می‌شود. در این جا برای افزایش کارآیی چندین فضای نمونه در هر گروه ایجاد می‌شود.

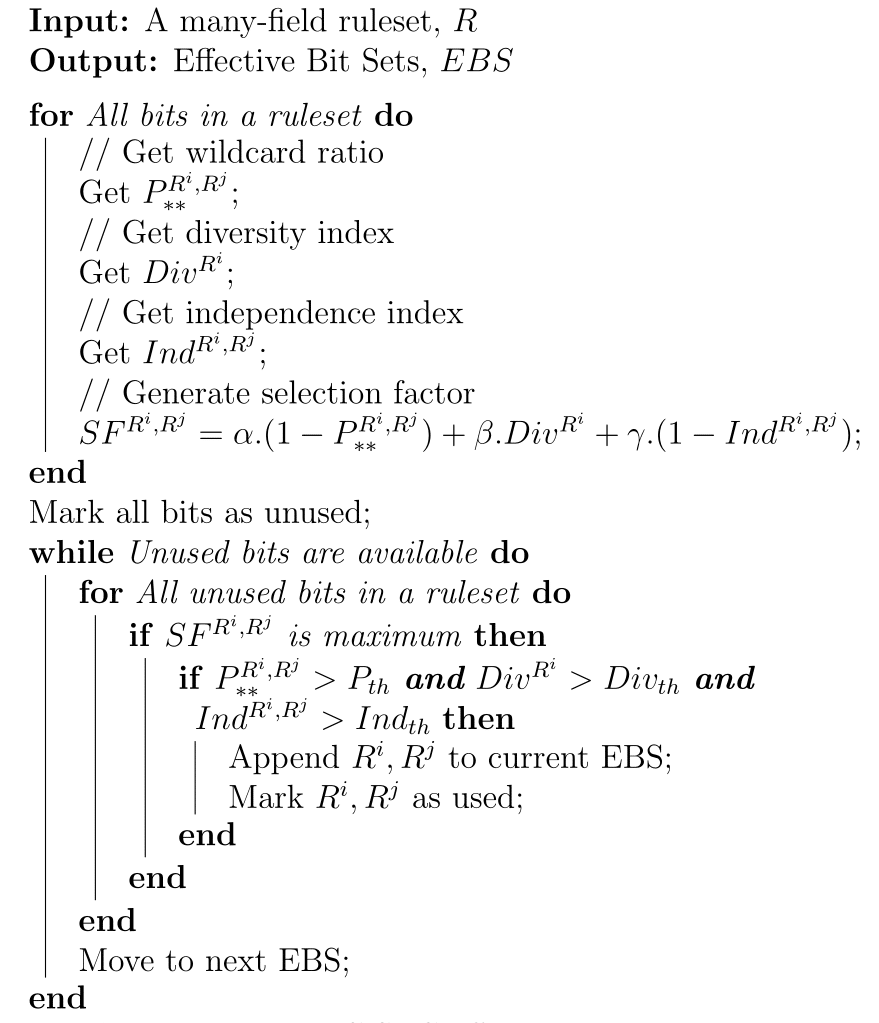


مثالی از ساختار داده MC-SBC

* + 1. برنامه‌ریزی قانون[[164]](#footnote-164) به صورت برون‌خط

این مرحله هزینه محاسباتی فرآیند بسته‌بندی بسته‌ها را با پیش‌محاسبه جدول‌های جستجو متحمل می‌شود. در این گام مجموعه قوانین به مجموعه‌های کوچک‌تر افراز شده، و نتایج افراز در جدول‌های جستجو به صورت یک درخت تصمیم ذخیره می‌شوند. در دسترسی به جدول‌های جستجو، از بیت‌های موثر به عنوان اندیس استفاده می‌شود. MC-SBC از معیارهای نسبت wildcard[[165]](#footnote-165)، شاخص‌ استقلال[[166]](#footnote-166)، و شاخص تنوع[[167]](#footnote-167) برای تشخیص موقعیت بیت‌های موثر در یک مجموعه قوانین استفاده می‌کند.

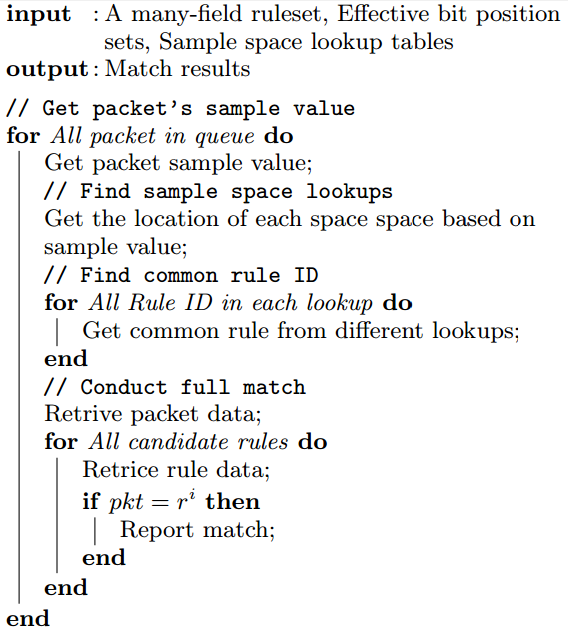
شبه کد فرآیند برنامه‌ریزی قانون، در الگوریتم 3 نشان داده شده است، به طوری که در آن R مجموعه قوانین، مقدار بیت i-ام، و فاکتور انتخاب برای بیت‌های i-ام، و j-ام می‌باشند. نسبت wildcard بیت‌های i-ام و j-ام و مقدار آستانه این فاکتور است. شاخص تنوع در بیت i-ام با مقدار آستانه است. و در نهایت شاخص استقلال بیت‌های i-ام، و j-ام با مقدار آستانه می‌باشد



الگوریتم 3 شبه کد الگوریتم برنامه‌ریزی قانون در MC-SBC

* + 1. فرآیند دسته‌بندی بسته به صورت برخط

دسته‌بندی بسته به صورت برخط، خود شامل دو فرآیند است: فرآیند پیش-تصفیه[[168]](#footnote-168) و فرآیند تطابق کامل[[169]](#footnote-169). شبه کد مربوطه در الگوریتم 4 آمده است، به طوری که در آن r قانون هدف[[170]](#footnote-170)، و pkt بسته ورودی است. فرآیند پیش-تصفیه طوری طراحی شده که ID قانون‌های مرتبط را برای بسته ورودی پیدا کند. در این‌جا از مقدار بیت‌های موثر در یک بسته، به عنوان اندیس جستجو استفاده شده، و جدول‌های جستجو برای به دست آوردن نتایج اولیه مورد بررسی قرار می‌گیرند. سپس این نتایج اولیه برای به دست آوردن قانون‌های نامزد با هم ادغام می‌شوند. جهت شتاب دادن به فرآیند ادغام، شناسه قانون‌ها در جدول‌های جستجو به ترتیب ذخیره می‌شود. در نهایت ID قانون‌های نامزد به مرحله تطابق کامل ارسال می‌گردد.



الگوریتم 4 شبه کد مرحله بر خط در MC-SBC

در فرآیند تطابق کامل، بسته‌ها با قوانین نامزد تطبیق داده می‌شوند. در این گام مقادیر هر فیلد بر اساس مشخصات خاص خود، با هم مقایسه می‌شوند (برای مثال تطبیق پیشوندی، تطبیق محدوده‌ای، و یا تطبیق دقیق). در طول این فرآیند، اگر تطبیق یک فیلد با شکست روبرو شود، آن قانون کنار گذاشته شده و فوراً به سراغ قانون بعدی می‌رود. اگر بسته با قانونی با بیشترین اولویت مطابقت یابد، فرآیند خاتمه یافته و این قانون گزارش داده می‌شود. اگر هیچ تطابقی بین قوانین و بسته یافت نشود، عمل پیش‌فرض[[171]](#footnote-171) بازگردانده می‌شود.

* + 1. معیارهای انتخاب بیت‌های موثر

بیت‌های موثر، بیت‌هایی هستند که تمای سعی خود را می‌کنند که یک مجموعه قوانین را به صورت متعادل افراز کنند. در فرآیند برنامه‌ریزی قانون، MC-SBC از این بیت‌ها برای برش[[172]](#footnote-172) مجموعه قوانین به زیرمجموعه‌ها[[173]](#footnote-173) استفاده می‌کند، تا جدول‌های جستجویی را به عنوان خروجی تولید کند. در فرآیند دسته‌بندی برخط بسته‌ها، از مقدار بیت‌های موثر بسته، برای یافتن قوانین نامزد استفاده می‌شود. از این رو هدف MC-SBC یافتن زیرمجموعه‌هایی است که حتی الامکان اندازه کوچکی داشته باشند و قوانین مختلف را به صورت کارا از هم جدا کنند. بنابراین در یک مجموعه بیت‌های موثر، هر دو بیت از آن ها باید به گونه‌ای باشند، که دارای نسبت wildcard کمتر، و تنوع بیشتری باشند، تا تعداد قوانین تکراری کاهش یابد. هم چنین، شاخص استقلال برای جلوگیری از تولید ساختمان داده جهت‌دار مورد استفاده قرار گرفته. در ادامه مبانی نظری برای محاسبه این شاخص‌ها را آورده‌ایم.

* نسبت Wildcard

نسبت Wildcard مقدار نمادهای wildcard (\*) موجود در دو بیت انتخاب شده را مشخص می‌کند. وقتی چنین نمادی در یک قانون وجود داشته باشد، این نماد باید به 0 و 1 تبدیل شود تا تمامی حالت‌های امکان‌پذیر مختلف را پوشش دهد. به همین دلیل باعث ایجاد تکرار[[174]](#footnote-174) می‌شود. اگر تعداد نمادهای wildcard در بیت‌های موثر زیاد باشد، آن قانون به تعداد زیادی تکرار می‌شود. نسبت wildcard بالاتر، به معنی افزایش تکرار است.

