این تحقیق بعنوان بخشی از پروژه درس شبکه های مخابرات داده پیشرفته است که توسط دکتر پاکروان ارائه میشود.

برنامه نویسی مسیر داده: فراتر از Open-Flow

علیرضا شیرزاد[1]

[1] دانشجوی مقطع کارشناسی ارشد مخابرات امن و رمزنگاری دانشگاه صنعتی شریف

چکیده: در سیر تکاملی شبکه های مخابراتی داده به سمت "باز" شدن و خارج شدن از انحصار تولید کنندگان ابزارهای شبکه، یکی از مهم ترین تکنولوژی های عصر حاضر، زبان های توسعه ی صفحه ی داده ی شبکه اند. سوییچ های مخابراتی از انتزاعی ترین لایه یعنی لایه ی مدیریت شروع به "باز" شدن کردند و در دنیای شبکه های نرم افزار محور، لایه ی کنترل نیز در اختیار برنامه نویسان و توسعه دهندگان شبکه قرار گرفت، اما کماکان صفحه ی داده، جعبه سیاهی بود که توسط تولید کنندگان طراحی می شد و هیچگونه تغییری در آن ممکن نبود. با مطرح شدن زبان P۴، محققان توانستد فارغ از پروتکل های صفحه ی داده و مستقل از اینکه چه سخت افزار یا نرم افزاری میزبانی سوییچ را برعهده دارد، پایپ لاین پردازش یک بسته را به صورت استاندراد توصیف کنند. این تکنولوژی که برنامه نویسی صفحه ی داده نام دارد باعث تغییر کلی در پارادایم توسعه ی شبکه شده است و فرصت ها و چالش های بسیار زیادی را مطرح کرده است. در این پژوهش به آشنایی با زبان P۴ که یکی از اصلی ترین زبان های توسعه صفحه داده است می پردازیم، تاریخ شبکه های نرم افزار محور را مورد بررسی قرار می دهیم، مبانی برنامه نویسی با P۴ را مرور می کنیم، از کاربرد های این زبان در صنعت می گوییم و افق های پیش روی این زبان و برنامه نویسی صفحه ی داده را شرح می دهیم.

کلمات کلیدی: شبکه، شبکه های نرم افزار محور، مسیر داده، P۴

۱ مقدمه

در سال های اخیر، با گسترش طیف استفاده از شبکه های مخابرات داده در میان جوامع مختلف، پژوهش ها و تحقیقات گسترده ای برای تولید الگوریتم ها و پروتکل های مختلف شبکه شدت گرفته است. در واقع، اپراتورها و هدایت کنندگان شبکه های مختلف، برای پاسخگویی به نیاز های روزافزون، متغیر و جدید مشتریان خود، ناچار به معرفی نوآوری هایی در زیرساخت شبکه های خود شدند. لکن درهای نوآوری در ابتدا بر روی پژوهشگران و اپراتور ها بسته بود، چرا که ابزار های زیرساختی یک شبکه، که مهم ترین آن ها سوییچ ها و روتر ها هستند، ساختاری کاملا بسته داشتند و همه چیز از قبل در ساختار سوییچ به صورت سخت افزاری ،نرم افزاری یا ادغام شده تعبیه شده بود. با معرفی شدن شبکه های نرم افزار محور و معرفی پروتکل های مدیریتی و کنترلی سوییچ های شبکه، دروازه ای جدید به سوی نوآوری برای محققان و صنعتگران باز شد تا با این قابلیت بتوانند شبکه های خود را منعطف و قابل برنامه نویسی کنند. در این مقاله به آخرین شکل تکامل یافته ی شبکه های نرم افزار محور می پردازیم که در آن صفحه ی داده ی یک سوییچ، با یک زبان واحد و فارغ از سخت افزار، قابل برنامه نویسی است. این تکنولوژی به محققان و راهبران شبکه این امکان را می دهد که تست و بهره برداری از پروتکل های جدید را بسیار آسان و بدون دغدغه انجام دهند.

در این مقاله به بررسی کامل این تکنولوژی می پردازیم و تمرکز خود را بر روی مهم ترین بستر عملیاتی توصیف صفحه داده، یعنی P۴ قرار می دهیم [۲] . در ادامه ی این مقاله در بخش ۲ به تاریخچه ی شبکه های نرم افزار محور و تکامل آن تا P۴ می پردازیم، در بخش ۳ ساختار برنامه نویسی سوییچ ها با زبان P۴ را توضیح می دهیم و بخش ۴ را به معرفی کاربردهای عملی در دنیای آکادمیک و صنعتی اختصاص می دهیم. در نهایت در بخش ۵ به چالش ها و پروژه های پیش رو می پردازیم و مسیر های پژوهشی را معرفی می کنیم.

۲ مبانی و تاریخچه‌ی شبکه های نرم افزار محور

الگوریتم های اجرا شده و پیام های مبادله شده توسط سوییچ های مخابراتی در سه دسته ی کلی قرار می گیرند:

۱. مسیر (صفحه) مدیریت [۱]
۲. مسیر (صفحه) کنترلی [۲]
۳. مسیر (صفحه) داده [۳]

هر کدام از این مسیر ها، مسئول بخشی از عملیات انتقال بسته های شبکه هستند که در ادامه آن ها را توضیح می دهیم.

صفحه ی مدیریت برای پیکربندی و اعمال مدیریتی یک سوییچ ایجاد شده است. این بخش از یک سوییچ توسط اپراتور ها به صورت سطح بالا کنترل می شود و سیاست ها کلی شبکه را اعمال می کند. لازم به ذکر است که این لایه از شبکه از ابتدا قابل برنامه نویسی بوده و در واقع برای این بوجود آمده که واسط بین اپراتور ها و تجهیزات شبکه باشد. به عنوان مثال در سال ۱۹۸۸ پروتکل SNMPv۱ معرفی شده، لذا اولین بخش قابل برنامه ریزی تجهیزات شبکه از لحاظ تاریخی، صفحه ی مدیریت می باشد.

صفحه ی کنترل مغز متفکر یک شبکه و در واقع بخش تصمیم گیر که الگوریتم های اصلی مسیریابی در آن اجرا می شود، همین صفحه ی کنترلی شبکه است. در واقع پاسخ به این سئوال که یک سوییچ "چگونه" عمل کند در اختیار صفحه ی کنترل است. به عنوان مثال الگوریتم های مسیریابی معروف BGP و OSPF در این لایه اجرا می شوند و پیام های کنترلی این الگوریتم ها در شبکه رد و بدل می شود.

صفحه ی داده این مسیر از سوییچ ها ، وظیفه ی دریافت، پردازش و ارسال بسته های شبکه در مسیر های مخصوص خود را دارد. در واقع مسیر داده ی یک سوییچ مانند اعضای حرکتی یک بدن است که دستورات چگونگی عملکرد آن از سمت صفحه ی کنترل به آن ارسال می شود.

شکل ۱: رابطه ی کلی صفحات مدیریت، کنترل و داده در یک شبکه ی متشکل از تعدادی سوییچ

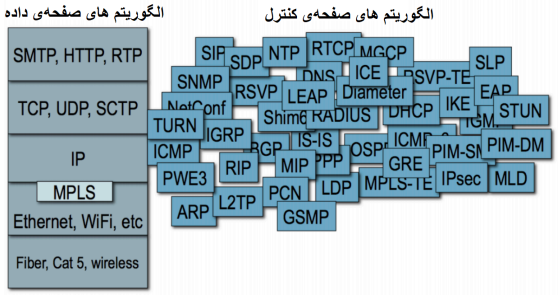
با معرفی بخش های اصلی یک سوییچ، در بخش های بعدی به ایده ی شبکه های نرم افزار محور در سطح صفحه ی کنترل می پردازیم و پس از بیان کم و کاستی های آن به سراغ موضوع اصلی این مقاله، یعنی برنامه نویسی صفحه ی داده می رویم.

[1] Management Plane
[2] Control Plane
[3] Data Plane

۲-۱ مشکلات شبکه‌های سنتی

همانطور که در بخش قبل به آن اشاره شد، تا اوائل دهه ی آغازین قرن ۲۱ تنها بخشی از شبکه که به صورت کامل باز و در اختیار اپراتور ها قرار گرفته بود، بخش مدیریت شبکه بود لکن مشکلات و مسائل عدیده ای محققان را به این فکر واداشت که صفحه ی کنترل شبکه نیز باید قابل برنامه ریزی باشد و به صورت کاملا باز در اختیار اپراتور ها قرار بگیرد. از جمله ی این مسائل می توان به موارد زیر اشاره کرد:

۱. به علت ماهیت توزیع شده ی الگوریتم های صفحه ی داده، پیچیدگی راه اندازی، تغییر و توسعه ی این الگوریتم ها ، فرایندی بسیار سخت و پیچیده بود.



شکل ۲: نمایی از تعدد الگوریتم ها و پیچیدگی پروتکل های اجرایی در صفحه ی داده و صفحه ی کنترلی

۲. افزایش نرخ خطاهای فنی و انسانی در صفحه ی مدیریت و غیر قابل رصد بودن مشکلات در صفحه ی کنترلی که نتیجه ی آن عیب یابی و تعمیر بسیار سخت و کند بود. به عنوان مثال به علت پیکربندی نامناسب پروتکل BGP در گوگل، ترافیک کشور ژاپن به یک سیاهچاله (جایی که ترافیک به مقصد خود نمی رسد) فرستاده شده بود. این اتفاق که تنها برای چند ساعت باعث قطع شدن اینترنت در کشور ژاپن بود، خسارات بسیار زیادی را به همراه داشت. در مطالعه ای، نشان داده شد که ۵۰ الی ۸۰ درصد دلایل در دسترس نبود شبکه ها ، مربوط به خطاهای انسانی در مدیریت شبکه است. [۳]

۳. نبودن فضایی برای نوآوری در طراحی و تست پروتکل های شبکه، یکی از مشکلات اساسی پژوهشگران بود. پیاده سازی در محیط آکادمیک معنا نداشت و همه ی مطالعات و پژوهش ها در سطح شبیه سازی باقی می ماند. برای پیاده سازی یک نوآوری می بایست کل ابزارهای شبکه از پایین ترین لایه یعنی سخت افزار تا بالاترین لایه یعنی اپلیکیشن های در حال اجرا تغییر بکنند که عملا ممکن نبود چرا که این کار تنها توسط تولیدکننده ی ابزار شبکه قابل انجام بود.

