The eXpress Data Path: Fast Programmable Packet Processing in the Operating System Kernel

ACM CoNEXT 2018

254 citations

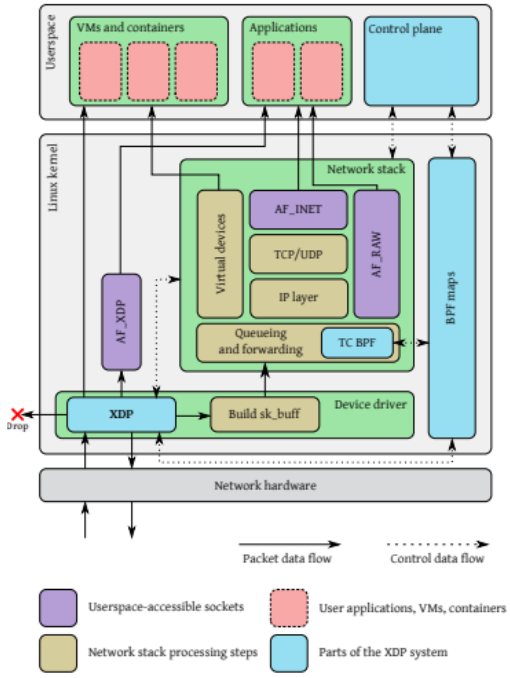
یکی از راه‌های افزایش سرعت پردازش بسته برای اپلیکیشن‌های پردازش بسته در فضای کاربر، دور زدن کرنل به منظور جلوگیری از سربار زیاد به هنگام وقفه سوییچ محتوا بین کرنل و فضای برنامه کاربردی می‌باشد.

اما این روش به دلیل ایزوله سازی برنامه عدم امکان استفاده از مکانیزم‌های امنیتی و مدیریتی (جداول مسیریابی و پروتکل‌های لایه‌های بالاتر) فراهم شده توسط کرنل (به دلیل عدم دسترسی مستقیم کرنل به سخت‌افزار) و همچنین عدم امکان برقراری ارتباط بین برنامه‌های دیگر سیستم‌عامل را به دنبال خواهد داشت. در XDP، کرنل محیطی مجازی در فضای گرداننده دستگاه به منظور اجرای برنامه‌های پردازش بسته در کنار پشته شبکه کرنل فراهم می‌آورد. XDP همانند روش‌های NetMap و PF_RING کرنل را به طور کامل دور نمی‌زند. برنامه‌هایمان را به زبان‌های سطح بالا مثل C می‌نویسیم و به بایت کد eBPF تبدیل شده که خود کرنل آن‌ها را از لحاظ ایمنی بررسی می‌کند (توسط verifier) و به صورت پویا به کد نیتو تبدیل می‌کند. یک محیط مجازی داریم که برنامه eBPF را در فضای کرنل قبل از اینکه کرنل داده بسته‌ها را دستکاری کند، اجرا می‌کند. چون eBPF در فضای کرنل اجرا می‌شود، علاوه بر XDP کاربردهای دیگری هم دارد و برنامه‌های eBPF در پاسخ به یک رویداد در کرنل اجرا می‌شوند.

روش‌های دیگری برای تسریع پردازش بسته علاوه بر روش‌های دورزدن کرنل مثل DPDK داریم:

- ماژول شبکه مخصوص پردازش بسته: به دلیل تغییر API های داخلی کرنل هزینه بر بودند.مثل: OpenVSwitch. اما XDP پایدارتر خواهد بود وAPI های‌آن همانند اینترفیس‌های دیگر قابل استفاده در فضای کاربر خواهد بود (به دلیل پشتیبانی توسط جامعه توسعه دهندگان کرنل). و لازم نیست دیگر مثل ماژول‌ها به پشته شبکه hook شوند.
- دستگاه‌های سخت‌افزاری برنامه‌پذیر: از رابط‌های برنامه‌نویسی که سخت‌افزارهای مخصوص مثل FPGA ارائه می‌دهند، می‌توان استفاده کرد. البته زبان P4 امکان اجرا روی سخت‌افزارهای مختلف را تا حد زیادی گسترش می‌دهد.