اگر سایر پارامترها یکسان باشند، MC-SBC بیت‌هایی را انتخاب می‌کند که نسبت wildcard پایین‌تری دارند. برای این منظور از رابطه (5-1) برای تخمین نسبت wildcard ترکیبی برای هر دو بیت انتخاب شده استفاده کرده است، به طوری که در آن N تعداد قوانین، و i و j موقعیت دو بیت انتخاب شده هستند.

|  |  |
| --- | --- |
| (5-1) |  |

* شاخص استقلال

بیت‌هایی که هم‌بستگی کمتری با هم داشته باشند، بهتر می‌توانند بین قوانین تمایز قائل شوند. شاخص استقلال این اطمینان را می‌دهد که بیت‌های مختلف نسبت به هم استقلال زیادی داشته باشند. MC-SBC یک مجموعه بیت موثر را به نحوی انتخاب می‌کند، که استقلال بالایی بین آن‌ها وجود داشته باشد. برای تعیین مقدار این شاخص رابطه (5-2) رای هر دو بیت محاسبه می‌شود.

|  |  |
| --- | --- |
| (5-2) |  |

* شاخص تنوع

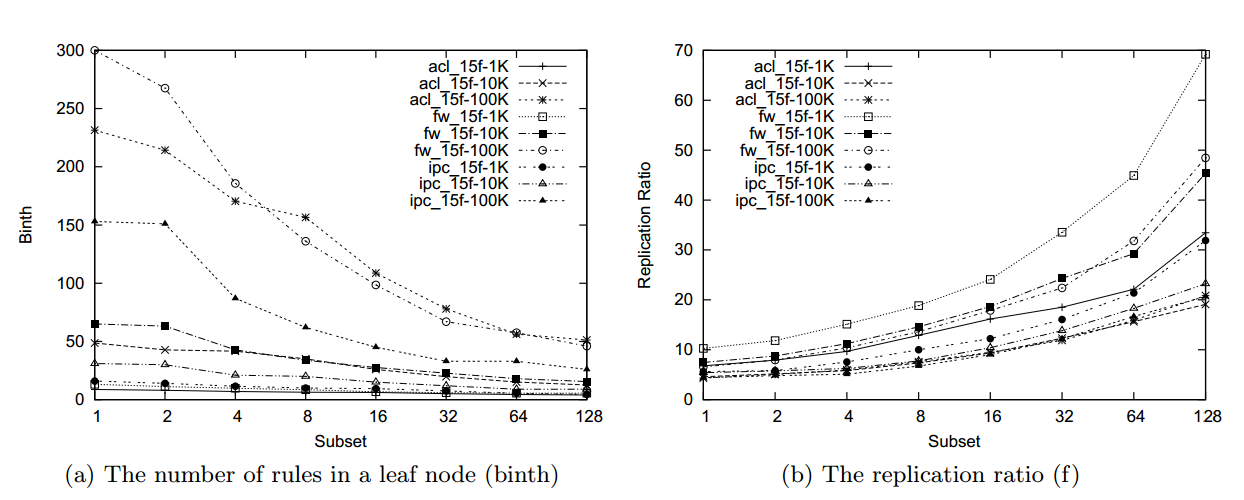
شاخص تنوع این اطمینان را به ما می‌دهد که تعداد قوانین در زیرمجموعه‌ها با هم برابر باشند. در نتیجه کارآیی بهتر و تاخیر پردازشی کمتر می‌شود. شاخص تنوع برای یک بیت با محاسبه بی‌نظمی توزیع صفر و یک‌ها طبق رابطه (5-3) به دست می‌آید. با افزایش شاخص تنوع، اندازه زیرمجموعه‌ها به طور یکسانی توزیع می‌شود، و اگر اندازه زیرمجموعه ها توزیع یکنواختی داشته باشد، تاخیر پردازشی تحت بدترین شرایط، بسیار کاهش می‌یابد.

|  |  |
| --- | --- |
| (5-3) |  |

شاخص‌های فوق را می‌توان برای تعداد بیت‌های بیشتر هم محاسبه کرد. در این صورت الگوریتم پیشنهادی با افزایش هزینه محاسباتی، دقت بیشتری خواهد داشت. در [14] از این شاخص‌ها برای یک یا دو بیت استفاده شده تا پیچیدگی محاسباتی کاهش یابد.

* + 1. نتایج MC-SBC

در [14] برای ارزیابی بخش برون‌خط، دو معیار ارائه شده است. معیار اول نسبت تکرار[[175]](#footnote-175) است که بیانگر درصد تکرار قوانین بعد از ایجاد جدول‌ها می‌باشد. معیار دوم بیشینه تعداد قانون در یک گره برگ (یا در واقع یک مدخل از جدول جستجو) است، که Binth نامیده شده است. پایین بودن Binth نشان‌دهنده این است، که الگوریتم به صورت خوبی توانسته قوانین را بین مدخل‌های جریان توزیع کند.



نتایج بخش برون‌خط MC-SBC

در ‏شکل (4-3) مقدار این فاکتورها برای 9 مجموعه داده 15 فیلدی نشان داده شده است. در این جا اندازه هر فضای نمونه 15 بیت در نظر گرفته شده که ثابت است، و اندازه گروه از 1 تا 128 (معادل 0 تا 7 بیت) تغییر داده شده. بخش a مقدار Binth را نشان می‌دهد. همان طور که مشخص است، با افزایش تعداد گروه‌ها مقدار این پارامتر کاهش یافته است. چرا که در این صورت، تعداد قانون کمتری در هر گروه قرار می‌گیرد. در بخش b نتایج نسبت تکرار آورده شده. نسبت تکرار با افزایش تعداد گروه‌ها، افزایش یافته است. این به دلیل آن است که تعداد بیت‌های موثر و در نتیجه تعداد بیت‌هایی با مقدار \* بیشتر شده است.

بخش برخط این معماری روی یک پردازنده گرافیکی پیاده‌سازی شده و نتایج مربوط به آن در ‏جدول (3-2) نشان داده شده است. نکته‌ای که باید به آن اشاره کرد، این است که در این کار فرض شده تاخیر دسته‌بندی بسته، تنها شامل تاخیر پردازش در تابع کرنل در GPU است و از تاخیر انتقال داده بین CPU و GPU صرف نظر شده. بنابراین نتایج نشان داده شده در ‏جدول (3-2) با این فرض محاسبه شده‌اند.

گذردهی بخش برخط در MC-SBC

|  |  |
| --- | --- |
| گذردهی | اندازه مجموعه قوانین |
| 198 MPPS | 1K |
| 163 MPPS | 100K |

* 1. استفاده از الگوریتم ژنتیک برای یافتن بیت‌های موثر

در بخش ‏2-5- الگوریتم ژنتیک و گام‌‌های مختلف آن را به طور مختصر توضیح دادیم. در این بخش نحوه استفاده از این الگوریتم برای یافتن مجموعه بیت‌های موثر جهت افراز مجموعه قوانین را شرح می‌دهیم.

برای آن که یک مجموعه بیت موثر به طول n، برای افراز مجموعه قوانین به زیرمجموعه پیدا کنیم، فرض می‌کنیم هر فرد (یا کروموزوم) در الگوریتم ژنتیک یک مجموعه بیت موثر باشد، که در آن هر ژن نشان دهنده یک موقعیت بیتی است. در ‏شکل (4-4) یک کروموزوم نمونه به طول 5 نشان داده شده است. با استفاده از این کروموزوم بیت‌های 12، 254، 127، 2، و 19 برای افراز مجموعه قوانین به 32 زیرمجموعه استفاده می‌شوند. در ادامه گام‌های مختلف الگوریتم را توضیح می‌دهیم.

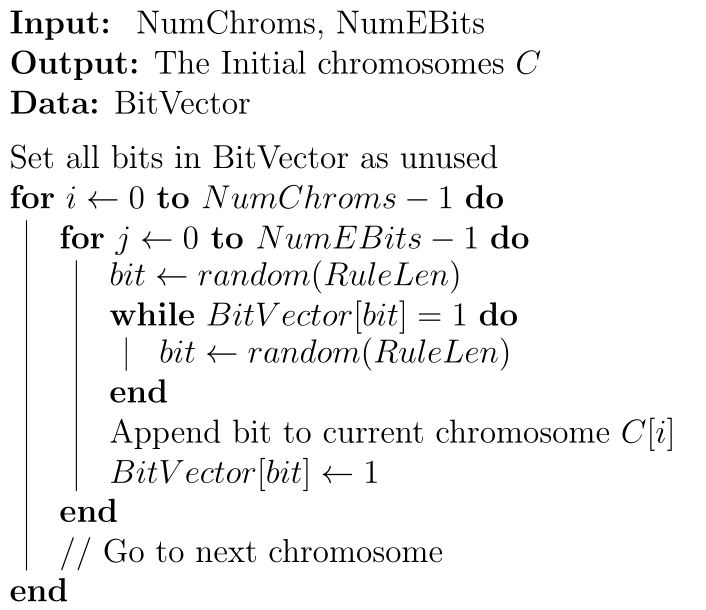


یک نمونه کروموزوم شامل بیت‌های موثر

* + 1. مقداردهی اولیه

جمعیت اولیه کروموزوم‌ها به صورت تصادفی تولید می‌شود. طول کروموزوم‌ها (تعداد بیت‌های موثر) با توجه به مقدار تعیین شده توسط کاربر به عنوان ورودی به الگوریتم داده شده، و با توجه به آن، تعداد کروموزوم‌ها را طبق رابطه (5-4) محاسبه می‌کنیم. در این رابطه تعداد بیت‌های هر قانون در مجموعه قوانین است (در کار ما با توجه به فیلدهای موجود در ‏جدول (2-4) طول این متغیر برابر با 356 می‌باشد). از طرفی در تولید کروموزوم‌ها دقت می‌شود که بیت تکراری در آن‌ها وجود نداشته باشد. به این ترتیب جمعیت اولیه شامل حداقل نیمی از بیت‌ها خواهد بود.

|  |  |
| --- | --- |
| (5-4) |  |



الگوریتم 5 شبه کد گام مقداردهی اولیه در الگوریتم ژنتیک

* + 1. انتخاب

در مرحله انتخاب باید کروموزوم‌ها را مورد ارزیابی قرار داده و بر اساس کیفیت آن‌ها تعدادی را برای گام بعدی، که بازتولید جمعیت است، انتخاب کنیم. ما به دنبال یافتن کروموزومی هستیم که مجموعه قوانین را با در نظر داشتن دو معیار به بهترین شکل افراز می‌کند. معیار اول اندازه زیرمجموعه‌ها است. افرازی که در آن اندازه زیرمجموعه‌ها با هم برابر باشند در گام برخط برای جستجوی بسته‌ها نتیجه بهتری به همراه خواهد داشت. بنابراین کروموزومی خوب است، که در آن اندازه زیر مجموعه‌ها در حد امکان با هم یکسان و یا به هم نزدیک باشند. معیار دوم نسبت تکرار است. نسبت تکرار به نوعی افزونگی را در خروجی الگوریتم نشان می‌دهد. هر چه این نسبت پایین باشد، یعنی قانون‌های کمتری تکرار شده و در نتیجه مقدار حافظه مصرفی کمتر خواهد شد.