۴. غیرقابل انعطاف بودن شبکه ها معضل بزرگ اپراتور ها بود. انعطاف شبک های یک اپراتور، محدود به انعطاف تولید کننده ی تجهیزات شبکه بود. هیچگونه پروتکل با طرح جدیدی بدون پیاده سازی تولید کنندگان تجهیزات قابل اجرا نبود.

۵. انحصار تجهیزات شبکه از سخت افزار تا نرم افزار در دستان یک تولیدکننده بود چرا که تولیدکنندگان، تجهیزات خود را هماهنگ با یک استاندارد واحد نمی ساختند و قابل تعامل با یکدیگر نبودند. به عنوان مثال اگر یک اپراتور، یک سوییچ شبکه را از تولید کننده X تهیه می کرد، مجبور بود تا تمام تجهیزات شبکه که در تعامل با این سوییچ بودند را از همین تولیدکننده X خریداری کند.

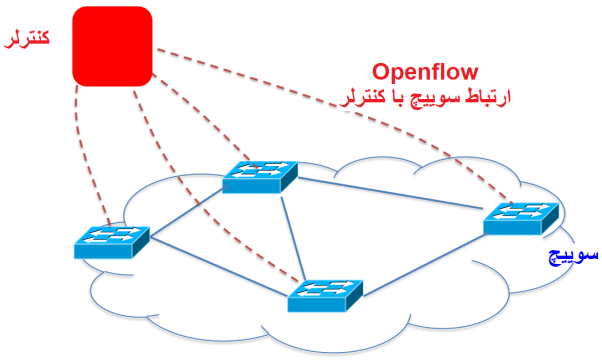
موارد ذکر شده، جرقه ی ایده ی شبکه های نرم افزار محور [۴] را در ذهن پژوهشگران زد.

۲-۲ شبکه های نرم افزار محور

مشکلات مطرح شده در بخش قبل، همگی ناشی از ساختار غیرمنعطف و فروشنده [۵] محور شبکه های سنتی بود. در واقع تجهیزات شبکه ، به صورت یک جعبه سیاه غیر قابل برنامه ریزی و کاملا غیر منعطف در اختیار مصرف کنندگان و اپراتورها قرار می گرفت. تنها نقش یک اپراتور در استفاده از این تجهیزات، پیکربندی و اتصال آن ها بود که باعث اتفاق های ناگزیری می شد که در بخش قبل به آن ها اشاره شد. در معماری شبکه های نرم افزار محور، تجهیزات شبکه "باز" شدند،

شکل ۳: نمایی از تجهیزات بسته ی شبکه پیش از معرفی شبکه های نرم افزارمحور

یدین معنی که تنها قابلیت انجام کارهای بسیار ساده و کارهای مختص به مسیر داده را داشتند، مثلا فروارد بسته ها و تغییر و بروزرسانی در سربار [۶] ها. در نتیجه مغز متفکر کنترلی هر سوییچ از آن گرفته شد و همگی به یک واحد متمرکز به نام کنترلر (کنترل کننده) [۷] برون سپاری شد. با معرفی پروتکل open-flow در مقاله [۱] ، همه ی سوییچ ها با یک پروتکل واحد با کنترلر در ارتباط بودند و دستورات کنترلی خود را از این واحد دریافت می کردند. حال تمام الگوریتم های شبکه در این واحد مرکزی اجرا می شد و سوییچ ها صرفا تابع دستورات کنترلر بودند.



شکل ۴: معماری شبکه های نرم افزار محور با معرفی پروتکل open-flow

۲-۳ برنامه نویسی صفحه‌ی داده

پس از معرفی شبکه های نرم افزار محور، کماکان معضل هایی وجود داشت که حل نشده باقی مانده بود. این معضل ها عبارتند از:

۱. امکان معرفی پروتکل صفحه ی داده هنوز وجود نداشت، به عنوان مثال یک پژوهشگر نمی توانست پروتکلی جایگزین برای IP ارائه دهد. هر پروتکل جدید صفحه ی داده می بایست در ابتدا در استاندارد open-flow درج می شد، سپس فروشندگان تجهیزات آن را پیاده می کردندو در نهایت این امکان برای همگان قرار می گرفت تا از آن استفاده کنند.

۲. استاندارد open-flow رفته رفته بزرگ و پیچیده تر شده است و فروشندگان و تولیدکنندگان سوییچ ها، عموما تنها بخشی از استاندارد را پیاده می کنند.

۳. خیلی از عملیات های شبکه باید به صورت بی درنگ و در مسیر داده اجرا شوند. هرچند ورود کنترلر در عملیات های شبکه باعث بهبود تصمیم گیری می شود اما کارایی شبکه کاهش پیدا می کند، زیرا بسته ها می بایست زمان یک rtt تا کنترلر را تحمل کنند. این اتفاق در عملیات های حساس به تاخیر که جنس مسیر داده را دارند، به عنوان مثال عملیات وارسی بسته ها [۸] ، اصلا مناسب نیست لذا کنترلر باعث کاهش کارایی شبکه می شود.

این انگیزه ها، جامعه ی پژوهشی را به سمتی سوق داد تا صفحه های داده ی قابل برنامه ریزی طراحی کنند و شبکه های نرم افزارمحور را فراتر از مفهوم برنامه نویسی صفحه ی کنترل به مرحله اجرایی برسانند. اما منعطف کردن صفحه ی داده، به دلیل اعمال سطح پایین سخت افزاری و نزدیکی به مدار تعبیه شده در لایه ی پایینی، اصلا کار آسانی نیست و استاندارد سازی های متعددی را می طلبد. در ادامه ی این مقاله، به بررسی اصلی ترین و معروف ترین زبان توصیف صفحه‌ی داده یعنی P۴

[4] Software Defined Network (SDN)
[5] Vendor
[6] header
[7] controller
[8] Deep Packet Inspection(DPI)

می پردازیم. بدیهی است که مفاهیم این زبان قابل بسط به زبان های دیگر توصیف صفحه ی داده است، هرچند P۴ تقریبا بخش اعظمی از فضای پژوهشی و صنعتی را تحت پوشش خود قرار داده و تقریبا زبان استاندارد جامعه ی شبکه شده است.

۳ زبان P۴

در این بخش زبان P۴ را معرفی می کنیم و مبانی برنامه نویسی با استفاده از این زبان را بررسی می کنیم. باید توجه داشته باشیم که این زبان، یک زبان یک منظوره [۹] است، بدین معنی که تنها برای برنامه نویسی و هدایت صفحه ی داده ی سوییچ استفاده می شود و هیچگونه کاربردی در برنامه نویسی های عمومی ندارد.در ادامه ی این بخش به معرفی تاریخچه ی این زبان، مدل های منطقی سوییچ، سوییچ های هدف و مبانی برنامه نویسی در این زبان گفته می پردازیم.

۳-۱ تاریخچه

ایده ی اولیه ی این زبان در سال ۲۰۱۳ مطرح شد [۴] و در سال ۲۰۱۴ در مقاله ی [۵] ورژن کامل آن به نمایش گذاشته شد. دو نسخه ی اصلی از این زبان به نام های $P4_{14}$ و $P4_{16}$ وجود دارند که توسعه ی نسخه ی ۱۴ از سال ۲۰۱۵ شروع شد و در سال ۲۰۱۸ متوقف شد ولی نسخه ی ۱۶ از سال ۲۰۱۷ شروع شد و تا کنون در حال توسعه است. در واقع تفاوت مهم نسخه ۱۶ با ۱۴ در این است که نسخه ی ۱۶ کمتر به سخت افزار یا نرم افزار زیرین خود وابسته است و عباراتی که در برنامه نویسی به کار می برد کلی تر است و مخصوص یک سوییچ هدف نیست. به همین دلیل این نسخه از اقبال عمومی بیشتر برخوردار شد و پیشرفت بهتر و سریع تری داشت.

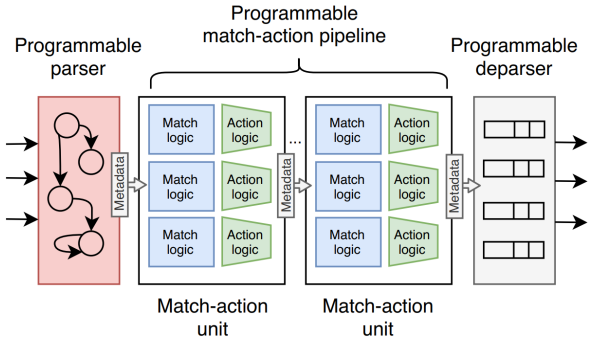
۳-۲ مدلسازی سوییچ

مهم ترین بخش در برنامه نویسی صفحه ی داده مدل سازی منطقی صفحه ی داده است که بر اساس آن زبان مورد نظر پیاده سازی می شود. در واقع هر مدل، یک انتزاع از امکاناتی است که لایه ی زیرین یک سوییچ (سخت افزار یا نرم افزار) در اختیار ما قرار می دهد. در واقع هر الگوریتمی که در صفحه ی داده، تعریف می شود باید در نهایت به شکل اجرایی روی همین مدل تبدیل بشود. در زبان P۴ از مدل PISA [۱۰] استفاده شده است که مدل بسیار خوبی از سوییچ های مدرن است. مدل PISA براساس سه مولفه ساخته شده است:

۱. **Parser**
۲. **Match-Action**
۳. **Deparser**

در واقع مدل PISA به ما این اجازه را می دهد که این سه بخش را برنامه نویسی کنیم. در ادامه به توضیح این سه بخش می پردازیم:

[9] Domain Specific
[10] Protocol-Independent Switching Architecture



شکل ۵: مدل PISA

۳-۲-۱ بخش Parser

در این بخش می توانیم سربار [۱۱] دلخواه خود را طراحی کنیم و آن را از درون یک بسته استخراج کنیم. در واقع این بخش از یک سوییچ به ما این اجازه را می دهد که پروتکل های صفحه ی داده ی جدید معرفی کنیم. این بخش یک بسته ی سریال را از یک ماشین حالت محدود (FSM) [۱۲] عبور می دهد و با استفاده از سربار های معرفی شده، آن ها را استخراج می کند و در داده ساختار های قابل پردازش ذخیره می کند. این داده ساختار ها در مراحل بعدی پردازش سوییچ، اعم از فعالیت های انطباق-عمل (MAT) [۱۳] استفاده می شوند.

۳-۲-۲ بخش Match-Action

این بخش که به طور خلاصه به MAT معروف است، از تعداد زیادی جدول تشکیل شده است که این جداول با یک کلید قابل آدرس دهی هستند و مقداری که در آن ذخیره شده، عملی است که باید روی آن بسته انجام شود. این جداول معمولا با استفاده از حافظه های TCAM [۱۴] پیاده سازی می شود که در آن، دسترسی به یک کلید، در $O(1)$ انجام می شود.