بخش‌های اصلی XDP:



شکل 1: بخش‌های اصلی XDP در کنار کرنل

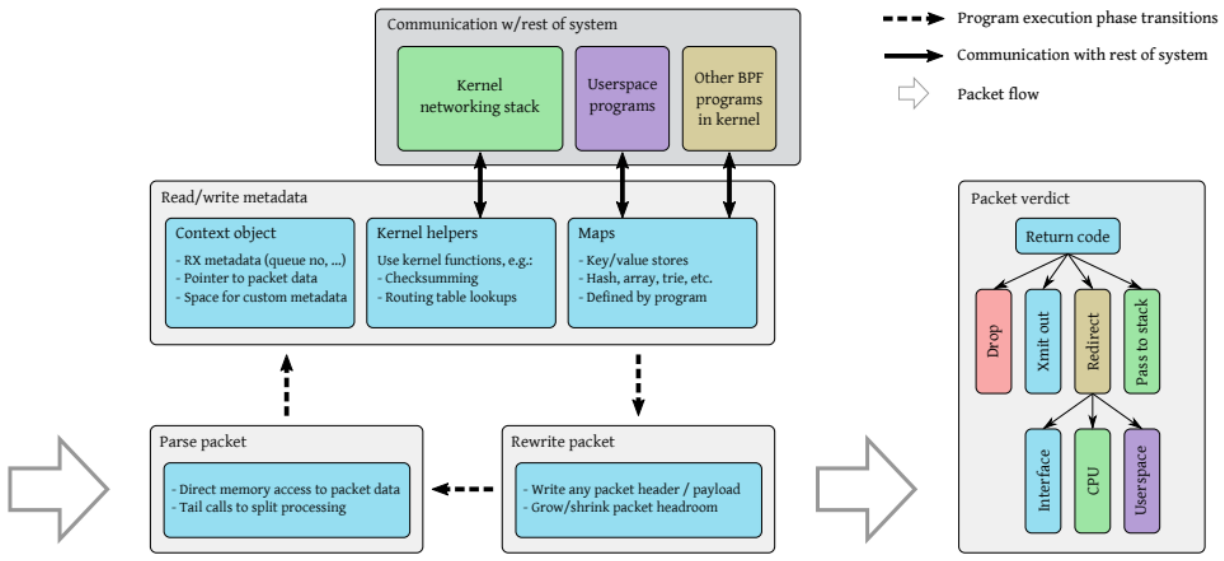
Driver Hook: برنامه XDP که زمانی که یک بسته از سخت‌افزار دریافت شد در محیط eBPF اجرا می‌شود. برنامه به همراه یک context object اجرا می‌شود. این شئ شامل نشانگرهایی به دیتای خام بسته و فیلدهای فرادادهی نشانگر درگاه و صف ورودی بسته می‌باشد. برنامه با parse کردن محتوای داده‌ای بسته اجرا می‌شود و می‌تواند با استفاده از tail call ها پردازش را بین برنامه‌های XDP دیگر تقسیم کند (logical-sub-unit). همچنین XDP می‌تواند در بخش حافظه خصوصی بسته، اطلاعاتی را بنویسد که بخش‌های مختلف سیستم در هنگام طی مسیر بسته بتوانند از آن‌ها استفاده کنند. البته یک BPF map هم هست که می‌تواند داده‌های خود را به صورت دائم در آن بنویسد. همچنین از توابع کمکی کرنل می‌تواند استفاده کند(توسط جامعه کرنل آپدیت می‌شن). برنامه بنا به نیازش می‌تواند محتوای داده بسته‌ها را تغییر دهد. در انتها یک بخش verdict هست که می تواند چهار تا کد مختلف به منظور چهار اقدام متفاوت برای forward بسته بازگرداند: ۱)دورانداختن بسته‌ها ۲)ارسال مجدد بسته به اینترفیس ورودی ۳)هدایت به اینترفیس دیگر مثل vNIC، فضای کاربر با استفاده از سوکت AF_XDP، یا یک پردازنده دیگر با استفاده از کدی که در map ها می‌نویسیم۴) هدایت به استک معمول شبکه توسط TC BPF hook به منظور پردازش قبل از انتقال بسته.