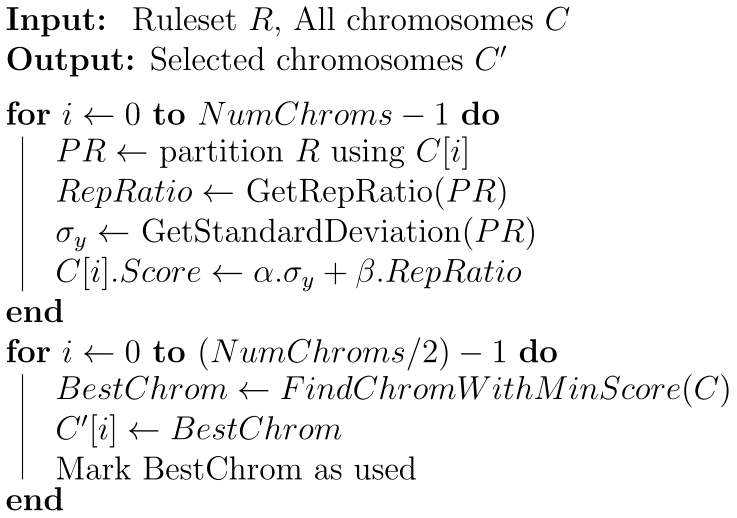
فرض کنید X یک متغیر تصادفی باشد که تعداد قوانین قرار گرفته در هر زیرمجموعه را پس از افراز توسط کروموزوم C، نشان می‌دهد. در این صورت متغیر تصادفی Y و متغیر RepRatio را به صورت رابطه‌های (5-5)، و (5-6) تعریف می‌کنیم. در این روابط N تعداد قوانین موجود در مجموعه قوانین می‌باشد. متغیر تصادفی Y کسری از قوانین را نشان می‌دهد که در هر زیر مجموعه قرار گرفته، و متغیر RepRatio نشان‌دهنده نسبت تکرار است.

|  |  |
| --- | --- |
| (5-5) |  |
| (5-6) |  |

با استفاده از روابط (5-5)، و (5-6) تابع برازش[[176]](#footnote-176) را به صورت رابطه (5-7) تعریف می‌کنیم. در این رابطه انحراف معیار متغیر تصادفی Y است، و نشان‌دهنده پرداکندگی اندازه زیرمجموعه‌ها می‌باشد. ، و ، ضرایب ثابتی هستند که حساسیت الگوریتم را نسبت به هر معیار تغییر می‌دهند. اگر بزرگتر باشد، الگوریتم نسبت به متعادل بودن اندازه زیرمجموعه‌ها حساس خواهد بود. از طرفی با افزایش مقدار کروموزوم‌هایی انتخاب می‌شوند که مجموعه قوانین را با نسبت تکرار کمتری افراز می‌کنند.

|  |  |
| --- | --- |
| (5-7) |  |

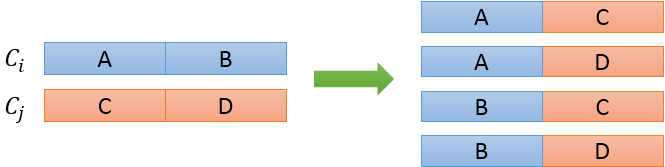
با استفاده از تابع f در رابطه (5-7)، در هر تکرار الگوریتم ژنتیک، نیمی از کروموزوم‌هایی که مقدار تابع برازش آن‌ها کمتر است را برای بازتولید جمعیت انتخاب می‌کنیم.



الگوریتم 6 شبه کد گام انتخاب در الگوریتم ژنتیک

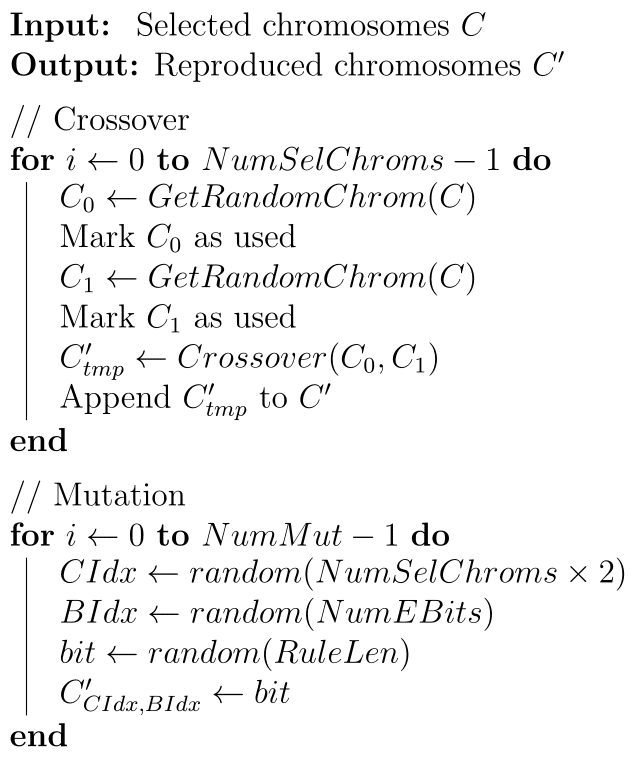
* + 1. تقاطع و جهش

در گام بازتولید با اعمال تقاطع[[177]](#footnote-177) و جهش[[178]](#footnote-178) روی کروموزوم‌های منتخب، جمعیت نسل بعدی تولید می‌شود. در بخش ‏4-2-3- نیمی از کروموزوم‌های موجود را با استفاده از تابع برازش برای بازتولید انتخاب کردیم. در این گام ابتدا با انجام عمل تقاطع روی هر دو کروموزوم، چهار کروموزوم جدید تولید، و با استفاده از جهش سعی می‌کنیم تنوع را در جمعیت نسل بعدی حفظ کنیم.



انجام عمل تقاطع روی دو کروموزوم منتخب و

فرض کنید کروموزوم‌های ، و دو کروموزوم انتخاب شده باشند. هم‌چنین فرض کنید کروموزوم از دو بخش A، و B، و کروموزوم از دو بخش C، و D تشکیل شده‌اند. عمل تقاطع به صورت ‏شکل (4-5) روی این دو کروموزوم اعمال شده و چهار کروموزوم جدید به عنوان خروجی این عمل ایجاد می‌شوند.



الگوریتم 7 شبه کد گام بازتولید در الگوریتم ژنتیک

با انجام عمل تقاطع تعداد کروموزوم‌ها به اندازه مقدار ابتدایی خود خواهد شد. پس از انجام این عمل برای حفظ تنوع در جمعیت نسل بعدی، و نیز جهت خروج جواب از نقاط کمینه[[179]](#footnote-179) یا بیشینه[[180]](#footnote-180) محلی عمل جهش روی برخی کروموزوم‌ها انجام می‌شود. در واقع تعدادی از ژن‌های برخی کروموزوم‌ها به صورت تصادفی انتخاب شده، و مقدار آن‌ها با مقدار جدیدی که آن هم به صورت تصادفی تولید می‌شود، جایگزین می‌گردد. تعداد ژن‌هایی که عمل جهش روی آن‌ها انجام می‌شود، طبق رابطه (5-8) بر اساس تعداد کروموزوم‌ها، و تعداد بیت‌های موثر (ژن) در هر کروموزوم، مشخص می‌گردد. در این رابطه ضریب ثابتی است که با آن تعداد جهش قابل تنظیم است.

|  |  |
| --- | --- |
| (5-8) |  |

* + 1. شرط خاتمه الگوریتم

شرط خاتمه الگوریتم را تعداد مشخصی از تکرار آن در نظر می‌گیریم. به عبارت دیگر وقتی الگوریتم به تعداد K بار تکرار شود، الگوریتم را خاتمه می‌دهیم. برای آن که مقدار K را تعیین کنیم، ابتدا تعریفی از یک کروموزوم خوب ارائه می‌دهیم. کروموزوم خوب، کروموزومی با بهترین مقدار تابع برازش است، که بعد از M تکرار پس از انتخاب شدن، همچنان بهترین باقی بماند. سپس الگوریتم را با تعداد تکرار بی‌نهایت اجرا کرده و شماره تکراری که بهترین کروموزوم در آن انتخاب می‌شود، را ثبت می‌کنیم. به این ترتیب می‌توانیم تقریب خوبی از تعداد تکرار مناسب برای اجرای الگوریتم را به دست آوریم.

* 1. نتایج بخش برون‌خط

در این بخش نتایج مربوط به اجرای الگوریتم برون‌خط MC-SBC و الگوریتم ژنتیک آورده شده است. برای ارزیابی از مجموعه قوانینی که در بخش ‏2-6- تولید شد، استفاده شده است.

* + 1. تعیین تعداد تکرار الگوریتم ژنتیک

نحوه مشخص کردن تعداد تکرار الگوریتم در بخش ‏4-2-5- شرح داده شد. ‏شکل (4-6) شماره تکرار انتخاب شدن بهترین کروموزوم را برای تعداد بیت‌های موثر 1 تا 15 نشان می‌دهد. در این جا M برابر با 20 در نظر گرفته شده، و الگوریتم برای 3 مجموعه داده با 1000 قانون و نسبت Wildcard 0.1 اجرا شده است. همان طور که مشاهده می‌شود در همه حالات بهترین کروموزوم بین تکرار 30-ام تا 60-ام مشخص شده است. در ادامه کار مقدار K را برابر با 100 در نظر گرفته‌ایم، تا حاشیه امنیت خوبی داشته باشیم.

شماره تکرار مشخص شدن بهترین کروموزوم برای تعداد بیت‌های موثر مختلف

* + 1. مقایسه دو الگوریتم از لحاظ نسبت تکرار و Binth

در این بخش دو الگوریتم را از نظر دو معیار نسبت تکرار و Binth با هم مقایسه کرده‌ایم. جهت انجام این کار اندازه جدول جستجو برابر با 10 بیت (1024 مدخل) برای هر زیر مجموعه در نظر گرفته شده است. هم چنین از آن جایی که الگوریتم MC-SBC جواب را به صورت قطعی تعیین می‌کند، و الگوریتم ژنتیک ذات تصادفی داشته و نتیجه آن ممکن است در اجراهای مختلف با هم تفاوت داشته باشد، برای جمع‌آوری نتایج، MC-SBC را یک بار و ژنتیک را چندین بار اجرا کرده و میانگین نتایج به دست آمده را گزارش کرده‌ایم.

* مقایسه دو روش با تغییر نسبت Wildcard

‏شکل (4-7) تا ‏شکل (4-9) نمودار نسبت تکرار و Binth را برای سه مجموعه قوانین IPC، ACL، و FW با اندازه 10000 قانون، نشان می‌دهند. در این نمودارها اندازه زیرمجموعه 4 بیت (16 زیرمجموعه) و اندازه جدول‌های جستجو برابر با 10 بیت (1024 مدخل) در نظر گرفته شده است. در هر یک از این نمودارها نسبت Wildcard را از 0 تا 5 تغییر داده‌ایم. همان طور که مشاهده می‌شود، وقتی نسبت Wildcard پایین است (0 و 0.1)، خروجی دو الگوریتم تقریباً مانند هم است. وقتی نسبت Wildcard بیشتر می‌شود، الگوریتم ژنتیک، هم از لحاظ نسبت تکرار و هم از لحاظ Binth رفتار بهتری داشته است.

نمودار نسبت تکرار و Binth برای مجموعه قوانین IPC با نسبت Wildcard مختلف

نمودار نسبت تکرار و Binth برای مجموعه قوانین ACL با نسبت Wildcard مختلف

نمودار نسبت تکرار و Binth برای مجموعه قوانین FW با نسبت Wildcard مختلف

* مقایسه دو روش با تغییر تعداد زیرمجموعه‌ها

در قسمت قبل دیدیم که وقتی نسبت Wildcard، کم است دو روش تفاوت چندانی ندارند. اما با افزایش نسبت Wildcard الگوریتم ژنتیک رفتار بهتری دارد. بنابراین در این قسمت مقایسه را با نسبت Wildcard 1 درصد انجام داده‌ایم. از طرفی اندازه جدول‌های جستجو را برابر با مقدار ثابت 10 بیت (1024 مدخل) در نظر گرفته و تعداد زیرمجموعه‌ها را از 1 (0 بیت) تا 128 (7 بیت) تغییر داده‌ایم.

نمودار نسبت تکرار و Binth برای مجموعه قوانین IPC با تغییر تعداد زیرمجموعه‌ها

‏شکل (4-10) تا ‏شکل (4-12) نمودار نسبت تکرار و Binth را برای سه مجموعه قوانین IPC، ACL، و FW با اندازه 10000 قانون، نشان می‌دهند. نسبت تکرار در الگوریتم ژنتیک برای هر سه مجموعه قوانین و هر تعداد زیرمجموعه، بهبود داشته است. در مجموعه قوانین ACL، نسبت تکرار در ژنتیک رفتار بسیار بهتری از MC-SBC داشته است. در مجموعه قوانین IPC، بهبود نسبت تکرار، نسبت به دو مجموعه دیگر کمتر است. Binth در الگوریتم ژنتیک هنگامی که تعداد زیرمجموعه‌ها کم است، نسبت به زمانی که تعداد زیرمجموعه‌ها بیشتر می‌شود، بهبود قابل توجهی دارد.