بخش MAT ، نقطه ی اتصال صفحه ی داده و صفحه ی کنترل است. در واقع شکل جدول، نوع کلید ها و اعمال با استفاده از برنامه نویسی صفحه ی داده مشخص می شود اما پر کردن این جداول، وظیفه ی صفحه ی کنترل است.

۳-۲-۳ بخش Deparser

پس از عبور بسته از Parser و MAT ها، اگر نیاز به خروج بسته ها از پورت های سوییچ باشد، سربار ها باید به ترتیب به بخش داده ی بسته اضافه شوندو دوباره به صورت سریال از سوییچ خارج شوند. وظیفه ی این کار بر عهده ی Deparser است.

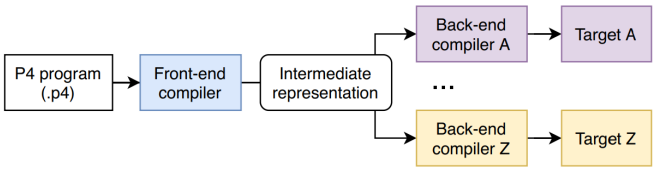
[11] Header
[12] Finite State Machine
[13] Match-Action
[14] Ternary Content-Addressable Memory

۳-۳ سوییچ‌های هدف

یک برنامه ی P۴ ، پس از نوشته شدن باید وارد یک سوییچ بشود و منطق در آن سوییچ پیاده شود. منظور از سوییچ در اینجا یک سوییچ خام و بدون منطق است که صرفا توانایی برنامه ریزی شدن را دارد. به سوییچی که این برنامه ی P۴ قرار است در آن اجرا شود، میزبان یا هدف [۱۵] می گویند. یک برنامه ی P۴ باید توسط یک کامپایلر، به برنامه ی قابل فهم سوییچ هدف تبدیل بشود و سپس سوییچ قابل استفاده بشود. نکته ی مهم این است که به دلیل استاندارد بودن P۴ و متغیر بودن سوییچ های هدف، برای اهداف مختلف، کامپایلرهای مختلفی طراحی شده است، بدین شکل زبان توصیف یکتاست اما تعبیر این زبان برای اهداف مختلف فرق می کند. به صورت خلاصه در P۴ دو نوع کامپایلر وجود دارد که باید با هماهنگی یکدیگر کار کنند. رابط ی این دو کامپایلر در شکل ۶ آمده است و در ادامه توضیح داده شده است:

۱. **کامپایلر Front-End**: این کامپایلر برای همه ی سوییچ های هدف یکسان است و وظیفه ی تعبیر کد نوشته شده توسط توسعه کننده را دارد. خروجی این کامپایلر که فایل میانی است که باید توسط سوییچ ها هدف مصرف شود. کامپایلرهای Front-End توسط جامعه ی P۴ توسعه داده می شود و ربطی به سازندگان سوییچ ندارد. از نمونه های مهم این کامپایلر ها می توان به P۴c اشاره کرد که کامپایلر $P4_{16}$ است و فایل میانی JSON تولید می کند.

۲. **کامپایلر Back-End**: این کامپایلر بستگی کامل به سوییچ هدف دارد. هر سوییچ هدف یک کامپایلر Back-End دارد که شرکت سازنده ی سوییچ آن را توسعه می دهد. این کامپایلر وظیفه ی دریافت فایل میانی و تبدیل آن به زبان قابل فهم سوییچ را دارد.



شکل ۶: رابطه ی کامپایلرهای Front-End و Back-End

از مهم ترین سوییچ های هدف زبان P۴ عبارتند از:

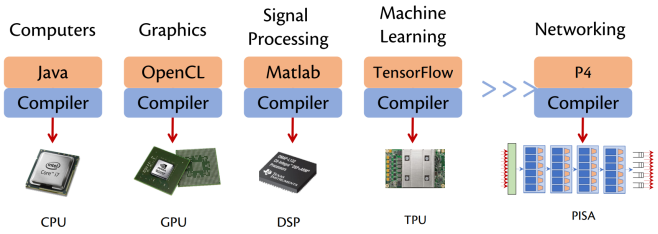
- **سوییچ های نرم افزاری** مثل BMv۲ ، T۴P۴s و p۴c-behavioural
- **سوییچ های FPGA** مثل NetFPGA و P۴FPGA
- **سوییچ های ASIC** مثل Barefoot و Tofino
- **سوییچ های NPU** مثل Netronome

هر کدام از سوییچ های بالا، کامپایلر Back-End خاص خودش را دارد که شرکت سازنده ی سوییچ باید آن را تولید کند.

[15] Target

۴-۳ مبانی برنامه نویسی P۴

در ابتدا، باید روی این نکته تاکید کرد که P۴ یک زبان برنامه نویسی با دامنه ی مشخصی از کاربردها است، یعنی مانند Java ، C++ ، Python و ... برای برنامه نویسی عمومی تولید نشده است. زبان های زیادی در دسته ی زبان های یک منظوره قرار می گیرند که هر کدام برای هدف خاص خود اختراع شده اند، به عنوان مثال در شکل ۷ می توانید نمونه هایی از این زبان ها را مشاهده کنید. در واقع برنامه ای که در



شکل ۷: نمونه هایی از زبان های دامنه-محدود (Domain Specific)

P۴ نوشته می شود، اجزای مختلف یک مدل سوییچ PISA را توصیف می کند و آن را برنامه نویسی می کند. در ادامه به بخش های مهم یک برنامه P۴ می پردازیم:

۱-۴-۳ توصیف سربار

مهم ترین بخش توصیف یک سوییچ در P۴ ، توصیف سربار های پروتکل هایی است که قرار است توسط این سوییچ پشتیبانی شوند. در واقع برنامه نویس می بایست با استفاده از امکانات و داده ساختارهای موجود، داده ساختار های جدید به نام سربار ها رو تولید کند که در طول پردازش بسته در یک سوییچ، در بخش های مختلف آن منتقل می شود، تغییر پیدا می کند و در نهایت Deparse می شود. به عنوان مثال در قطعه کد ۸ ، تعریف یک سربار پروتکل IP را مشاهده می کنید.

```
header ipv4_t {
    bit<4>    version;
    bit<4>    ihl;
    bit<8>    diffserv;
    bit<16>   totalLen;
    bit<16>   identification;
    bit<3>    flags;
    bit<13>   fragOffset;
    bit<8>    ttl;
    bit<8>    protocol;
    bit<16>   hdrChecksum;
    ip4Addr_t srcAddr;
    ip4Addr_t dstAddr;
}
```

شکل ۸: تعریف سربار IP

۳-۴-۲ توصیف Parser

در این بخش برنامه نویس می بایست سربار های احتمالی عبوری از سوییچ را پیش بینی کند و اگر نیاز به پردازش آن ها در سوییچ بود، این سربار ها را در Parser ، Parse کند. همانطور که در بخش های قبلی ذکر شد، یک Parser در واقع یک ماشین حالت محدود است که پس از parse کردن هر سربار، به حالت جدید می رود و سربار بعدی را Parse می کند تا آن که به محتوای داده ی بسته برسد. به عنوان مثال

```
parser MyParser(packet_in packet,
                out headers hdr,
                inout metadata meta,
                inout standard_metadata_t standard_metadata) {

    state start {

        packet.extract(hdr.ethernet);
        transition select(hdr.ethernet.etherType){

            TYPE_IPV4: ipv4;
            default: accept;

        }

    }

    state ipv4 {

        packet.extract(hdr.ipv4);
        transition accept;
    }

}
```

شکل ۹: طراحی Parser برای سربار های Ethernet و IP

در قطعه کد ۹ مشاهده می کنید که در ابتدا سربار Ethernet را Parse کردیم، سپس با توجه فیلد Type ، تصمیم گرفتیم که آیا به حالت پایان برویم یا به حالت Parse کردن سربار IP برویم.

۳-۴-۳ بخش کنترل

در این بخش از برنامه، طراح می تواند روی سربار هایی که در بخش Parser تعریف کرد، عملیات انجام دهد. در واقع در این بخش می برنامه نویس می تواند action تعریف کند. یک action در واقع همان نقش توابع را در برنامه نویسی عمومی بازی می کند.

همچنین می توانیم بخش هایی از کنترل را به صفحه ی کنترل بسپاریم، یعنی از جداول استاندارد تعریف شده در معماری شبکه های نرم افزار محور استفاده کنیم. بدین شکل که جداول و action های مجاز در این جدول را تعریف کنیم و وظیفه ی پر کردن Match ها و Action ها را به صفحه ی کنترلی بسپاریم.

```
control MyIngress(inout headers hdr,
                  inout metadata meta,
                  inout standard_metadata_t standard_metadata) {

    action drop() {
        mark_to_drop(standard_metadata);
    }

    action ipv4_forward(macAddr_t dstAddr, egressSpec_t port) {

        //set the src mac address as the previous dst, this is not correct right?
        hdr.ethernet.srcAddr = hdr.ethernet.dstAddr;

       //set the destination mac address that we got from the match in the table
        hdr.ethernet.dstAddr = dstAddr;

        //set the output port that we also get from the table
        standard_metadata.egress_spec = port;

        //decrease ttl by 1
        hdr.ipv4.ttl = hdr.ipv4.ttl -1;

    }
```

شکل ۱۰: طراحی action های دراپ کردن بسته و ارسال بسته، به علت مروری بودن این مقاله، وارد اعمال جزیی درون یک action نمی شویم.

```
table ipv4_lpm {
    key = {
        hdr.ipv4.dstAddr: lpm;
    }
    actions = {
        ipv4_forward;
        drop;
        NoAction;
    }
    size = 1024;
    default_action = NoAction();
}
```

شکل ۱۱: نمنونه پیاده سازی یک جدول که قابلی های دراپ ، ارسال و NoAction را دارد. این جدول براساس آدرس IP منطبق می شود و سایز ۱۰۲۴ تایی دارد.

۳-۴-۴ بخش Deparser

در Deparser سربار ها دوباره در کنار هم آورده می شوند و به دیتای بسته ملحق می شوند و آماده ی خروج از سوییچ می شوند. در واقع Deparser برعکس کار Parser را انجام می دهد.