eBPF Virtual Machine: BPF از رجیسترها برای توصیف و انجام عملیات فیلترینگ مختلف استفاده می‌کند. eBPF با افزایش اندازه این رجیسترها و در نتیجه نگاشت راحت تر به رجیسترهای ۶۴ بیتی کرنل و همچنین افزایش تعداد دستورات قابل قبول (دستورات ریاضی و منطقی و امکان صدازدن توابع)، امکان کامپایل JIT به کد

نیتیو به منظور کارایی بهتر را فراهم می‌آورد. با استفاده از کامپایلر LLVM امکان کامپایل کد سی به eBPF هر چند به صورت محدود فراهم است. با اینکه هر محاسباتی را با دستورات eBPF می‌توان انجام داد اما verfier محدودیت‌هایی برای برنامه‌های بارگذاری شده در کرنل تعریف می‌کند تا از آسیب زدن به آن جلوگیری شود. ماشین مجازی eBPF به صورت پویا می‌تواند بخش‌های مختلف برنامه را در صورت نیاز بارگذاری کند.

BPF Maps: امکان ارتباط بین برنامه‌های eBPF مختلف ویا ارتباط با فضای کاربری به کمک این نگاشت‌های دوتایی کلید-مقدار صورت می‌گیرد. هر زمان که برنامه‌های eBPF اجرا می‌شوند، در یک state اولیه هستند که امکان دسترسی به فضای حافظه پایدار program context را ندارند ولی با استفاده از توابع کمکی کرنل می‌توانند به این نگاشت‌ها دسترسی داشته باشند. محتویات این نگاشت‌ها هرچیزی حتی آدرس یک نگاشت دیگر یا tail call ها و... می‌تواند باشد.

eBPF Verifier: به بررسی کد بایتی برنامه به صورت ایستا و اطمینان از کارکرد صحیح آن و عدم آسیب به فضای حافظه‌ای کرنل قبل از بارگذاری آن می‌پردازد. به همین دلیل به منظور عدم وجود حلقه و جلوگیری از افزایش اندازه برنامه به کمک یک ساختار DAG جریان کنترلی برنامه را بررسی می‌کند. باید توجه داشت که وظیفه این بخش اطمینان از اجرای برنامه به بهترین نحو نیست و تنها به حفاظت از داده‌های کرنل به منظور سوءاستفاده می‌پردازد. بارگذاری برنامه‌های نیز سطح دسترسی روت نیاز دارد.



شکل ۲: روند اجرایی یک برنامه XDP

ارزیابی کارکرد: به مقایسه با DPDK (به دلیل کارایی بالا) و نمونه testpmd آن و همچنین استک کرنل می‌پردازیم . پردازنده‌ای که استفاده می‌کنیم قابلیت DDIO را خواهد داشت که به DMA اجازه قرار دادن بسته‌ها

به صورت مستقیم در cache پردازنده را می‌دهد. از دو کارت شبکه با قابلیت ۱۰۰ گیگابیت بر ثانیه استفاده می‌کنیم. از trex برای تولید ترافیک استفاده می‌کنیم. معیارهایی که درنظر می‌گیریم: ۱) نرخ دورانداختن بسته‌ها: ساده‌ترین عملیات برای نشان‌دادن کارکرد پردازشی شبکه، ۲) میزان استفاده از پردازنده: یکی از مزیت‌هایی که XDPدارد گسترش استفاده از پردازنده متناظر با میزان لود بسته‌ها به جای استفاده از هسته‌های اختصاصی می‌باشد، ۳) کارایی forward بسته‌ها نیز چون معمولاً با یک اینترفیس دیگر کارداریم مفید است و نرخ‌گذر و تاخیر را محاسبه می‌کنیم (این آزمایش را با ساده ترین نوع forward یعنی عوض کردن آدرس مبدا و مقصد انجام می‌دهیم).

چون XDP با سایز بسته حداکثری یعنی ۱۵۰۰ بایت می‌تواند تنها روی یک هسته که نصفش نیز بیکار است به سرعت ۱۰۰ گیگابیت برسد، پس سایز بسته‌ها را (چون منظور از کارایی نرخ تعداد بسته‌های پردازشی در هر ثانیه می‌باشد) برابر مقدار کمینه یعنی ۶۴ بایت در نظرمی‌گیریم. برای اندازه‌گیری ارتباط میان تعداد هسته‌های اختصاصی فیزیکی و کارایی پردازش تعداد هسته‌ها را نیز افزایش می‌دهیم.