نمودار نسبت تکرار و Binth برای مجموعه قوانین ACL با تغییر تعداد زیرمجموعه‌ها

نمودار نسبت تکرار و Binth برای مجموعه قوانین FW با تغییر تعداد زیرمجموعه‌ها

* کمینه، میانگین، و بیشنیه تعداد قوانین در برگ‌ها

‏جدول (4-2) تا ‏0مقادیر کمینه، میانگین، و بیشینه تعداد قوانین در برگ‌های درخت را به ترتیب برای مجموعه قوانین ACL، FW، و IPC با اندازه‌های مختلف نشان می‌دهند. در این بخش تعداد گروه‌ها 8 (معادل 3 بیت) و تعداد برگ‌ها در هر گروه 256 (معادل 8 بیت) قرار داده شده است.

در مجموعه قوانین ACL الگوریتم‌ ژنتیک و الگوریتم برون‌خط MC-SBC تقریبا رفتار مشابه‌ای دارند. در این مجموعه قوانین در برخی از موارد الگوریتم ژنتیک خروجی بهتری دارد و در برخی الگوریتم MC-SBC، با این حال اختلاف زیادی بین آن‌ها وجود ندارد. ‏شکل (4-13) نمودار مقادیر کمینه، میانگین، و بیشینه تعداد قوانین در برگ‌ها را برای مجموعه قانون IPC\_100k در نسبت‌های Wildcard مختلف است. رفتار تقریباً یکسان دو الگوریتم برای این مجموعه قوانین در این نمودار نیز قابل مشاهده است.

در مجموعه قوانین FW، و IPC، الگوریتم ژنتیک خروجی بهتری تولید کرده است. مخصوصاً زمانی که اندازه مجموعه قوانین بزرگ‌تر و نسبت Wildcard بیشتر می‌شود. ‏شکل (4-14) و ‏شکل (4-15) نمودار مقادیر کمینه، میانگین، و بیشینه تعداد قوانین در برگ‌ها را به ترتیب برای مجموعه قوانین FW\_100k، و IPC\_100k نشان می‌دهند. بهبود کیفیت افراز توسط الگوریتم ژنتیک همان‌طور که در بخش ‏4-4- آمده است، باعث بهبود کارآیی در بخش برخط می‌شود.

کمینه، میانگین، و بیشینه تعداد قوانین در برگ‌ها برای مجموعه قوانین ACL

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Wildcard Ratio | Alg. | acl\_1k | | | acl\_10k | | | acl\_100k | | |
| min | avg | max | min | avg | max | min | avg | max |
| 0.1 | MC-SBC | 0 | 0.45 | 5 | 0 | 4.87 | 15 | 28 | 49.7 | 79 |
| Genetic | 0 | 0.45 | 4 | 0 | 4.89 | 13 | 30 | 49.8 | 80 |
| 1 | MC-SBC | 0 | 0.51 | 5 | 0 | 5.43 | 28 | 32 | 57.4 | 94 |
| Genetic | 0 | 0.51 | 4 | 0 | 5.3 | 14 | 10 | 54.4 | 113 |
| 5 | MC-SBC | 0 | 1.1 | 16 | 0 | 7.32 | 29 | 13 | 67.9 | 143 |
| Genetic | 0 | 0.77 | 5 | 0 | 6.42 | 33 | 16 | 70.2 | 140 |

کمینه، میانگین، و بیشینه تعداد قوانین در برگ‌ها برای مجموعه قوانین FW

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Wildcard Ratio | Alg. | fw\_1k | | | fw\_10k | | | fw\_100k | | |
| min | avg | max | min | avg | max | min | avg | max |
| 0.1 | MC-SBC | 0 | 0.4 | 4 | 0 | 4.87 | 14 | 26 | 48.8 | 72 |
| Genetic | 0 | 0.4 | 4 | 0 | 4.86 | 13 | 26 | 47.8 | 71 |
| 1 | MC-SBC | 0 | 0.51 | 5 | 0 | 5.91 | 16 | 37 | 66.8 | 99 |
| Genetic | 0 | 0.47 | 4 | 0 | 5.45 | 17 | 23 | 54 | 80 |
| 5 | MC-SBC | 0 | 1.1 | 9 | 0 | 9.9 | 28 | 7 | 109 | 267 |
| Genetic | 0 | 0.71 | 6 | 1 | 8.67 | 23 | 59 | 91.2 | 132 |

کمینه، میانگین، و بیشینه تعداد قوانین در برگ‌ها برای مجموعه قوانین IPC

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Wildcard Ratio | Alg. | ipc\_1k | | | ipc\_10k | | | ipc\_100k | | |
| min | avg | max | min | avg | max | min | avg | max |
| 0.1 | MC-SBC | 0 | 0.48 | 5 | 0 | 4.89 | 15 | 23 | 49.9 | 78 |
| Genetic | 0 | 0.48 | 4 | 0 | 4.74 | 13 | 30 | 49.4 | 73 |
| 1 | MC-SBC | 0 | 0.6 | 5 | 0 | 5.49 | 15 | 33 | 57.5 | 88 |
| Genetic | 0 | 0.55 | 5 | 0 | 5.37 | 17 | 37 | 56 | 86 |
| 5 | MC-SBC | 0 | 1.01 | 7 | 1 | 10.4 | 29 | 69 | 130 | 207 |
| Genetic | 0 | 0.82 | 6 | 1 | 8.47 | 19 | 52 | 83.4 | 121 |

نمودار کمینه، میانگین، و بیشینه تعداد قوانین در برگ‌ها بر اساس نسبت Wildcard برای ACL\_100k

نمودار کمینه، میانگین، و بیشینه تعداد قوانین در برگ‌ها بر اساس نسبت Wildcard برای FW\_100k

نمودار کمینه، میانگین، و بیشینه تعداد قوانین در برگ‌ها بر اساس نسبت Wildcard برای IPC\_100k

به طور کلی می‌توان گفت هر چقدر نسبت Wildcard بیشتر می‌شود، یا نسبت اندازه مجموعه قوانین به تعداد کل برگ‌ها افزایش می‌یابد، الگوریتم ژنتیک خروجی بهتری دارد. در بخش بعدی تاثیر این تفاوت را در بخش بر‌خط بررسی کرده‌ایم.

* + 1. زمان اجرای الگوریتم‌ها

عمل یافتن بیت‌های موثر، افراز مجموعه قوانین توسط آن‌ها، و ایجاد ساختمان داده نهایی به صورت برون‌خط انجام می‌شود. بنابراین سرعت انجام این کار در کارآیی بخش برخط تاثیر نخواهد داشت. اما اگر بخواهیم در بازه‌های زمانی ثابت و یا تحت شرایط خاص، ساختمان داده را مجدداً بازسازی کنیم، هر چقدر که این کار سریع‌تر انجام شود، سربار کمتری خواهد داشت. در ادامه ابتدا زمان اجرا به صورت تئوری تحلیل شده و سپس نتایج تجربی زمان اجرای دو الگوریتم نشان داده شده است.

* پیچیدگی زمانی

فرض کنید تعداد قوانین، طول مدخل در جدول جریان، تعداد مجموعه بیت‌های موثر، و تعداد بیت‌ها در هر مجموعه بیت موثر باشند. در بخش برون‌خط الگوریتم MC-SBC ابتدا معیارهای نسبت Wildcard، شاخص استقلال، شاخص تنوع، و فاکتور انتخاب محاسبه می‌شوند، سپس بر اساس آن‌ها بیت‌های موثر مشخص می‌شوند. نسبت Wildcard برای هر دو بیت و محاسبه می‌شود. از آن جا که بنابراین این معیار برای جفت محاسبه می‌گردد و برای هر یک مجموعه قوانین یک بار پیمایش می‌شود. بنابراین پیچیدگی زمانی آن خواهد بود. به همین صورت پیچیدگی زمانی برای محاسبه شاخص استقلال هم است. شاخص تنوع برای هر بیت به صورت جداگانه محاسبه می‌گردد و در هر کدام مجموعه قوانین یک بار پیمایش می‌شود. پس پیچیدگی زمانی آن است. فاکتور انتخاب بر اساس سه معیار قبلی برای هر جفت بیت با پیچیدگی زمانی محاسبه می‌شود. در نهایت پس از مشخص شدن مقادیر، برای انتخاب هر بیت موثر می‌بایست یک جفت بیت که مقدار فاکتور انتخاب در آن‌ها بیشینه است، مشخص گردند. این کار با زمان انجام می‌شود. از آن جا که باید s مجموعه با b بیت انتخاب گردند، زمان این بخش خواهد بود. به طور کلی، پیچیدگی زمانی به صورت رابطه (5-9) خواهد شد. از آن جا که است پیچیدگی زمانی به صورت رابطه (5-10) می‌شود و با توجه به این که رابطه نهایی به صورت رابطه (5-11( قابل بیان است.

|  |  |
| --- | --- |
| (5-9) |  |
| (5-10) |  |
| (5-11) |  |

حال فرض کنید در الگوریتم ژنتیک تعداد کروموزوم‌ها، طول کروموزوم، تعداد تکرار الگوریتم، و تعداد جهش در هر تکرار باشد. در گام مقداردهی اولیه بیت به صورت تصادفی انتخاب می‌شوند، پس پیچیدگی زمانی آن است. پس از آن الگوریتم بار تکرار می‌شود، و در هر تکرار اعمال انتخاب، تقاطع و جهش انجام می‌شوند. در گام انتخاب مجموعه قوانین ابتدا برای هر کروموزوم یک بار پیمایش می‌شود تا امتیاز آن کروموزوم مشخص گردد، سپس نیمی از کروموزوم‌ها برای بازتولید جمعیت انتخاب می‌شوند. بنابراین زمان اجرای گام انتخاب است. عمل تقاطع روی نیمی از کروموزوم‌های منتخب انجام می‌شود و پیچیدگی زمانی آن است. در نهایت عمل جهش به تعداد بار انجام شده و هر بار یک بیت در کروموزوم‌ها تغییر می‌کند، پس پیچیدگی زمانی این گام است. به طور کلی، پیچیدگی زمانی این الگوریتم به صورت رابطه (5-12) می‌باشد.

|  |  |
| --- | --- |
| (5-12) |  |

از آن جا که تعداد کروموزوم‌ها با توجه به طول مدخل و تعداد بیت موثر در هر مجموعه (طول کروموزم ) تعیین می‌گردد، و تعداد جهش‌ها ضریب ثابتی از است، داریم:

|  |  |
| --- | --- |
| (5-13) |  |
| (5-14) |  |

با توجه به رابطه (5-13) را نیز می‌توان بر اساس بیان کرد.

|  |  |
| --- | --- |
| (5-15) |  |

بنابراین رابطه (5-12) به شکل رابطه (5-16) قابل بیان است.

|  |  |
| --- | --- |
| (5-16) |  |

رابطه (5-16) پیچیدگی زمانی برای یافتن یک مجموعه بیت موثر است. اگر بخواهیم q مجموعه بیت موثر انتخاب کنیم، پیچیدگی زمانی الگوریتم به صورت رابطه (5-17) می‌شود.

|  |  |
| --- | --- |
| (5-17) |  |

بنابراین پیچیدگی زمانی الگوریتم ژنتیک با تعداد مجموعه بیت‌های موثر (q) نسبت خطی دارد. اگر q کوچک باشد، زمان اجرای الگوریتم ژنتیک بستگی به تعداد تکرار الگوریتم (K) خواهد داشت. به طوری که اگر باشد، زمان اجرای آن کمتر از زمان اجرای الگوریتم MC-SBC خواهد بود.