۳-۴-۵ Metadata

در هنگام عبور بسته از بخش های مختلف سوییچ، جداول مختلف ، action های مختلف و ... گاها نیاز به خواندن و نوشتن داده بوجود می آید. برای این کار ما نیاز به داده هایی داریم که بتوانیم در حین پردازش بسته، از آن ها بخوانیم و یا در آن ها بنویسیم، به این دیتاها که در خود بسته نیستند و صرفا برای استفاده ی سوییچ بوجود آمده اند، متادیتا می گویند. دو نوع متادیتا داریم:

۱. **متادیتای کاربر** [۱۶] :
متادیتایی است که توسط خود برنامه نویس P۴ تعریف می شود و در حین پردازش از آن استفاده می شود.

۲. **متادیتای ذاتی** [۱۷] :
متادیتایی است که به طور ذاتی در خود سوییچ تعبیه شده و خود سوییچ آن را نگه می دارد، مثلا پورت ورودی، زمان ورود بسته، پورتی که قرار است بسته از آن خارج شود و ... برخی از متادیتاهای ذاتی مهم که در اکثر سوییچ ها پشتیبانی می شوند به صورت زیر می باشد:

- طول صف
- مدت زمان سپری شده در صف
- شماره گروه Multicast
- خطای Checksum
- وضعیت Parser

توجه داشته باشید که متادیتا برای هر بسته بوجود می آید و پس از خروج بسته متادیتای آن بسته حذف می شود.

۶-۴-۳ Extern

یک سری از قابلیت های یک سوییچ، به شکل یک واسط برنامه نویسی(API) [۱۸] و در قالب یک تابع در P۴ در اختیار طراح سوییچ قرار گرفته تا از آن استفاده کند. این کار برای این است که طراح وارد جزئیات طراحی اینگونه عملکرد ها نشود. به عنوان مثال اعمالی مثل دراپ کردن بسته، تولید Checksum ، چک کردن Checksum و اعمالی از این قبلی که صرفا قرار است به عنوان تابع صدا زده شوند، به صورت Extern پیاده سازی شده اند. برخی از Extern های مهمی که اکثر سوییچ های پشتیبانی می کنند:

- تولید و چک کردن Checksum
- خواندن و نوشتن در Register
- Clone کردن بسته
- چرخاندن بسته [۱۹]

لازم به ذکر است که در زبان P۴ ، مانند زبان های برنامه نویسی عمومی، نمی توان Extern های جدید معرفی کرد و به شکل تابع از آن استفاده کرد، چرا که پیاده سازی Extern توسط سخت افزار میزبان P۴ صورت گرفته است و قابل برنامه ریزی توسط نویسنده کد P۴ نیست. اما در برخی سناریو ها که میزبان سوییچ قابل تغییر است، به عنوان مثال در سوییچ های نرم افزاری یا سوییچ های سخت افزاری قابل برنامه نویسی با Verilog ، می توان در میزبان Extern های جدیدی تولید کرد و آن ها را در اختیار برنامه نویس P۴ قرار داد. همانطور که می توان حدث زد، این وابستگی به میزبان، از اهداف P۴ که عدم وابستگی به میزبان است، به دور است و یک نوع نقص محسوب می شود که در بخش تکامل P۴ به آن می پردازیم.

۵-۳ صفحه کنترل در P۴

تا بدین جای کار دیدیم که چگونه می توان از ویژگی های زبان P۴ برای توصیف صفحه ی داده ی ابزاره های شبکه استفاده کرد. در این بخش به این می پردازیم که چگونه می توان از این ابزار های شبکه رابط های اصطلاحا "باز" خارج کرد و آن ها را کنترل کرد. نکته ی مهمی که لازم است دوباره تاکید شود، این است که ابزاره های شبکه، علاوه بر قابل برنامه نویسی بودن، باید قابل کنترل نیز باشند، در واقع مفهوم شبکه های نرم افزار محور به همین خاطر شکل گرفته است. پس ما به یک کانال برای ارتباط با این سوییچ ها نیاز داریم تا بتوانیم اعمال کنترلی را روی آن اعمال کنیم، به عنوان مثال جداول آن را پر کنیم، از وضعیت صف ها مطلع شویم و ...
برای این کار، P۴ چیزی تحت عنوان رابط کاربری برنامه نویسی کنترلی [۲۰] را در اختیار توسعه دهندگان خود قرار می دهد که گاها با اسم API در حال اجرا [۲۱] نیز شناخته می شود. وظیفه ی این رابط برنامه نویسی، این است که امکانات صفحه ی داده را به صورت استاندارد در اختیار صفحه ی کنترلی قرار بدهد،دقیقا مانند Open-flow که این کار برای شبکه های نرم افزار محور انجام می دهد. بدین ترتیب صفحه ی کنترلی بتواند با توان پردازشی ای که دارد، پروتکل های کنترلی را اجرا کند و نهاد های موجود در صفحه ی داده را به صورت متناسب هدایت کند.
صفحه ی کنترلی P۴ اما یک ویژگی مهم دارد که آن را از صفحه های کنترلی سوییچ های عادی یا حتی سوییچ های open-flow جدا می کند و آن هم قابلیت برنامه ریزی صفحه ی داده است. یعنی با استفاده از صفحه ی کنترل در سوییچ های P۴ می توان برنامه ی P۴ را در سوییچ بارگذاری کرد و صفحه ی داده را در حین اجرا تغییر داد. این ویژگی باعث بوجود آمدن یک انعطاف بی نظیر در سوییچ های P۴ می شود که قابلیت برنامه ریزی از بالاترین تا پایین ترین نقطه ی یک سوییچ را در عمل محقق می سازد. البته توجه داشته باشید که یک سوییچ P۴ می تواند بسته تعریف شود، به عنوان مثال توسعه دهنده P۴ می تواند انتخاب کند تا محتوای جداول مسیریابی به صورت ایستا تعریف شوند و هیچ تغییری نکنند. لذا باز یا بسته بودن یک سوییچ به توسعه دهنده صفحه ی داده ی آن بستگی دارد.
در ادامه انواع واسط های کنترلی را از منظرهای مختلف بررسی می کنیم و آن ها را دسته بندی می کنیم. همچنین تعدادی از پیاده سازی های معروف واسط کاربری کنترلی سوییچ های P۴ در عمل را معرفی خواهیم کرد: واسط های کنترلی را می توان از نظر میزان انتزاع و دسترسی به دو دسته تقسیم کرد:

۱. **دسترسی مستقیم به میزبان:** در این رابط ها می توانیم جزئیات

[16] User-Defined Metadata
[17] Intrinsic Metadata
[18] Application Programming Interface
[19] Recirculate
[20] Control Plane Application Programming Interface
[21] runtime-API

میزبان را تا ریزدانگی یک رجیستر تغییر دهیم.

۲. **دسترسی به صفحه ی داده:** در این سناریو کنترلر به چیزهایی دسترسی دارد که توسط صفحه ی داده در اختیار آن قرار داده شده است، نه بیشتر. مثلا ساختار های داده ی مشخصی توسط صفحه ی داده برای کنترلر باز شده است تا بتوان آن را تغییر داد که نمونه ی بارز آن جداول MAT است.

همچنین واسط های کنترلی را می توان از حیث وابستگی به ساختار صفحه ی داده به دو دسته تقسیم کرد:

۱. **وابسته به صفحه ی داده:** این واسط ها کاملا به ساختار صفحه ی داده وابسته است و اگر منطق صفحه ی داده تغییر کند، این واسط های دیگر کارایی ندارند. **مستقل از صفحه ی داده:** در این سناریو صفحه ی کنترلی کاملا مستقل از صفحه ی داده عمل می کند و واسط به صورت استاندارد برای هر منطقی از صفحه ی داده تعبیه شده است.

در آخر، واسط های کنترلی را می توان از نظر مکانی به دو دسته تقسیم کرد:

۱. **کنترلر محلی:** این کنترلر ها در کنار صفحه ی داده پیاده سازی می شود. به عنوان مثال یک واحد پردازشی CPU به همراه ادوات در کنار صفحه ی داده، تشکیل یک سوییچ مجتمع را می دهند که اکثر سوییچ های سنتی نیز به همین شکل می باشند. در این مدل، API در واقع همان توابع C هستند که فراخوانی می شوند. **کنترل از راه دور:** این کنترلر ها در محلی به دور از صفحه ی داده قرار دارند و با رابط های استانداردی به صورت از راه دور، سوییچ را مدیریت می کنند، شبیه سناریوی شبکه های نرم افزار محور. پیاده سازی این روش از طریق پروتکل های ارتباطی اعم از REST-API ، gRPC و انواع پروتکل های لایه کاربرد می باشد.

در ادامه به معرفی سه واسط کنترلی سوییچ های P۴ در عمل می پردازیم:

۱. **P۴-Runtime API:** این واسط کنترلی که معروف ترین واسط کنترلی P۴ است با استفاده از gPRC پیاده سازی شده است. ارتباط کنترلی بین یک سرور gRPC با کلاینت توسط یک تونل TLS رمزگذاری می شود تا امنیت تامین شود. سرور gRPC در سوییچ پیاده سازی می شود و کنترلر تعدادی کلاینت gRPC را در خود نگه می دارد که با سوییچ ها ارتباط برقرار کنند. این پیاده سازی تحت عنوان کتابخانه PI در دسترس همگان قرار دارد. همچنین لازم به ذکر است که کنترلرهای معروف دنیای شبکه های نرم افزار محور نیز از این رابط کنترلی پشتیبانی می کنند، به عنوان مثال Open Daylight و Operating System Open Network افزونه هایی برای کار با p۴-Runtime دارند.

۲. **Barefoot Runtime Interface (BRI):** این واسط کنترلی نیز با زبان C و Python قابل برنامه نویسی است و در سوییچ های Tofino (که در بخش معرفی میزبان ها توضیح دادیم) قابلیت بهره برداری دارد. این رابط کنترلی نیز از gRPC برای پروتکل انتقال استفاده می کند.

۳. **BM Runtime API:** این واسط کنترلی برای میزبان های bmv۲ ساخته شده است و از تکنولوژی Thrift gRPC برای انتقال پیام های خود استفاده می کند. همچنین چون bmv۲ یک سوییچ نرم افزاری است، این قابلیت برای این رابط کنترلی وجود دارد که از طریق پوسته ی برنامه [۲۲] بتوان آن را تغییر داد و برنامه ریزی کرد.