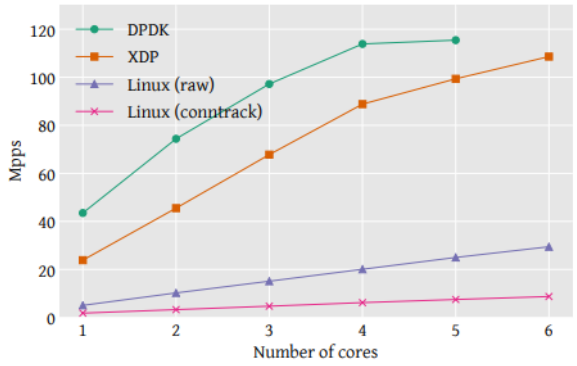


Figure 3: Packet drop performance. DPDK uses one core for control tasks, so only 5 are available for packet processing.

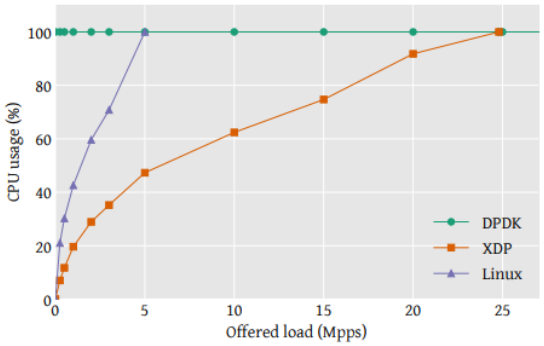


Figure 4: CPU usage in the drop scenario. Each line stops at the method's maximum processing capacity. The DPDK line continues at 100% up to the maximum performance shown in Figure 3.

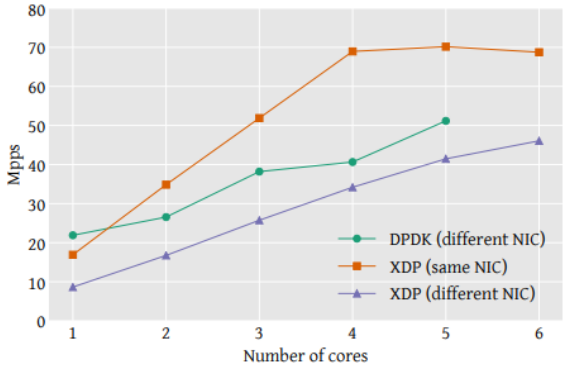


Figure 5: Packet forwarding throughput. Sending and receiving on the same interface takes up more bandwidth on the same PCI port, which means we hit the PCI bus limit at 70 Mpps.

بررسی در کاربردهای مختلف در دنیای واقعی: تنها با استک کرنل مقایسه می‌کنیم چون می‌خواهیم امکان استفاده از XDP در دنیای واقعی را نشان دهیم. مسیریابی لایه ۳، شناسایی منع خدمت inline، متعادل سازی لود لایه ۴.

کارهای آینده و بهبودهایی XDP:

- کاهش محدودیت‌های ایجاد شده به خاطر verifier
- بهبود رابط کاربری و دیباگ: چون eBPF درفضای کرنل اجرا می‌شود، استفاده از ابزارهای مخصوص برنامه‌های سطح کاربر ممکن نمی‌باشد.
- پشتیبانی گرداننده‌های مختلف
- بهبود کارایی نسبت به DPDK
- پشتیبانی از QOS که حتی کرنل نیز AQM را پیاده سازی و استفاده می‌کتد.
- و...

نتیجه‌گیری: پردازش ۲۴ میلیون بسته در ثانیه توسط تنها یک هسته با XDP ممکن می‌باشد. اگرچه مقدار از اپلیکیشن‌های نوشته شده به DPDK ودرکل kernel bypassهای دیگر مثل PF_RING بر روی همان سخت افزار کمتر می‌باشد اما می‌توان از قابلیت‌های مدیریتی و امنیتی فراهم شده توسط کرنل نیز استفاده کرد.