* نتایج تجربی زمان اجرا

‏شکل (4-16) زمان اجرای الگوریتم برون‌خط MC-SBC، و ژنتیک را برای یافتن 1 تا 4 مجموعه بیت‌های موثر، که هر کدام شامل 10 بیت موثر باشند، در مجموعه قوانین FW با 1000 قانون، نشان می‌دهد. الگوریتم ژنتیک زمان اجرای بسیار کمتری نسبت به الگوریتم برون‌خط MC-SBC دارد. در الگوریتم برون‌خط MC-SBC، داده‌های آماری شامل نسبت Wildcard، شاخص استقلال، و شاخص تنوع یک بار محاسبه می‌شوند، و از آن‌ها برای یافتن هر تعداد مجموعه قوانین استفاده می‌شود. بنابراین زمان اجرای این الگوریتم با افزایش تعداد مجموعه‌های بیت‌های موثر تغییر نمی‌کند و ثابت می‌ماند. در مقابل الگوریتم ژنتیک برای یافتن هر مجموعه بیت موثر به طور مستقل از هم کار می‌کند، و به همین دلیل با افزایش تعداد مجموعه‌ها، زمان اجرای آن به صورت خطی افزایش می‌یابد.

زمان اجرای الگوریتم برون‌خط MC-SBC، و ژنتیک

* 1. نتایج بخش بر‌خط

در این بخش یک نمونه از بخش بر‌خط MC-SBC را بر روی یک پردازنده گرافیکی که مشخصات آن در ‏جدول (4-5) آمده است، پیاده‌سازی کرده و تاثیر استفاده از دو الگوریتم ژنتیک و MC-SBC در بخش برون‌خط را، بر تاخیر دسته‌ای[[181]](#footnote-181) در فرآیند جستجوی برخط، مورد بررسی قرار داده‌ایم. در این پیاده‌سازی با توجه به محدودیت اندازه حافظه ثابت[[182]](#footnote-182) GPU، تعداد زیر مجموعه‌ها را برابر با 8 (3 بیت) و اندازه جدول‌های جستجو را برابر با 256 مدخل (8 بیت) در نظر گرفته‌ایم.

مشخصات پردازنده گرافیکی استفاده شده

|  |  |
| --- | --- |
| GT 540 | مدل |
| 96 | تعداد هسته‌های CUDA |
| 1344 | فرکانس ساعت پردازنده (MHz) |
| 900 | فرکانس ساعت حافظه (MHz) |
| DDR3 | واسط حافظه |
| 128 بیت | عرض واسط حافظه |
| 28.8 | پهنای باند حافظه (GB/sec) |

تاخیر دسته‌ای بخش بر‌خط

‏شکل (4-17) تاخیر دسته‌ای جستجوی بسته‌ها در بخش برخط الگوریتم MC-SBC را برای سه مجموعه قوانین IPC، ACL، و FW، با اندازه‌های 1000، 10000، و 100000 قانون، در دو حالت استفاده از الگوریتم ژنتیک و الگوریتم برون‌خط MC-SBC برای یافتن بیت‌های موثر، نشان می‌دهد. در هر اجرای برنامه تعداد بلوک‌های GPU برابر با 32، تعداد ریسمان‌ها در هر بلوک برابر با 64، و اندازه دسته برابر با 2048 چندتایی بوده است. هم چنین از مجموعه‌های قوانین با نسبت Wildcard برابر با 1 درصد، استفاده شده است.

تقریباً در تمامی حالات وقتی از الگوریتم ژنتیک در بخش برون‌خط استفاده کرده‌ایم، تاخیر دسته‌ای بهبود یافته است. البته مقدار این بهبود در موارد مختلف، با هم متفاوت است. هنگامی که اندازه مجموعه قوانین کوچک است، تفاوت اندکی (در حدود 30 میکروثانیه) بین دو روش وجود دارد. هنگامی که اندازه مجموعه قوانین افزایش می‌یابد تفاوت دو روش محسوس‌تر می‌شود. به عنوان مثال برای مجموعه قوانین FW با 100000 قانون تاخیر دسته‌ای حدود 269 میکروثانیه بهبود یافته است.

* + 1. بررسی تاثیر پارامترهای GPU در کارآیی

در این قسمت تاثیر پارامترهای GPU در سرعت عمل دسته‌بندی بسته بررسی شده است. ‏شکل (4-18) نمودار تاخیر دسته‌ای را بر اساس تعداد مختلف بلوک در GPU نشان می‌دهد. در آزمایش ‏شکل (4-18) تعداد ریسمان‌ها در هر بلوک برابر با 64 و اندازه دسته برابر با 2048 قرار داده شده است. با تغییر تعداد بلوک از 1 تا 16، تاخیر تقریباً به همان نسبت کاهش می‌یابد. اما بعد از آن تاخیر ثابت است و افزایش تعداد بلوک‌ها تاثیری در سرعت دسته‌بندی ندارد. این امر به دلیل محدودیت سخت‌افزاری است.

نمودار تاخیر دسته‌ای بر اساس تعداد بلوک

‏شکل (4-19) نمودار تاخیر را بر اساس تعداد مختلف ریسمان‌ها در هر بلوک نشان می‌دهد. در این جا تعداد بلوک‌ها برابر با 16 و اندازه دسته برابر با 2048 قرار داده شده است. طبق این نمودار تاخیر با افزایش تعداد ریسمان‌ها از 1 تا 64 کاهش می‌یابد. در ادامه با افزایش تعداد ریسمان‌ها بهبودی در تاخیر ایجاد نمی‌شود. این مسئله نیز به دلیل محدودیت سخت‌افزاری است.

نمودار تاخیر بر اساس تعداد ریسمان‌ها در هر بلوک

نمودار تاخیر بر اساس اندازه دسته

‏شکل (4-20) نمودار تاخیر را بر اساس اندازه دسته نشان می‌دهد. در این بخش تعداد بلوک‌ها برابر با 16 و تعداد ریسمان‌ها برابر با 64 در نظر گرفته شده است. مقدار تاخیر با تغییر اندازه دسته از 128 تا 1024 تغییر خاصی نمی‌کند. علت این امر این است که بر اساس تعداد بلوک‌ها و ریسمان‌ها در مجموع 1024 ریسمان در هر اجرا راه‌اندازی می‌شود. بنابراین اگر اندازه دسته کمتر از این مقدار باشد، تاخیر پردازشی تغییری نمی‌کند. با افزایش اندازه دسته بیش از 1024 تاخیر بیشتر می‌شود. چرا که در این صورت هر ریسمان بیش از یک بسته را در قوانین جستجو می‌کند.

* 1. جمع‌بندی

در این فصل از الگوریتم ژنتیک برای یافتن مجموعه بیت‌های موثر جهت افراز مجموعه قوانین استفاده کردیم. نتایج ما نشان داد وقتی نسبت Wildcard در مجموعه قوانین کم است، خروجی الگوریتم ژنتیک تفاوت چندانی با خروجی الگوریتم برون‌خط MC-SBC ندارد. اما وقتی نسبت Wildcard افزایش می‌یابد، الگوریتم ژنتیک مجموعه قوانین را بهتر افراز می‌کند. به طوری که در آن هم نسبت تکرار کاهش می‌یابد، و هم بیشینه اندازه برگ‌ها در ساختمان داده MC-SBC (Binth).

بهبود نحوه افراز در بخش برون‌خط MC-SBC، تاثیر مثبتی در کارآیی بخش برخط دسته‌بندی بسته‌ها خواهد داشت. از طرفی چون نسبت تکرار کاهش یافته متوسط تعداد قوانین در زیرمجموعه‌ها و برگ‌ها کمتر می‌شود. از طرفی دیگر چون Binth کمتر شده، پس قوانین به صورت بهتری بین برگ‌ها توزیع شده‌اند. تجربیات ما نشان داد وقتی اندازه مجموعه قوانین بزرگ‌تر می‌شود، استفاده از ژنتیک در گام برون‌خط تاثیر قابل توجهی در تاخیر دسته‌ای دسته‌بندی بسته‌ها دارد. به طوری که می‌تواند سرعت عمل دسته‌بندی را تا حدود 10 درصد بهبود دهد.

در بخش پایانی این فصل تاثیر پارامترهای مختلف GPU در سرعت دسته‌بندی بررسی شد. سرعت عمل دسته‌بندی با افزایش تعداد بلوک‌ها و تعداد ریسمان‌ها در هر بلوک از GPU بیشتر می‌شود تا به یک حد آستانه برسد. این حد آستانه با توجه به قدرت پردازنده گرافیکی تعیین می‌گردد. برای مثال در کار ما تعداد بهینه بلوک‌ها 16 و تعداد ریسمان‌ها در هر بلوک 64 می‌باشد. این مقادیر نسبت مستقیمی با اندازه دسته دارند که گذردهی کلی سیستم را تعیین می‌کند. مقدار بهینه اندازه دسته ضریبی از تعداد کل ریسمان‌ها است. چرا که در غیر این صورت بخشی از ریسمان‌ها در تکرار آخر داده‌ای برای پردازش نخواهند داشت و منابع سیستم اتلاف می‌شود.

1. فصل5: نتیجه‌گیری و کارهای آینده

ظهور شبکه‌های نرم‌افزار محور و ارائه طرحی نو برای مدیریت شبکه، هر چند ویژگی‌های مثبت و مزایای بسیاری به همراه داشته و چالش‌هایی مانند عدم انعطاف‌پذیری و برنامه‌ریزی در شبکه را به صورت بنیادی حل کرده است، با این حال، خود با چالش‌های جدیدی مانند مقیاس‌پذیری، امنیت، کارآیی و ... روبرو است. از طرفی، برخی از چالش‌هایی که در شبکه‌های سنتی بسیار مورد مطالعه قرار گرفته و تا حدی حل شده بود، در شبکه‌های SDN مجدداً جدی شده‌اند. دسته‌بندی بسته یکی از این موارد است. به این صورت که امکان تعریف جریان بر اساس تعداد زیادی فیلد، از سرآیند لایه‌های مختلف بسته در شبکه‌های SDN، دو اثر جانبی مهم در مسئله دسته‌بندی بسته دارد: اولاً، با افزایش طول مدخل جریان، جدول جریان به صورت عرضی گسترش می‌یابد؛ ثانیاً، کنترل بهتر و دقیق‌تر ترافیک نیازمند افزایش تعداد مدخل‌ها در جدول جریان است. این امر گسترش عمقی جدول جریان را به همراه خواهد داشت.