۳-۶ داده ساختارهای پیچیده

سئوال مهمی که مطرح می شود این است که P۴ چه داده ساختار هایی در اختیار ما قرار می دهد تا بتوانیم از حالت ها در حافظه ی سوییچ نگه داری کنیم؟ همانطور که دیدیم، با هر بسته ی ورودی، سوییچ P۴ می تواند داده ساختار های گوناگون اما محدودی را پیاده کند، مثلا Struct و Tuple از گزینه های پیشرفته ای بودند که در اختیار توسعه دهنده قرار دارد. حال اگر قرار باشد متغیر هایی را بین بسته ها نگه داری کنیم، P۴ چه امکاناتی را به ما می دهد؟ این امکانات عبارتند از:

۱. **جدول:**
همانطور که دیدیم، جداول نقطه ی ارتباط صحفه ی داده با صفحه کنترل است و وابسته به بسته ها نیست، بلکه براساس جریان های عبوری تنظیم می گردد.

۲. **رجیستر:**
برای ذخیره سازی داده های دلخواه و بدون قاعده بوجود آمده است. نحوه ی تعامل با رجیستر ها بواسطه ی extern های خواندن و نوشتن است.

۳. **شمارنده:**
شمارنده، همانطور که از اسمش پیداست، داده ساختاری مناسب برای شمارش و نگه داری تعداد است. به عنوان مثال هر جدول به صورت پیش فرض برای هر سطر خود یک شمارنده دارد که تعداد بسته های ورودی به آن را می شمارد.

۴. **مصرف سنج:** [۲۳]
داده ساختاری است برای تنظیم نرخ خروجی بسته ها از پورت های مختلف. در واقع انواع الگوریتم های تخصیص منابع اعم از صف بندی عادلانه را می توان با این داده ساختار پیاده کرد و در طول برقرار بودن یک جریان نگه داشت.

حال مشکل این است که آیا این داده ساختارها برای نیاز های روزافزون شبکه کافی هستند؟ پاسخ قاعدتا منفی است. اما P۴ به صورت ذاتی

[22] Shell
[23] Meter

امکانات دیگری در اختیار ما قرار نمی دهد لذا می بایست داده ساختار های مختلف را به صورت دستی بسازیم. به عنوان مثال برای پیاده سازی یک مجموعه درهم سازی شده [۲۴] می توانیم با استفاده از Extern های درهم سازی و رجیستر ها، آن را پیاده سازی کنیم .

در این بخش ، با مبانی زبان P۴ آشنا شدیم، در بخش بعدی به کاربردهای P۴ در صنعت و در فضای آکادمیک می پردازیم و چند نمونه از پروژه هایی که براساس P۴ بنا شده اند را معرفی می کنیم.

۴ کاربرد P۴

در این بخش به پیاده سازی های مهم براساس P۴ می پردازیم. این پیاده سازی ها عموما در قالب مقالات آکادمیک هستند اما با اضافه شدن یک سوییچ هدف مناسب، به راحتی در صنعت نیز کاربرد دارند. شرکت های بسیار بزرگی مثل Alibaba در حال سرمایه گذاری روی تکنولوژی P۴ هستند و در حال مهاجرت به سمت استفاده از سوییچ های قابل برنامه ریزی هستند.

کاربردهای P۴ می تواند در درسته های مختلفی قرار بگیرد که در این مقاله به موارد زیر می پردازیم:

۱. نظارت [۲۵]
۲. مدیریت ترافیک و کنترل ازدحام
۳. مسیریابی
۴. امنیت شبکه
۵. شبکه های پیشرفته مانند IOT ، NFV ،5G و TSN

۴-۱ نظارت شبکه

در این بخش، کاربردهای نظارتی زبان P۴ را توضیح می دهیم و مشاهده می کنیم که چگونه با ابداع الگوریتم های صفحه ی داده، می توانیم معضل ها و مسائل نظارتی شبکه را سریع تر و بهینه تر حل کنیم.

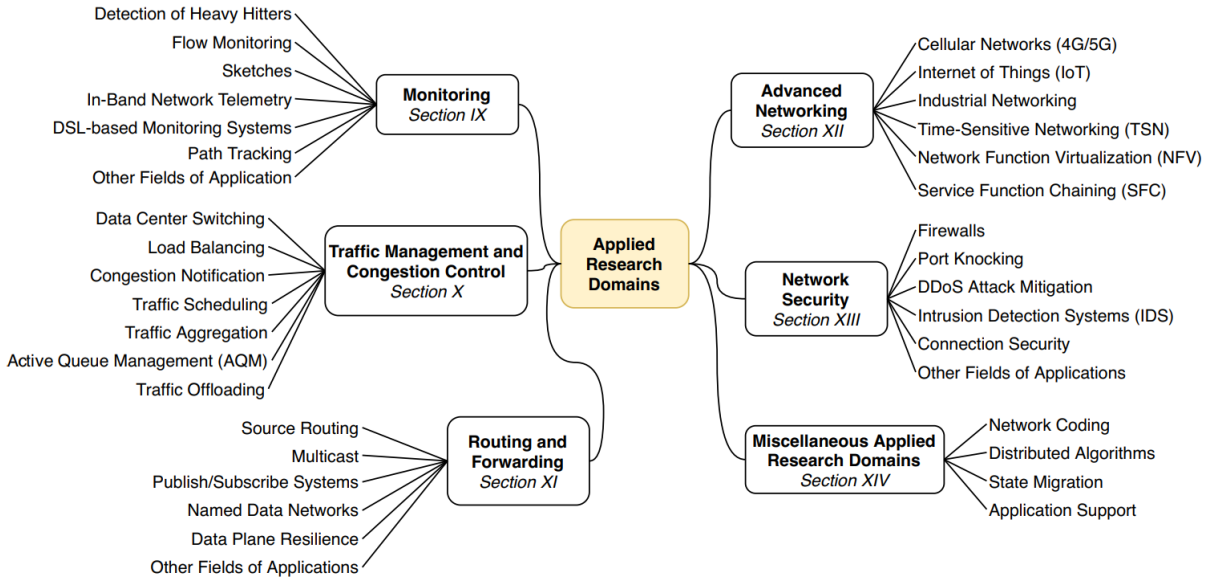
به عنوان مثال اول، مسئله ی تشخیص جریان های بزرگ [۲۶] و تفکیک آن ها از جریان های کوچک [۲۷] یکی از مهم ترین مسائل تصمیم گیری در شبکه است که به مسئله ی "تشخیص جریان سنگین [۲۸] " شناخته می شود. تصمیم گیری اینکه یک جریان، جریان بزرگ و سنگین است منجر به برنامه ریزی منابع شبکه، کنترل ازدحام، خطایابی و مسیریابی بهینه تر می شود. از کاربردهای مهم این الگوریتم می توان به مراکز داده [۲۹] اشاره کرد که در پردازش داده های بزرگ با ابزارهایی مانند Hadoop، مقدار بسیار زیادی جریان بزرگ تولید می کند که باعث ازدحام شدید و از کار افتادن اپلیکیشن های دیگر می شود. علاوه بر مسائل کیفیت سرویس، گاهی اوقات جریان های سنگین، نشانگر حمله ی "منع خدمت (DOS) [۳۰] " می باشند، لذا تشخیص به موقع آن بسیار حیاتی است. در راه حل های سنتی، هر سوییچ، از بسته های عبوری نمونه برداری انجام می دهد و این نمونه ها را به صفحه ی داده ارسال می کرد و صفحه ی داده پس از بررسی نمونه ها، نتیجه را اعمال می کرد. این روش دو عیب اساسی دارد، اول اینکه انتقال این وظیفه ی مهم و حساس به تاخیر به صفحه ی کنترل، باعث بوجود آمدن تاخیر زیادی در جریان می شود، ثانیا با نمونه برداری، حجم زیادی از بسته ها را از دست می دهیم و دقت ما در تشخیص پایین می آید. از آن جایی که جنس این کار، بلادرنگ و از نوع اعمال صفحه ی داده است، با بوجود آوردن جداول درهم سازی [۳۱] و نگه داری حالت جریان ها، می توان عملیات تشخیص را به خوبی انجام داد. نکته ی مهم این است که به خاطر قابلیت برنامه نویسی با استفاده از P۴ ، می توان بین پارامترهای حافظه ی جداول، دقت تشخصی، سرعت تشخیص و پارامتر های دیگر مصالحه [۳۲] برقرار کرد. مقالات زیادی از جمله [۶] [۷] با استفاده از P۴ عمل تشخیص جریان های سنگین را با استفاده از رجیسترها و جداول درهم سازی P۴ انجام داده اند.

علاوه بر "تشخیص جریان های سنگین" ، اعمالی اعم از نظارت و نگه داری آماره های جریان ها نیز مورد نظر راهبرهای شبکه است که با قابلیت های P۴ به راحتی قابل پیاده سازی است.

یکی از مهم ترین کاربردهای P۴ در بهبود کیفیت سرویس و پایش دائمی سوییچ ها، مفهوم "پایشِ درون باندی شبکه [۳۳] " است که به اختصار به آن INT می گویند. این ویژگی به سوییچ این قابلیت را می دهد که وضعیت کارکرد حال حاضر خود، اعم از طول صف، پهنای باند موجود، قدرت پردازشی موجود و ... را به بسته های عبوری خود، به شکل سربارهای استاندارد بچسباند و در میزبان مقصد، این سرباره ها تعبیری از وضعیت شبکه در طول مسیر، به گیرنده می دهند و گیرنده می تواند از این اطلاعات استفاده کند و همچنین به شکل یک بازخورد در بسته های برگستی به سمت فرستنده ارسال کند تا فرستنده، نحوه ارسال (نرخ و اندازه ی ارسال بسته ها) را مطابق با بازخوردها تنظیم کند. به این ویژگی سوییچ ها، "پایش درون باندی شبکه می گویند. در P۴ این قابلیت را داریم که اطلاعات را به دلخواه خود به بسته ها اضافه یا از آن ها کم کنیم. این اطلاعات غالبا همان متادیتاهایی هستند که در بخش های قبل از آن ها نام بردیم. به عنوان مثال در مقاله ی [۱۰] محققان دانشگاه MIT و شرکت Alibaba ، روشی برای جلوگیری از ازدحام در مراکز داده پیشنهاد می کنند که نقیصه های پروتکل های قبلی اعم از TCP را ندارد و می تواند از درصد بسیار بالایی از پهنای باند شبکه بدون بوجود آمدن هرگونه صفی استفاده کند. این کار کاملا براساس INT و در بستر P۴ انجام شده است.

روش INT اگرچه بسیار کارا است و به جای درمان ازدحام، به ما این

[24] Hash Set
[25] monitoring
[26] Elephant Flows
[27] Mice Flows
[28] Heavy Hitter Detection
[29] Datacenter
[30] Denial of service
[31] Hash Table
[32] trade off
[33] In band Network Telemetry



شکل ۱۲: شمای کلی کاربردهای زبان P۴ در دنیای پژوهش براساس موضوع

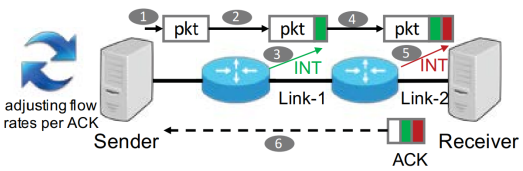
قابلیت را می دهد که ازدحام را پیش بینی کنیم، اما سرباری که به شبکه اضافه می کند، قابل توجه است. مثلا در مسیر های بلندی که یک بسته ممکن است از تعداد بالایی سوییچ عبور کند، حجم سربارهای اضافه شده زیاد می شود با حجم داده های بسته قابل مقایسه می شود. در پژوهشی به نام PINT [۹] اطلاعات INT به صورت احتمالاتی بین بسته های پشت سر هم در یک جریان تقسیم می شود تا جلوی افزونگی اطلاعات گرفته شود و از سربار پایش، کاسته شود. در این مقاله اثبات می شود که PINT با این که احتمالاتی است، اما عملکردی به خوبی INT دارد. مقوله ی INT آنقدر در P۴ پر طرفدار و مهم تلقی می شود که چارچوب [۳۴] جدیدی به P۴ تحت عنوان INT اضافه شده که به برنامه نویسان این قابلیت را می دهد که از INT به صورت آماده استفاده کنند.

۲-۴ مدیریت ترافیک و کنترل ازدحام

روند صنعت در سال های اخیر در مسئله ی کنترل ازدحام بسیار دو جهت گیری عمده داشته است. بخشی از تحولات از جهت وجود کنترلر و مسئله ی پذیرش ترافیک و اختصاص منابع (Int-Serve) به مسئله حمله کرده اند و بخش دیگری از طریق اطلاع رسانی درست اتفاق ازدحام و فهم تولید کننده های ترافیک از ظرفیت شبکه و تغییر ظرفیت ارسال داده ی آن ها مسله را حل کرده اند. در دومین سناریو روش های معروفی توسعه داده شده، به عنوان مثال XCP تلاش موفقی در این زمینه بوده است. با توسعه ی زبان P۴ و امکاناتی که INT در اختیار شبکه قرار می دهد می توان اطلاعات دلخواه را در هنگام عبور از سوییچ ها بارگیری کرد و آن ها را به فرستنده ی جریان اطلاع داد. یکی از مهم ترین مقالات در این زمینه ، مقاله ی HPCC [۱۰] است که با همکاری شرکت Alibaba و محققانی از MIT و Cambridge توسعه داده شده است. در این مقاله به مشکلات اساسی مراکز داده ی Alibaba پرداخته می شود و سعی می شود با HPCC که روش ابداعی جلوگیری از ازدحام است، این مشکلات برطرف شود. سه شاخصه ی مهم مراکز داده ی مدرن باعث شده است که مسئله ی کنترل ازدحام دوباره مورد بررسی قرار بگیرد:

۱. جدا سازی منابع پردازشی از ذخیره سازی که نیازمند تاخیر از جنس میکروثانیه می باشند
۲. پردازش ناهمگون که در آن سخت افزارهایی مانند FPGA ، CPU و GPU که از جنس های مختلفی هستند باید با هم ارتباط برقرار کنند.
۳. کاربرد های مدرن و جدید که پارادایم مراکز داده را تغییر داده، به عنوان مثال کاربردهای یادگیری ماشین، کلان داده ها، استریم بازی و ...

به دلایل فوق الذکر، HPCC با استفاده از INT ، تمام منابع سوییچ های مسیر را به صورت بازخورد به فرستنده گزارش می کند و با این کار، شبکه هیچگاه وارد ازدحام نمی شود و همزمان استفاده از پهنای باند شبکه به بیشینه ی خود می رسد.



شکل ۱۳: شمای کلی از سیستم HPCC بر پایه ویژگی INT سوییچ های P۴

[34] Framework

۳-۴ مسیریابی

مسائل مسیریابی به علت ماهیت پایه ای در شبکه، ابعاد بسیار گسترده ای دارند که در هر یک از این ابعاد می توان از ویژگی های مناسب P۴ استفاده کرد. خوشبختانه P۴ این امکان را به توسعه دهنده می دهد که پروتکل مسیر داده ی شخصی خود را تعریف کند و با استفاده از آن مسیریابی را انجام دهد، کاری که در شبکه های سنتی، حتی در شبکه های SDN کاری غیر ممکن به نظر می رسید. مثلا در مقالات [۱۱] و [۱۲] به ترتیب پروتکل های صحفه ی داده از جنس "مسیریابی منبع"[۳۵] و "مسیریابی قطعه ای"[۳۶] توسعه داده شده و در عمل پیاده سازی شده است. همچنین نوآوری های در زمینه ی چندپخشی[۳۷] انجام شده است که برپایه ی P۴ است، به عنوان مثال در مقاله ی به مسئله ی مقیاس پذیری[۳۸] چند پخشی اشاره شده و الگوریتمی طراحی شده که در آن با تغییر موقعیت دریافت کننده ترافیک، چندپخشی شبکه دچار مشکل نشود. روند دیگری که در مسائل مسیریابی دنبال می شود، شبکه های NDN[۳۹] هستند که در آن، تقاضا ها با توجه مقصد مسیریابی نمی شوند، بلکه با توجه به منابع موجود شبکه مسیریابی می شوند. مثلا در مقاله ی [۱۴] ایده ی شبکه های ndn توسط $P4_{14}$ پیاده سازی شده، منتها با چالش نگه داشتن حالت شبکه در سوییچ های P۴ مواجه شده که بعدا در $P4_{16}$ این مشکل بواسطه ی وجود Register های اضافی و extern های لازم برای خواندن و نوشتن در این Register ها حل شد.

۴-۴ امنیت شبکه

یکی از مهم ترین کاربرد های P۴ ، طراحی ابزار های امنیت شبکه به صورت کاملا منعطف و دلخواه است. براساس موارد موجود در سربارهای بسته های ورودی می توان اجازه ی ورود و انتقال را به یک بسته داد یا آن را دراپ کرد که این مصداق عملی یک دیواره ی آتش[۴۰] است. کارهایی از جمله P۴-Guard [۱۵] و Cofilter [۱۶] از جمله تلاش های محققان برای پیاده سازی یک دیوار آتش کامل برای شبکه های دیتا است.واضحا دیواره های آتش مدرن و با کیفیت نیاز به نگه داشتن حالت جریان های عبوری از خود می باشند که همانطور که در بخش قبل مطرح شد، با بوجودن آمدن مفهوم Register در $P4_{16}$ می توان این دیواره های آتش را پیاده سازی کرد. همچنین یکی از مهم ترین ابزارهای امنیتی در شبکه های کامپیوتری، "سیستم های کشف نفوذ(IDS)"[۴۱] و "سیستم های پیشگیری از نفوذ(IPS)"[۴۲] می باشند. معضل مهم این گونه از سیستم ها در شبکه های سنتی و حتی در شبکه های SDN این است که کل ترافیک باید به یک میزبان ثالث وارد شود و هر کدام از بسته ها به صورت جداگانه مورد بررسی و آنالیز قرار گیرند. حال مسئله این است که این شیوه ی کارکرد سربار مخابراتی و پردازشی بسیار زیادی را به میزبان تحمیل می کند. این سناریو در شبکه های SDN بدتر می شود چرا که هر کدام از این بسته ها باید توسط کنترلر بررسی شوند که خود نقشی اساسی در شبکه دارد و این حجم از ترافیک و محاسبات را نمی تواند تحمل کند. با این تحلیل می توان دریافت که کار IDS و کار IPS یک عمل از جنس صفحه ی داده است که باید در مسیر عبور جریان ها انجام بگیرد، نه در یک میزبان سوم. قطعا پیاده سازی IDS با یک سوییچ P۴ نیازمند Parse شدن بسته ها تا بیت آخر می باشد چرا که در یک ابزار IDS مهم ترین بخش بسته که مورد بررسی قرار می گیرد، بخش Payload آن است. به عنوان مثال در مقاله ی [۱۷] با یک حرکت هوشمندانه، قوانین Snort [۱۸] به ورودی های جداول ،TCAM ترجمه شده است که باعث می شود پردازش بسته ها در مسیر صفحه ی داده انجام بشود. همچنین در مقاله ی [۱۹] عمل IDS به صورت ادغام شده با اعمال صفحه ی داده به کمک NIC انجام شده است، یعنی IDS به صورت کلی در P۴ پیاده نشده بلکه با micro-c در پردازشگر NIC ذکر شده به همراه زبان P۴ ادغام شده است که نتیجه ی آن دستگاه DeepMatch است که در سرعت های بسیار نزدیک به سرعت خط، پردازش ها را انجام می دهد و در حد و اندازه های IDS های صنعتی مورد بهره برداری قرار گرفته است.

یکی از بخش های مهم امنیت شبکه، بحث تونل زدن، امنیت اتصالات منطقی و رمزنگاری است که عمدتا به صورت انتها به انتها صورت می گیرد یا اینکه تنها سوییچ های لبه ای درگیر این کارمی شوند. در شبکه های مدرن اما اطلاعات بسیار ارزشمندی اعم از INT در صفحه های داده در جریان است که تغییر یا مشاهده ی آن ها می تواند اطلاعات خوبی را در اختیار حمله کننده قرار دهد. به این دلایل، نیاز رمزنگاری در سوییچ های P۴ مطرح شده است، یعنی هر سوییچ بتواند به انتخاب خود بخشی از ترافیک خروجی خود را رمزنگاری یا بخشی از ترافیک ورودی خود را رمزگشایی کند. در طراحی اولیه P۴ چنین قابلیتی در نظر گرفته نشده است لذا اخیرا پژوهشگران تلاش های پراکنده ای جهت اضافه کردن چنین قابلیتی به سوییچ های P۴ کرده اند که این تلاش ها در راستای نوشتن Extern های وابسته به میزبان بود. همانطور که در بخش های قبل گفته شد، اضافه کردن Extern های جدید به سوییچ های P۴ ، باید از سمت میزبان صورت بگیرد و به صورت دستی به زبان P۴ اضافه شود. در این راستا کارهایی از جمله P۴-macsec [۲۰] و P۴-ipsec [۲۱] انجام شده است که در آن ها چند نمونه اولیه های رمزنگاری در میزبان bmv۲ پیاده شده و به صورت Extern در اختیار توسعه دهنده P۴ قرار گرفته.