با افزایش اندازه جدول جریان در هر دو جهت، استفاده از TCAM در سوییچ‌های سخت‌افزاری با گذردهی بالا، هم از لحاظ فضای سخت‌افزاری و هم توان مصرفی، سربار بیشتری به سیستم تحمیل می‌کند. این مسئله در سوییچ‌های نرم‌افزاری نیز باعث افزایش تعداد دسترسی به حافظه و کاهش کارایی می‌شود. بنابراین محققان سعی کرده‌اند، روش‌های جدیدی برای مدیریت جدول‌جریان و مسئله دسته‌بندی بسته ارائه کنند. در برخی از این روش‌ها از ویژگی‌های SDN مانند وجود کنترلر مرکزی و قابلیت برنامه‌ریزی در شبکه استفاده شده، و سعی کرده‌اند بار محاسباتی مسئله دسته‌بندی بسته را به سوئیچ‌های لبه منتقل کنند. برخی کارهای دیگر نیز بدون توجه به ویژگی‌ها و امکانات SDN سعی در حل مسئله داشته اند. راه‌حل‌های ارائه شده را هم‌چنین می‌توان از آن جهت که برای سوئیچ‌های سخت‌افزاری ارائه شده‌اند یا برای سوئیچ‌های نرم‌افزاری، به دو دسته تقسیم بندی نمود.

با پیشرفت سخت‌افزارهای همه منظوره در سال‌های اخیر، سوئیچ‌های نرم‌افزاری توانسته‌اند به گذردهی بالا دست یابند و از آن‌جا که انعطاف‌پذیری بیشتر و امکان توسعه سریع‌تری از سوییچ سخت‌افزاری دارند، مورد توجه افراد بسیاری در محیط‌های علمی و صنعتی قرار گرفته‌اند. هم‌چنین در سال‌های اخیر، پردازنده‌های گرافیکی به دلیل قدرت پردازش موازی بالا، در حل مسائل غیرگرافیکی که قابلیت موازی‌سازی بالایی دارند، بسیار مورد استفاده قرار گرفته شده‌اند. در مسئله دسته‌بندی بسته هم کارهای ارزنده‌ای با استفاده از این پردازنده‌ها صورت گرفته است.

* 1. روش پیشنهادی و ارزیابی

در این پایان‌نامه مسئله دسته‌بندی بسته‌ها در سوئیچ‌، در شبکه‌های نرم‌افزار محور مورد بررسی قرار گرفت و سپس روشی برای افزایش کارآیی از لحاظ تاخیر پردازشی ارائه شد. در روش ارائه شده برای یافتن بیت‌های موثر جهت افراز مجموعه قوانین در الگوریتم MC-SBC، به جای استفاده از روش آماری، از الگوریتم ژنتیک بهره گرفته شده است. این کار در بخش برون‌خط این الگوریتم توانست به صورت بهتری مجموعه قوانین را افراز کند. به طوری که با افزایش نسبت Wildcard در مجموعه قوانین: اولاً نسبت تکرار تا حد قابل توجهی کاهش یافت که در نتیجه آن میانگین تعداد قوانین در برگ‌های درخت کاهش می‌یابد؛ ثانیاً، بیشینه تعداد قوانین در برگ‌ها کمتر شد که معنای آن این است که قوانین به صورت بهتری بین برگ‌ها توزیع می‌گردند. در نتیجه این دو بهبود، کارآیی بخش برخط افزایش می‌یابد.

نتایج گرفته شده از پیاده‌سازی یک نمونه از بخش برخط الگوریتم MC-SBC روی پردازنده گرافیکی نشان داد هنگامی که از الگوریتم ژنتیک در بخش برون‌خط استفاده می‌شود، تاخیر دسته‌ای برای یک مجموعه قوانین صدهزارتایی می‌تواند تا حدود 10 درصد کاهش یابد. الگوریتم ژنتیک برای یافتن بیت‌های موثر نسبت به الگوریتم برون‌خط MC-SBC بسیار سریع‌تر است. هرچند این مسئله تاثیری در کارآیی بخش برخط ندارد، اما باعث می‌شود بتوانیم عمل بازسازی ساختمان داده را با سرعت بیشتری انجام دهیم. ذکر این نکته ضروری است که بازسازی ساختمان داده در محیط‌هایی که مجموعه قوانین در آن‌ها بسیار تغییر می‌کند، به دفعات انجام می‌شود و تسریع این عملیات بر کاهش سربار کلی سیستم موثر است.

* 1. پژوهش‌های آینده

در این پژوهش از ساده‌ترین نوع الگوریتم ژنتیک استفاده شده است. در آینده برای بهبود آن کارهای زیر را می‌توان انجام داد:

1. در گام نخست این الگوریتم، به جای تولید تصادفی جمعیت اولیه، می‌توان این کار را با هوشمندی بیشتری انجام داد. برای مثال می‌توان کروموزوم‌ها را به صورتی تولید کرد که هر ژن آن‌ها (که یک بیت موثر را نشان می‌دهد) از یک فیلد در مجموعه قوانین انتخاب شوند. به این ترتیب سرعت همگرایی الگوریتم بهبود خواهد یافت.
2. در گام انتخاب می‌توان معیارهای دیگری را در تابع برازش دخیل کرد. به عنوان مثال از میانگین تعداد قوانین در برگ‌ها برای ارزیابی کیفیت کروموزوم‌ها می‌توان بهره جست. این کار می‌تواند باعث افراز بهتر مجموعه قوانین گردد.
3. در قسمت بازتولید، اعمال تقاطع و جهش را می‌توان به صورت دقیق‌تر و با کنترل بیشتری انجام داد. در عمل تقاطع می‌توان کروموزوم‌های جدید را به جای آن که از پیوند نیمه‌های دو کروموزوم والد تولید کرد، از پیوند بخش‌هایی با طول متفاوت این کار را انجام داد. هم‌چنین عمل جهش را می‌توانیم در هر کروموزوم، روی ژن‌هایی که مقدار تکراری در آن‌ها قرار گرفته است، انجام دهیم.
4. مراجع

[1] N. Feamster, J. Rexford, and E. Zegura, “The Road to SDN : An Intellectual History of Programmable Networks,” *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 2, pp. 87–98, 2014.

[2] D. Kreutz, E. Verıssimo, and S. Azodolmolky, “Software-Defined Networking: A Comprehensive Survey,” *Proc. IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[3] R. Masoudi and A. Ghaffari, “Software defined networks: A survey,” *J. Netw. Comput. Appl.*, vol. 67, no. C, pp. 1–25, 2016.

[4] “Software-Defined Networking : The New Norm for Networks,” 2012. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf.

[5] H. Hata, “A Study of Requirements for SDN Switch Platform,” in *Intelligent Signal Processing and Communications Systems (ISPACS), 2013 International Symposium on*, 2013, pp. 79–84.

[6] Y. R. Qu, S. Zhou, and V. K. Prasanna, “A Decomposition-based Approach for Scalable Many-field Packet Classification on Multi-core Processors \*,” *Int. J. Parallel Program.*, vol. 43, no. 6, pp. 965–987, 2015.

[7] Y. R. Qu and V. K. Prasanna, “Power-efficient range-match-based packet classification on FPGA,” in *Field Programmable Logic and Applications (FPL)*, 2015.

[8] S. Banerjee and K. Kannan, “Tag-In-Tag : Efficient Flow Table Management in SDN Switches,” in *Network and Service Management (CNSM), 2014 10th International Conference on*, 2014, pp. 109–117.

[9] H. Farhadi and A. Nakao, “Rethinking Flow Classification in SDN,” in *Cloud Engineering (IC2E), 2014 IEEE International Conference on*, 2014, pp. 598–603.

[10] K. G. Pérez, X. Yang, and S. Sezer, “A Configurable Packet Classification Architecture for Software-Defined Networking,” in *System-on-Chip Conference (SOCC)*, 2014, pp. 353–358.

[11] P. T. Congdon, P. Mohapatra, M. Farrens, and V. Akella, “Simultaneously reducing latency and power consumption in openflow switches,” *IEEE/ACM Trans. Netw.*, vol. 22, no. 3, pp. 1007–1020, 2014.

[12] Y. R. Qu, H. H. Zhang, S. Zhou, and V. K. Prasanna, “Optimizing many-field packet classification on FPGA, multi-core general purpose processor, and GPU,” in *Architectures for Networking and Communications Systems (ANCS)*, 2015, no. 3, pp. 87–98.

[13] M. Varvello, R. Laufer, F. Zhang, and T. V Lakshman, “Multi-Layer Packet Classification with Graphics Processing Units Categories and Subject Descriptors,” in *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*, 2014, pp. 109–120.

[14] C. Hsieh and N. Weng, “Many-Field Packet Classification for Software-Defined Networking Switches,” in *Proceedings of the 2016 Symposium on Architectures for Networking and Communications Systems*, 2016, pp. 13–24.

[15] H. Lim, N. Lee, G. Jin, J. Lee, Y. Choi, and C. Yim, “Boundary cutting for packet classification,” *IEEE/ACM Trans. Netw.*, vol. 22, no. 2, pp. 443–456, 2014.

[16] S. K. Oumya and C. H. S. E. M, “Selective Boundary Cutting For Packet Classification,” *Int. J. Sci. Eng. Technol. Res.*, vol. 4, no. 34, pp. 6786–6790, 2015.

[17] P. Gupta and N. Mckeown, “Packet Classification using Hierarchical Intelligent Cuttings,” in *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, 2003, pp. 213–224.

[18] S. Singh, F. Baboescu, G. Varghese, and J. Wang, “Packet Classification Using Multidimensional Cutting,” in *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, 2003, pp. 213–224.

[19] D. E. Taylor and J. S. Turner, “ClassBench: A packet classification benchmark,” *IEEE/ACM Trans. Netw.*, vol. 15, no. 3, pp. 499–511, 2007.

[20] W. Xia, Y. Wen, S. Member, C. H. Foh, and S. Member, “A Survey on Software-Defined Networking,” *IEEE Commun. Surv. Tutorials*, vol. 17, no. 1, pp. 27–51, 2014.

[21] “Traditional vs Software Defined Networking.” [Online]. Available: www.ipknowledge.net/wp-content/uploads/2014/12/SDN.pdf.

[22] Y. Gong, W. Huang, W. Wang, and Y. Lei, “A survey on software defined networking and its applications,” *Front. Comput. Sci.*, vol. 9, no. 6, pp. 827–845, 2015.

[23] V. Shamugam, I. Murray, L. J. A, and A. S. Sidhu, “Software Defined Networking challenges and future direction : A case study of implementing SDN features on OpenStack private cloud,” in *IOP Conference Series: Materials Science and Engineering*, 2016, vol. 121, no. 1, pp. 1–8.

[24] M. Karakus and A. Durresi, “A survey: Control plane scalability issues and approaches in Software-Defined Networking (SDN),” *Comput. Networks*, vol. 112, pp. 279–293, 2017.

[25] W. Li, W. Meng, and F. L. Kwok, “A survey on OpenFlow-based Software Defined Networks: Security challenges and countermeasures,” *J. Netw. Comput. Appl.*, vol. 68, no. C, pp. 126–139, 2016.

[26] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, “OpenFlow: Enabling Innovation in Campus Networks,” *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, p. 69, 2008.

[27] A. Doria, J. H. Salim, W. Wang, and L. Dong, “Forwarding and Control Element Separation (ForCES) Protocol Specification,” *Internet Engineering Task Force*, 2010. [Online]. Available: https://tools.ietf.org/html/rfc5810.

[28] B. Pfaff and B. Davie, “The Open vSwitch Database Management Protocol,” *Internet Engineering Task Force*, 2013. [Online]. Available: https://tools.ietf.org/html/rfc7047.

[29] H. Song, “Protocol-oblivious forwarding: unleash the power of SDN through a future-proof forwarding plane,” in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, 2013, pp. 127–132.