۵-۴ شبکه های پیشرفته

از کاربردهای P۴ ، می توان به بکارگیری آن در شبکه های موبایل اشاره کرد. از آن جایی که شبکه موبایل از دو صفحه کاربر و کنترل تشکیل شده، P۴ را می توان به شکل یک انعطاف در صفحه ی کاربر دید، چرا که در صفحه ی کاربر در شبکه ی موبایل تقریبا برابر با صفحه ی داده

[35] source routing
[36] segment routing
[37] multicast
[38] scalability
[39] Named Data Network
[40] Firewall
[41] Intrusion Detection Systems
[42] Intrusion Prevention Systems

در مفهوم عام شبکه است. به عنوان مثال نهاد SPGW-U در زبان P۴ پیاده سازی شده و همچنین CU و DU که جزوی از شبکه های موبایل ابری نسل جدید هستند، در قالب یک کد P۴ تولید شده اند. همچنین پروتکل GTP-U بارها توسط پژوهشگران مختلف پیاده سازی شده است و پیش بینی می شود که نهاد های این پروتکل در آینده ی شبکه ی موبایل در قالب سوییچ های P۴ بهره برداری خواهند شد. یکی از ویژگی های مهم شبکه ی موبایل، موتور های پردازش سیگنال دیجیتال قدرتمند این شبکه هاست که عمدتا در لبه های شبکه قرار دارند و P۴ عمدتا به تنهایی پاسخگوی این نیاز نیست و لازم است که در کنار یک ابزار پردازش سیگنال به کار گرفته شود. البته پیشنهاد شده است که قابلیت های پردازش سیگنال نیز به پایپ لاین P۴ اضافه شود، اما این ایده هنوز در دست بررسی است.

در شبکه های صنعتی و حساس به تاخیر (TSN) نیز پیاده سازی هایی در زبان P۴ صورت گرفته است. به عنوان مثال در مقاله ی [۲۲] یک سوییچ P۴ وظیفه ی انتخاب یک محرک موتور [۴۳] براساس دستور وارد شده و حالت فعلی سیستم را دارد. در واقع یک ضرب ماتریسی در سوییچ P۴ اتفاق می افتد و نتیجه ی این عمل در بسته نوشته می شود و به فعال کننده ی مورد نظر ارسال می شود.

۵ تکامل P۴

در این بخش به سناریو های محتمل پیش روی زبان P۴ می پردازیم و توسعه های احتمالی آن در سال های آتی اشاره می کنیم. این سناریوها ، سناریوهای واقعی صنعتی هستند که از نیازهای شرکت های بزرگی همچون Intel و مراکز پژوهشی سراسر جهان جمع آوری شده است.

۵-۱ عدم وابستگی به میزبان

مهم ترین بحثی که در جامعه ی P۴ در حال حاضر جریان دارد این است که چگونه می توان وابستگی یک پایپ لاین P۴ را نسبت به میزبان خود کاهش داد. در واقع P۴ برای این وارد دنیای شبکه شده که یک زبان و استاندارد واحد را برای نوشتن ابزارهای شبکه را به متخصصین و توسعه دهندگان معرفی کند. در $P4_{14}$ این امر تا حد خوبی محقق نشد و بسیاری از بخش های زبان کماکان به میزبان وابستگی کامل داشت. در $P4_{16}$ اما سعی شده که این وابستگی ها به حداقل خود برسد، لکن هنوز هم چالش های بسیار زیادی در این زمینه وجود دارد. مهم ترین این چالش ها نوشتن Extern های مختلف است که توسط میزبان به مدل سوییچ P۴ تحویل داده می شود. در واقع Extern ها جعبه سیاه هایی هستند که توسط میزبان تولید شده اند و صرفا رابط ها کاربری آن ها در اختیار توسعه دهندگان قرار گرفته. حال اگر میزبان ها تفاوت بسیار زیادی داشته باشند، Extern های متفاوتی می بینند و این اتفاق بسیار بد است. یکی از مشکلات اساسی P۴ همین وابستگی به میزبان است که به نظر می رسد مشکلی بنیادین باشد و به راحتی قابل حل نباشد. راه حل اولیه برای حل این مشکل توسعه ی مدل سوییچ P۴ از PISA به مدل های پیشرفته تر با extern های بیشتر است که توسعه دهندگان نیاز به نوشتن extern های خود نداشته باشند، اما بدیهی است که این راه حل مقیاس پذیر نیست چرا که مدل سوییچ نمی تواند تا بی نهایت بزرگ شود و این کار P۴ را به سرنوشت Open-Flow دچار می کند.

۵-۲ زبان P۴ در اکوسیستم ابری

در سال های اخیر، با اوج گرفتن پارادایم محاسبات ابری، شرکت ها و ارائه دهندگان سرویس ابری متعددی در سراسر جهان در حال رقابت با هم هستند مثلا، IBM Microsoft، Alibaba، Amazon و ... یکی از نقاط مهم رقابتی این شرکت ها با هم، نحوه ی اتصال بلوک های مختلف ابری مانند واحدهای پردازشی و ذخیره سازی با هم است، به طوری که حدود ۸۰ درصد ارائه دهندگان سرویس های ابری در سراسر جهان، از پروتکل های ذخیره سازی مختلفی استفاده می کنند و هر کدام برای نیازهای خود و با توجه به امکانات سخت افزاری، نرم افزاری و علمی، پایپ لاین ها مختلفی را طراحی کرده اند تا بتوانند با یکدیگر رقابت کنند.

یکی از مهم ترین ویژگی های پروتکل های ذخیره سازی [۴۴] این است که داده ها از سرور های ذخیره سازی به سرور های پردازشی اجاره شده، بدون تاخیر چندان و بدون آن که CPU هدف متوجه این انتقال شود، منتقل شود. برای این انتقال که از دید CPU باید شفاف [۴۵] باشد، نهادهای مخابراتی مختلفی من جمله سوییچ ها و Smart NIC های مختلفی باید عملیات های صفحه ی داده را به بهترین شکل انجام دهند. و نکته ی بسیار مهم این است که تمام این نهاد های میانی و انتهایی باید تحت هدایت و کنترل یک صفحه ی کنترل باشند. در سال های آتی In-tel قصد دارد به کمک زبان P۴ و با واسط کنترلی P۴-Runtime، تمام این پایپ لاین را با P۴ پیاده سازی کند به صورتی که P۴-Runtime از حالت کنترل صفحه ی داده خارج شود و در واقع بخشی از واحد کنترلی و مدیریت ابری [۴۶] شود. البته نگرانی هایی مبنی بر این وجود دارد که این کار P۴ را به سمت تبدیل شدن ب یک زبان عمومی و چند منظوره هدایت کند.

۵-۳ محاسبات تقریبی در P۴

با توجه به نیازهای روزافزون تکنولوژی به انتقال داده به صورت هوشمند و افزایش سطح انتظارات میزبان های انتهایی از عملکرد شبکه ی میانی متصل کننده ی آن ها در محیط های مختلف از جمله مراکز داده، خطوط انتقالی با مقیاس بالا، شبکه های دسترسی و ... ، تولید کنندگان ادوات شبکه، با دو نیاز متناقض روبرو شده اند: نیاز اول پهنای باند بالاتر و تاخیر کمتر و نیاز دوم انجام عملیات پیچیده تر در سوییچ ها smart-NIC های انتهایی که در عمل باعث بوجود آمدن تاخیر و کاهش پهنای باند می شود. این دو نیاز به این دلیل متناقض هم هستند که ظرفیت سوییچ ها به نسبت نیاز ترافیکی مشترکین رشد نداشته است. به همین

[43] Actuator

[44] Storage Protocol

[45] Transparent

[46] Cloud Orchestration

دلیل محققین دانشگاه Harvard در حال توسعه ی P۴ برای پشتیبانی از عملیات هایی هستند که بتوانند پردازش های سنگین سوییچ ها را در پهنای باند بالا پشتیبانی کنند. این عملیات ها برپایه محاسبات تقریبی پردازش هاست و ایده های خود را از تئوری های معروف دنیای پردازش وام گرفته است که عبارتند از: نمونه برداری [۴۷] ، خلاصه سازی [۴۸] فشرده سازی غیرقابل بازیابی [۴۹] ، کدگذاری [۵۰] ، الگوریتم های توزیع شده [۵۱] . در واقع دلیل مهاجرت صنعت به سمت محاسبات تقریبی این است که بخش عمده ای از سوییچ هایی که حتی از P۴ پشتیبانی می کنند، از انجام برخی از اعمال مهم عاجز هستند، به عنوان مثال توانایی انجام محاسبات Floating-Point را ندارند یا به تعداد کافی حالت نگهداری نمی کنند. طبیعی است که این تکنیک ها محاسبات را به صورت تقریبی انجام می دهند و سوییچ ها می توانند بنابر دقت موردنظر، میزان محاسبات را کاهش یا افزایش دهند.

۴-۵ بافر و تایمر

پروتکل های زیادی در صنعت نیاز به داشتن بافرهایی برای ذخیره سازی داده و تایمر هایی برای تصمیم گیری فروارد شدن یا دراپ شدن بسته ها دارند. به عنوان مثال در پروتکل RLC که در شبکه ی موبایل LTE استفاده می شود، بسته ها پس از ورود باعث شروع تایمرهایی می شوند که در این بازه اگر سیگنال به خصوصی وارد شود، بسته ی مورد نظر دراپ خواهد شد. همچنین در NIC هایی که از پروتکل هایی اعم از QUIC یا TCP پشتیبانی می کنند هم برای ارسال دوباره ی بسته ها نیاز به داشتن تایمر و بافر هایی برای ذخیره سازی موقت بسته ها کاملا حس می شود. یکی از تحولات احتمالی P۴ در سال های آینده، اضافه شدن تایمر ها و بافرهای مختلف به زبان P۴ و بدیهتا به میزبان های P۴ خواهد بود.

۵-۵ اتفاقات براساس Clock

برخی از پیاده سازی ها نیاز به ارسال بسته ها در زمان های بسیار دقیق و حساب شده و با دقت یک clock می باشند، به عنوان مثال Smart-NIC هایی که قرار است در شبکه های دسترسی چندگانه فعالیت داشته باشند یا تسک هایی را انجام دهند که به شدت وابسته به تایمینگ دقق می باشد، نیاز به ارسال بسته ها براساس اتفاقات براساس clock درونی [۵۲] می باشند که در حال حاضر زبان P۴ از آن پشتیبانی نمیکند، بلکه صرفا براساس اولویت ها می تواند این بسته ها را در صف هایی قرار دهد. پیش بینی می شود که این ویژگی در آینده ای نزدیک به P۴ اضافه شود.