[30] M. Smith, M. Dvorkin, and P. Garg, “OpFlex Control Protocol,” *Internet Engineering Task Force*, 2014. [Online]. Available: https://tools.ietf.org/html/draft-smith-opflex-00.

[31] G. Bianchi, M. Bonola, A. Capone, and C. Cascone, “OpenState: Programming Platform-independent Stateful OpenFlow Applications Inside the Switch,” *Sigcomm Ccr*, vol. 44, no. 2, pp. 44–51, 2014.

[32] A. Lara, A. Kolasani, and B. Ramamurthy, “Network Innovation using OpenFlow : A Survey,” *IEEE Commun. Surv. Tutorials*, vol. 16, no. 1, pp. 493–512, 2013.

[33] C. Hao, Chang, and Y.-D. Lin, “OpenFlow Version Roadmap,” 2015. [Online]. Available: http://speed.cis.nctu.edu.tw/~ydlin/miscpub/indep\_frank.pdf.

[34] P. Gupta and N. Mckeown, “Algorithms for Packet Classification,” *IEEE Netw.*, vol. 15, no. 2, pp. 24–32, 2002.

[35] J. Sanders and E. Kandrot, *CUDA by Example: An Introduction to General-Purpose GPU Programming*, 1st ed. Addison-Wesley Professional, 2010.

[36] M. Arora, “The Architecture and Evolution of CPU-GPU Systems for General Purpose Computing.” University of California, San Diago, 2012.

[37] S. Singh, S. Singh, V. Banga, and C. Durlabh, “CUDA for GPGPU Applications – A Survey,” *Natl. Conf. Contem-poraryTechniques Technol. Electron. Eng.*, pp. 1–4, 2013.

[38] M. Mukerjee, D. Naylor, and B. Vavala, “Packet Processing on the GPU.” [Online]. Available: http://www.cs.cmu.edu/~bvavala/misc/project740/15-740\_Project\_files/Report.pdf.

[39] S. Han, K. Jang, K. Park, and S. Moon, “PacketShader : a GPU-Accelerated Software Router,” in *Proceedings of the ACM SIGCOMM 2010 conference*, 2010, pp. 195–206.

[40] G. Vasiliadis, S. Antonatos, M. Polychronakis, and P. Evangelos, “Gnort : High Performance Network Intrusion Detection Using Graphics Processors,” in *Proceedings of the 11th international symposium on Recent Advances in Intrusion Detection*, 2008, pp. 116–134.

[41] Y. Zhu and Y. Chen, “Hermes : An Integrated CPU / GPU Microarchitecture for IP Routing,” in *Design Automation Conference (DAC)*, 2011, pp. 1044–1049.

[42] R. Garg and S. Mittal, “Optimization by genetic algorithm,” *Int. J. Adv. Res. Comput. Sci. Softw. Eng.*, vol. 4, no. 4, pp. 587–589, 2014.

[43] D. E. Taylor and J. S. Turner, “Scalable packet classification using distributed crossproducing of field labels,” *Proc. IEEE 24th Annu. Jt. Conf. IEEE Comput. Commun. Soc.*, vol. 1, pp. 1–12, 2005.

[44] D. Yuan, X. Yang, X. Shi, B. Tang, and Y. Liu, “Multi-protocol query structure for SDN switch based on parallel bloom filter,” in *International Conference on ICT Convergence*, 2014, pp. 206–211.

[45] ب. جعفریان, “دسته‌بندی بسته‌ها در شبکه‌های سرعت بالا,” پایان‌نامه کارشناسی ارشد, دانشگاه تهران, 1391.

[46] C. Thomas H, C. E. Leiserson, R. Ronald L, and S. Chiffoed, *Introduction to Algorithms*, 3rd ed. London, England: MIT Press, 1988.

[47] P. Gupta, “Algorithms for routing lookups and packet classification,” STANFORD, 2000.

[48] M. M. Buddhikot, S. Suri, and M. Waldvogel, “Space decomposition techniques for fast layer-4 switching,” in *Protocols for High-Speed Networks VI*, 2000, pp. 25–41.

[49] F. Baboescu, S. Singh, and G. Varghese, “Packet classification for core routers: is there an alternative to CAMs?,” in *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications*, 2003, vol. 1, pp. 53–63.

[50] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, A. Networks, and M. Casado, “The Design and Implementation of Open vSwitch,” in *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation*, 2015, pp. 117–130.

1. واژه‌نامه انگلیسی به فارسی

|  |  |
| --- | --- |
| تجرید | Abstraction |
| عمل | Action |
| دستگاه | Appliance |
| ناهمگام | Asynchronous |
| خودمختار | Autonomous |
| دسته | Batch |
| تاخیر دسته‌ای | Batch Delay |
| محک‌زنی | Benchmarking |
| توزیع جهت‌دار | Biased Distribution |
| بردار بیتی | Bit Vector |
| بلوک | Block |
| گذرگاه | Bus |
| متمرکز شده | Centralized |
| جزء | Component |
| حافظه ثابت | Constant Memory |
| سطح کنترل | Control plane |
| هسته | Core |
| تقاطع | Crossover |
| برش | Cut |
| چرخه | Cycle |
| سطح داده | Data plane |
| پیش‌فرض | Default |
| قطعی | Deterministic |
| تنوع | Diversity |
| راه‌انداز | Driver |
| تکرار | Duplicate |
| موثر | Effective |
| مدخل | Entry |
| انطباق دقیق | Exact Match |
| مثبت کاذب | False Positive |
| برازش | Fitness |
| جریان | Flow |
| تطابق کامل | Full Match |
| به طور کامل کارآمد | Fully Efficient |
| دانه‌بندی | Granularity |
| استقلال | Independence |
| لختی | Inertia |
| زیرساختار | Infrastructure |
| خط | Lane |
| نرخ خط | Line rate |
| منطقی | Logical |
| منطقی، به طور منطقی | Logically |
| ماسک | Mask |
| بیشینه | Maximum |
| ادغام | Merge |
| فراداده | Metadata |
| کمینه | Minimum |
| جهش | Mutation |
| دو به دو ناسازگار | Mutually exclusive |
| نقطه دسترسی شبکه | Network Access Point |
| حالت شبکه | Network state |
| واسط شمالی | North-bound Interface |
| برون‌خط | Offline |
| برخط | Online |
| خارج از نوبت | Out-of-order |
| کارآیی | Performance |
| خط لوله | Pipeline |
| سکو | Platform |
| مخزن | Pool |
| پیش‌تصفیه | Pre-filtering |
| انطباق پیشوندی | Prefix Match |
| پیش‌پردازش | Preprocessing |
| شمارنده برنامه | Program Counter |
| قابل برنامه‌ریزی | Programmable |
| انطباق بازه‌ای | Range Match |
| عبارت منظم | Regular Expression |
| نسبت تکرار | Replication Ratio |
| برنامه‌ریزی قانون | Rule Programming |
| مقیاس پذیری | Scalability |
| زمان‌بند | Scheduler |
| زمان‌بندی | Scheduling |
| گستره | Scope |
| قطعه‌بندی | Segmentation |
| شکاف | Slot |
| هموارسازی | Smoothing |
| واسط جنوبی | South-bound Interface |
| خلوتی | Sparsity |
| چندپردازنده جریانی | Streaming Multiprocessor |
| رشته | String |
| زیرمحدوده | Subrange |
| زیرمجموعه | Subset |
| راه‌گزینی | Switching |
| مصنوعی | Synthetic |
| قانون هدف | Target Rule |
| ریسمان | Thread |
| گذردهی | Throughput |
| چندتایی | Tuple |
| یکنواخت | Uniform |
| یک‌دست‌سازی | Unifying |
| منطقه | Zone |

1. واژه‌نامه فارسی به انگلیسی

|  |  |
| --- | --- |
| ادغام | Merge |
| استقلال | Independence |
| انطباق بازه‌ای | Range Match |
| انطباق پیشوندی | Prefix Match |
| انطباق دقیق | Exact Match |
| برازش | Fitness |
| برخط | Online |
| بردار بیتی | Bit Vector |
| بریدن | Cut |
| برنامه‌ریزی قانون | Rule Programming |
| برون‌خط | Offline |
| بلوک | Block |
| به طور کامل کارآمد | Fully Efficient |
| بیشینه | Maximum |
| پیش‌پردازش | Preprocessing |
| پیش‌تصفیه | Pre-filtering |
| پیش‌فرض | Default |
| تاخیر دسته‌ای | Batch Delay |
| تجرید | Abstraction |
| تطابق کامل | Full Match |
| تقاطع | Crossover |
| تکرار | Duplicate |
| تنوع | Diversity |
| توزیع جهت‌دار | Biased Distribution |
| جریان | Flow |
| جزء | Component |
| جهش | Mutation |
| چرخه | Cycle |
| چندپردازنده جریانی | Streaming Multiprocessor |
| چندتایی | Tuple |
| حافظه ثابت | Constant Memory |
| حالت شبکه | Network state |
| خارج از نوبت | Out-of-order |
| خط | Lane |
| خط لوله | Pipeline |
| خلوتی | Sparsity |
| خودمختار | Autonomous |
| دانه‌بندی | Granularity |
| دستگاه | Appliance |
| دسته | Batch |
| دو به دو ناسازگار | Mutually exclusive |
| راه‌انداز | Driver |
| راه‌گزینی | Switching |
| رشته | String |
| ریسمان | Thread |
| زمان‌بند | Scheduler |
| زمان‌بندی | Scheduling |
| زیرساختار | Infrastructure |
| زیرمجموعه | Subset |
| زیرمحدوده | Subrange |
| سطح داده | Data plane |
| سطح کنترل | Control plane |
| سکو | Platform |
| شکاف | Slot |
| شمارنده برنامه | Program Counter |
| عبارت منظم | Regular Expression |
| عمل | Action |
| فراداده | Metadata |
| قابل برنامه‌ریزی | Programmable |
| قانون هدف | Target Rule |
| قطعه‌بندی | Segmentation |
| قطعی | Deterministic |
| کارآیی | Performance |
| کمینه | Minimum |
| گذردهی | Throughput |
| گذرگاه | Bus |
| گستره | Scope |
| لختی | Inertia |
| ماسک | Mask |
| متمرکز شده | Centralized |
| مثبت کاذب | False Positive |
| محک‌زنی | Benchmarking |
| مخزن | Pool |
| مدخل | Entry |
| مصنوعی | Synthetic |
| مقیاس‌پذیری | Scalability |
| منطقه | Zone |
| منطقی | Logical |
| منطقی، به طور منطقی | Logically |
| موثر | Effective |
| ناهمگام | Asynchronous |
| نرخ خط | Line rate |
| نسبت تکرار | Replication Ratio |
| نقطه دسترسی شبکه | Network Access Point |
| واسط جنوبی | South-bound Interface |
| واسط شمالی | North-bound Interface |
| هسته | Core |
| هموارسازی | Smoothing |
| یک‌دست‌سازی | Unifying |
| یکنواخت | Uniform |

**Abstract:**

In Software Defined Networking, a flow can be defined using arbitrary set of header fields of each layer. For example, In OpenFlow 1.3.1, total of 15 fields may be used to define a flow. Allowing to define flows with more fields, on the one hand, causes the length of flow table entries to be enlarged; On the other hand, it enables finer control on network traffic which in turn, causes the number of flow table entries to be increased. Consequently, design of a solution for packet classification in high performance environments has been turned out to be a big challenge. To address this problem in soft switches, researchers try to reduce the number of candidate rules that a tuple must match with them. In the MC-SBC algorithm, a two-level trie based structure is proposed for lookup tables in which a set of effective bits are detected through a statistical method, in order to partition the ruleset and construct the trie structure.

In this dissertation, genetic algorithm has been applied to find the effective bit sets for ruleset partitioning. The results show that the proposed method partitions the rulesets in more balanced subsets, such that, both the replication ratio and the maximum number of rules in a leaf node of the trie are decreased. Moreover, the results of running our prototype on a single NVIDIA GPU, shows that packet classification can be performed up to 10 percent faster for large rule sets.

**Keywords: Sotware Defined Networking, Packet Classification, OpenFow Switch, Graphics Processing Unit, Genetic Algorithm**

|  |  |  |
| --- | --- | --- |
| Optimized_image_813b5ec2.png |  | Images__Logo_FE.gif |
| **University of Tehran**  **College of Engineering**    Faculty of Electrical and Computer Engineering | | |
| **Packet Classification in Software Defined Networking** | | |
| A thesis submitted to the Graduate Studies Office  In partial fulfillment of the requirements for  The degree of M.Sc in  Hardware Engineering | | |
| **By:**  **Mohammad Reza Piroozi**  **Supervisor:**  **Dr. Nasser Yazdani** | | |

1. Control Plane [↑](#footnote-ref-1)
2. Data Plane [↑](#footnote-ref-2)
3. Inertia [↑](#footnote-ref-3)
4. Software Defined Networking [↑](#footnote-ref-4)
5. Open Netwotking Foundation [↑](#footnote-ref-5)
6. Forwarding [↑](#footnote-ref-6)
7. Programmable [↑](#footnote-ref-7)
8. Flow [↑](#footnote-ref-8)
9. Autonomous [↑](#footnote-ref-9)
10. Flow table [↑](#footnote-ref-10)
11. Match [↑](#footnote-ref-11)
12. kernel function [↑](#footnote-ref-12)
13. Firewall [↑](#footnote-ref-13)
14. Quality of Service [↑](#footnote-ref-14)
15. Packet Classification [↑](#footnote-ref-15)
16. Throughput [↑](#footnote-ref-16)
17. Multi-Protocol Lable Switching [↑](#footnote-ref-17)
18. Field Programmable Gate Array [↑](#footnote-ref-18)
19. Tuple Space Search [↑](#footnote-ref-19)
20. Multidimensional-Cutting via Selective Bit-Concatenation [↑](#footnote-ref-20)
21. Offline [↑](#footnote-ref-21)
22. Online [↑](#footnote-ref-22)
23. Wildcard ratio [↑](#footnote-ref-23)
24. Independence index [↑](#footnote-ref-24)
25. Diversity index [↑](#footnote-ref-25)
26. Benchmarking [↑](#footnote-ref-26)
27. Packet classification [↑](#footnote-ref-27)
28. Information and Communications Technology [↑](#footnote-ref-28)
29. appliance [↑](#footnote-ref-29)
30. Segmentation [↑](#footnote-ref-30)
31. Zone [↑](#footnote-ref-31)
32. Open Networking Foundation [↑](#footnote-ref-32)
33. Forwarding [↑](#footnote-ref-33)
34. Destination [↑](#footnote-ref-34)
35. Flow [↑](#footnote-ref-35)
36. Actions [↑](#footnote-ref-36)
37. Unifying [↑](#footnote-ref-37)
38. Controller [↑](#footnote-ref-38)
39. Platform [↑](#footnote-ref-39)
40. Abstraction Layers [↑](#footnote-ref-40)
41. Applications [↑](#footnote-ref-41)
42. Logically [↑](#footnote-ref-42)
43. Network state [↑](#footnote-ref-43)
44. Infrastructure [↑](#footnote-ref-44)
45. South-bound interface [↑](#footnote-ref-45)
46. Centralized [↑](#footnote-ref-46)
47. North-bound interface [↑](#footnote-ref-47)
48. Application Layer [↑](#footnote-ref-48)
49. Scalability [↑](#footnote-ref-49)
50. Performane [↑](#footnote-ref-50)
51. Security [↑](#footnote-ref-51)
52. Reliability [↑](#footnote-ref-52)
53. Driver [↑](#footnote-ref-53)
54. Attacker [↑](#footnote-ref-54)
55. Denial-of-service [↑](#footnote-ref-55)
56. group table [↑](#footnote-ref-56)
57. Pipeline [↑](#footnote-ref-57)
58. Action [↑](#footnote-ref-58)
59. Metadata [↑](#footnote-ref-59)
60. Logical port [↑](#footnote-ref-60)
61. Reserved port [↑](#footnote-ref-61)
62. Link aggregation groups [↑](#footnote-ref-62)
63. Tunnels [↑](#footnote-ref-63)
64. Group entries [↑](#footnote-ref-64)
65. Bit mask [↑](#footnote-ref-65)
66. Network Access Point [↑](#footnote-ref-66)
67. Mutually exclusive [↑](#footnote-ref-67)
68. Component [↑](#footnote-ref-68)
69. Reqular experition [↑](#footnote-ref-69)
70. Exact match [↑](#footnote-ref-70)
71. Prefix match [↑](#footnote-ref-71)
72. Range match [↑](#footnote-ref-72)
73. Packet classifier [↑](#footnote-ref-73)
74. Kernel functions [↑](#footnote-ref-74)
75. General-purpose computing on graphics processing units [↑](#footnote-ref-75)
76. Compute Unified Device Architecture [↑](#footnote-ref-76)
77. Graphics processing unit [↑](#footnote-ref-77)
78. streaming multiprocessor [↑](#footnote-ref-78)
79. online [↑](#footnote-ref-79)
80. Lane [↑](#footnote-ref-80)
81. Single Instruction Multiple Thread [↑](#footnote-ref-81)
82. Cycle [↑](#footnote-ref-82)
83. Thread [↑](#footnote-ref-83)
84. Scheduling [↑](#footnote-ref-84)
85. Granularity [↑](#footnote-ref-85)
86. Program Counter [↑](#footnote-ref-86)
87. pool [↑](#footnote-ref-87)
88. Switching [↑](#footnote-ref-88)
89. Out-of-order [↑](#footnote-ref-89)
90. Fully efficient [↑](#footnote-ref-90)
91. Branch synchronization stack [↑](#footnote-ref-91)
92. Medical imaging [↑](#footnote-ref-92)
93. EnvIronmental science [↑](#footnote-ref-93)
94. string pattern-matching [↑](#footnote-ref-94)
95. batch [↑](#footnote-ref-95)
96. Genetic Algorithms [↑](#footnote-ref-96)
97. Crossover [↑](#footnote-ref-97)
98. Mutation [↑](#footnote-ref-98)
99. Fitness [↑](#footnote-ref-99)
100. Fitness function [↑](#footnote-ref-100)
101. Benchmarking [↑](#footnote-ref-101)
102. Unique values [↑](#footnote-ref-102)
103. Wildcard Ratio [↑](#footnote-ref-103)
104. Access Control List [↑](#footnote-ref-104)
105. Firewall [↑](#footnote-ref-105)
106. IP Chain [↑](#footnote-ref-106)
107. Smoothing [↑](#footnote-ref-107)
108. Scope [↑](#footnote-ref-108)
109. Line rate [↑](#footnote-ref-109)
110. Single Instructuin Multiple Data [↑](#footnote-ref-110)
111. Multi-Protocol Lable Switching [↑](#footnote-ref-111)
112. Tag-in-Tag [↑](#footnote-ref-112)
113. Path Tag [↑](#footnote-ref-113)
114. Flow Tag [↑](#footnote-ref-114)
115. deterministic [↑](#footnote-ref-115)
116. core switch [↑](#footnote-ref-116)
117. source edge [↑](#footnote-ref-117)
118. High Priority Matched Rule [↑](#footnote-ref-118)
119. Merge [↑](#footnote-ref-119)
120. Hash functions [↑](#footnote-ref-120)
121. Collision [↑](#footnote-ref-121)
122. Hierarchical Tries [↑](#footnote-ref-122)
123. Set Pruning Tries [↑](#footnote-ref-123)
124. Hirerachical Binary Search Tree [↑](#footnote-ref-124)
125. Fat Inverted Segment Tree [↑](#footnote-ref-125)
126. Grif of Segment Tree [↑](#footnote-ref-126)
127. Heuristic [↑](#footnote-ref-127)
128. Hierarchical Intelligent Cuttings [↑](#footnote-ref-128)
129. Tuple Space Search [↑](#footnote-ref-129)
130. Ternary Content Addressable Memory [↑](#footnote-ref-130)
131. Bitmap Intersection [↑](#footnote-ref-131)
132. Next-trie pointer [↑](#footnote-ref-132)
133. Merge [↑](#footnote-ref-133)
134. Preprocessing [↑](#footnote-ref-134)
135. Range tree [↑](#footnote-ref-135)
136. Subrange [↑](#footnote-ref-136)
137. Bit Vector [↑](#footnote-ref-137)
138. Rule ID [↑](#footnote-ref-138)
139. Bus [↑](#footnote-ref-139)
140. Asynchronous [↑](#footnote-ref-140)
141. GPU-Accelerated Software Switch [↑](#footnote-ref-141)
142. Tuple Space Search [↑](#footnote-ref-142)
143. class table [↑](#footnote-ref-143)
144. hash table [↑](#footnote-ref-144)
145. class mask [↑](#footnote-ref-145)
146. slot [↑](#footnote-ref-146)
147. uniform [↑](#footnote-ref-147)
148. False positive [↑](#footnote-ref-148)
149. Multidimentional Cutting via Seletive Bit Concatination [↑](#footnote-ref-149)
150. Effective Bits [↑](#footnote-ref-150)
151. Offline [↑](#footnote-ref-151)
152. wildcard ratio [↑](#footnote-ref-152)
153. independence indices [↑](#footnote-ref-153)
154. diversity indices [↑](#footnote-ref-154)
155. pre-filtering [↑](#footnote-ref-155)
156. full match [↑](#footnote-ref-156)
157. Default action [↑](#footnote-ref-157)
158. Multidimentional Cutting via Seletive Bit Concatination [↑](#footnote-ref-158)
159. Sparcity [↑](#footnote-ref-159)
160. Biased distribution [↑](#footnote-ref-160)
161. Group [↑](#footnote-ref-161)
162. Subset [↑](#footnote-ref-162)
163. Sample Space [↑](#footnote-ref-163)
164. Rule Programming [↑](#footnote-ref-164)
165. wildcard ratio [↑](#footnote-ref-165)
166. independence indices [↑](#footnote-ref-166)
167. diversity indices [↑](#footnote-ref-167)
168. pre-filtering [↑](#footnote-ref-168)
169. full match [↑](#footnote-ref-169)
170. target rule [↑](#footnote-ref-170)
171. Default action [↑](#footnote-ref-171)
172. Cut [↑](#footnote-ref-172)
173. Subsets [↑](#footnote-ref-173)
174. Duplicate [↑](#footnote-ref-174)
175. Replication Ratio [↑](#footnote-ref-175)
176. Fitness Function [↑](#footnote-ref-176)
177. Crossover [↑](#footnote-ref-177)
178. Mutation [↑](#footnote-ref-178)
179. Minimum [↑](#footnote-ref-179)
180. Maximum [↑](#footnote-ref-180)
181. Batch delay [↑](#footnote-ref-181)
182. Constant Memory [↑](#footnote-ref-182)