۶-۵ پردازش موازی

در نگاه اول، عبارت پردازش موازی با عبارت پایپ لاین در یک تناقض ظاهری قرار دارند، چرا که P۴ زبانی است برای توصیف یک پایپ لاین و اتفاقی به صورت موازی در آن انجام نمی گیرد. اما به دلایلی از جمله صرفه جویی در سخت افزار، گاها نیاز است که یک سوییچ که بر روی یک سیلیکون پیاده سازی شده است، چند عمل را به طور همزمان انجام دهد. به عنوان مثال در برخی از سناریوها نیاز می شود که یک سوییچ چندین پایپ لاین را به صورت همزمان پیاده سازی کند، به طوری که هر پایپ لاین، Parser ، جداول و Deparser های خود را داشته باشد. این شکل از طراحی ، با فلسفه پایپ لاین در تناقض نیست و صرفا بسته های ورودی بنابر صلاحدید، ممکن است وارد یک یا چند پایپ لاین شوند.

۷-۵ کامپایلر های نسل جدید

با گسترش استفاده از P۴ ، انواع مختلف کامپایلرهای back-end و front-end در پژوهش ها در حال استفاده هستند، به طوری که دیگر می توان گفت کامپایلر p۴c به طور کامل بر بازار حکمفرما نیست. یکی از کارهای جالب در این زمینه Graph-To-P۴ [۲۳] کامپایلری است که گراف پارسر را تبدیل به کد P۴ می کند. همچنین بهینه سازی های زیادی در فرایند کامپایل P۴ انجام گرفته که عمدتا به صورت مجموعه های نرم افزاری پیش پردازشی P۴ منتشر شده اند. به عنوان مثال کارهایی از جمله P۵ [۲۵] و pcube [۲۴] با هدف بهینه کردن خروجی کامپایلر p۴ و جلوگیری از اعمال اضافه در صفحه داده، انجام شده اند.

۶ نتیجه گیری

در این پژوهش با زبان P۴ که یک زبان توصیف پایپ لاین شبکه است آشنا شدیم. در ابتدا در باب فلسفه ی وجودی این زبان بحث کردیم و در ادامه به بخش های اصلی یک اسکریپت P۴ پرداختیم و در آخر نیز با اشاره به کاربرد ها و افق پیش روی این زبان، این پژوهش را به پایان رساندیم. می توان گفت P۴ همزمان یک قدم به عقب و یک قدم به جلو در صنعت شبکه های مخابرات داده بوده است. یک قدم به عقب از این منظر، که با مطرح شدن پارادایم شبکه های نرم افزار محور، پژوهشگران بر این باور بودند که مسیر داده ی سوییچ ها می بایست کاملا ساده، بدون قدرت پردازشی و به اصطلاح سبک باشند، اما با بوجود آمدن زبان P۴، جامعه ی آکادمیک و صنعتی مجبور به عقب نشینی و برداشتن چند قدم به عقب، به دنیای قبل از SDN شد که در آن سوییچ ها اندکی از پردازش های لازم را در صفحه ی داده خود انجام می دادند. و اما P۴ قدم بزرگ رو به جلویی بود که شاید بر دنیای شبکه، از ابزار های انتهایی اعم از Smart-NIC ها تا نهادهای میانی مانند سوییچ ها و میزبان های تک منظوره مانند IDS ها، محیط شود و عملا باعث بوجود آمدن شبکه هایی کاملا یکنواخت و P۴-based شود که این اتفاقی بسیار بزرگ است. اما قاعدتا این پایان کار نیست و با بوجود آمدن نیاز های

[47] Sampling
[48] Sketch
[49] Lossy Compression
[50] Coding
[51] Distributed Algorithms
[52] Clock-Driven Events

جدید تحت تاثیر شبکه های مدرن، از جمله ۵G ، جامعه ی P۴ نیز ممکن است دستخوش تغییراتی شود و مانند پارادایم SDN ، نیاز به بازنگری و تکامل داشته باشد. آنچه که مشخص است، این است که شبکه های مخابراتی در هر سطحی از عملیات از جمله سطوح پرداشی ، سوییچینگ، امنیتی تا سطوح پایین فیزیکی مانند آنتن ها، حرکتی را به سوی فرهنگ "باز" بودن آغاز کرده تا تولید کنندگان متفاوت بتوانند با ایده های نو باعث شکوفایی تکنولوژی انتقال داده بشوند.

مراجع

[1] McKeown, Nick, et al. "*open-flow: enabling innovation in campus networks*." ACM SIGCOMM computer communication review 38.2 (2008): 69-74.

[2] Hauser, Frederik, et al. "*A Survey on Data Plane Programming with P4: Fundamentals, Advances, and Applied Research.*" arXiv preprint arXiv:2101.10632 (2021).

[3] *What's Behind Network Downtime? Proactive Steps to Reduce Human Error and Improve Availability of Networks*,2008

[4] "Google Presentations: P4 Tutorial," http://bit.ly/p4d2-2018-spring, accessed 01-20-2021.

[5] Bosshart, P., Daly, D., Gibb, G., Izzard, M., McKeown, N., Rexford, J., Schlesinger, C., Talayco, D., Vahdat, A., Varghese, G. and Walker, D., 2014. *P4: Programming protocol-independent packet processors*. ACM SIGCOMM Computer Communication Review, 44(3), pp.87-95.

[6] Sivaraman V, Narayana S, Rottenstreich O, Muthukrishnan S, Rexford J. *Heavy-hitter detection entirely in the data plane*. InProceedings of the Symposium on SDN Research 2017 Apr 3 (pp. 164-176).

[7] Ding, Damu, Marco Savi, Gianni Antichi, and Domenico Siracusa. "*An incrementally-deployable P4-enabled architecture for network-wide heavy-hitter detection*." IEEE Transactions on Network and Service Management 17, no. 1 (2020): 75-88.

[8] Li Y, Miao R, Liu HH, Zhuang Y, Feng F, Tang L, Cao Z, Zhang M, Kelly F, Alizadeh M, Yu M. *HPCC: high precision congestion control*. InProceedings of the ACM Special Interest Group on Data Communication 2019 Aug 19 (pp. 44-58).

[9] Ben Basat, R., Ramanathan, S., Li, Y., Antichi, G., Yu, M. and Mitzenmacher, M., 2020, July. *PINT: probabilistic in-band network telemetry*. In Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication (pp. 662-680).

[10] Li, Y., Miao, R., Liu, H.H., Zhuang, Y., Feng, F., Tang, L., Cao, Z., Zhang, M., Kelly, F., Alizadeh, M. and Yu, M., 2019. *HPCC: High precision congestion control*. In Proceedings of the ACM Special Interest Group on Data Communication (pp. 44-58).

[11] Lewis, Benjamin, Lyndon Fawcett, Matthew Broadbent, and Nicholas Race. "*Using p4 to enable scalable intents in software defined networks*." In 2018 IEEE 26th International Conference on Network Protocols (ICNP), pp. 442-443. IEEE, 2018.

[12] Luo, Long, Hongfang Yu, Shouxi Luo, Zilong Ye, Xiaojiang Du, and Mohsen Guizani. "*Scalable explicit path control in software-defined networks.*" Journal of Network and Computer Applications 141 (2019): 86-103.

[13] Shahbaz, Muhammad, Lalith Suresh, Jennifer Rexford, Nick Feamster, Ori Rottenstreich, and Mukesh Hira. "*Elmo: Source routed multicast for public clouds.*" In Proceedings of the ACM Special Interest Group on Data Communication, pp. 458-471. 2019.

[14] Signorello, Salvatore, Radu State, Jérôme François, and Olivier Festor. "*Ndn. p4: Programming information-centric data-planes.*" In 2016 IEEE NetSoft Conference and Workshops (NetSoft), pp. 384-389. IEEE, 2016.

[15] Datta, Rakesh, Sean Choi, Anurag Chowdhary, and Younghee Park. "*P4guard: Designing p4 based firewall.*" In MILCOM 2018-

ference on Network Protocols (ICNP), pp. 430-435. IEEE, 2018.

[25] A. Abhashkumar, J. Lee, J. Tourrilhes, S. Banerjee, W. Wu, J.-M. Kang, and A. Akella, "*P5: Policy-driven Optimization of P4 Pipeline*," in ACM Symposium on SDN Research (SOSR), 2017.

2018 IEEE Military Communications Conference (MILCOM), pp. 1-6. IEEE, 2018.

[16] Cao, Jiamin, Jun Bi, Yu Zhou, and Cheng Zhang. "*CoFilter: A high-performance switch-assisted stateful packet filter.*" In Proceedings of the ACM SIGCOMM 2018 Conference on Posters and Demos, pp. 9-11. 2018.

[17] Lewis, Benjamin, Matthew Broadbent, and Nicholas Race. "*P4ID: P4 enhanced intrusion detection*." In 2019 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), pp. 1-4. IEEE, 2019.

[18] *https://www.snort.org/*

[19] Hypolite, Joel, John Sonchack, Shlomo Hershkop, Nathan Dautenhahn, André DeHon, and Jonathan M. Smith. "*DeepMatch: practical deep packet inspection in the data plane using network processors*." In Proceedings of the 16th International Conference on emerging Networking EXperiments and Technologies, pp. 336-350. 2020.

[20] Hauser, Frederik, Mark Schmidt, Marco Häberle, and Michael Menth. "*P4-MACsec: Dynamic topology monitoring and data layer protection with MACsec in P4-based SDN*." IEEE Access 8 (2020): 58845-58858.

[21] Hauser, Frederik, Marco Häberle, Mark Schmidt, and Michael Menth. "P4-IPsec: Site-to-Site and Host-to-Site VPN with IPsec in P4-Based SDN." IEEE Access 8 (2020): 139567-139586.

[22] Mai, Tianle, Haipeng Yao, Song Guo, and Yunjie Liu. "*In-Network Computing Powered Mobile Edge: Toward High Performance Industrial IoT*." IEEE Network 35, no. 1 (2020): 289-295.

[23] Zaballa, E.O. and Zhou, Z., 2019, September. *Graph-To-P4: A P4 boilerplate code generator for parse graphs.* In 2019 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS) (pp. 1-2). IEEE.

[24] Shah, Rinku, Aniket Shirke, Akash Trehan, Mythili Vutukuru, and Purushottam Kulkarni. "*pcube: Primitives for network data plane programming.*" In 2018 IEEE 26th International Con-