Fast Packet Processing with eBPF and XDP: Concepts, Code, Challenges and Applications 요약 정리

[18반 라민우]

- (!) 격주 보고서 작성에 필요한 내용만을 포함하기 위해 eBPF/XDP에 대한 설명을 최대한 간략하게 정리하고 핵심 정보를 중심으로 서술했다.
- (!) 이전에 전달했었던 요약본에서 더할 수 있는 부분을 추가했습니다.

▶ eBPF란?

다양한 명령어 체계 , 실행 환경(VM 환경)을 리눅스 커널에서 제공하는 시스템 커널 단에 프로그래머가 원하는 프로그램을 적용할 수 있게 하자는 것이 기본적인 목적이다. BPF는 cBPF(구버전), eBPF 로 나뉠 수 있으며, eBPF 기준 레지스터, 스택, 맵을 구현한 BPF machine 체계를 갖추고 있다.

▶ eBPF 프로그램의 실행 과정

[user space] 에서의 기본적인 과정을 설명하자면, 고수준 프로그래밍 언어를 통해 코드를 구성하고, LLVM/clang 을 통해 프로그램을 커널에서 작동할 코드를 생성한다.(기본적으로 많이들 사용하는 방법이지만, 다양한 개발 툴이 존재한다. ex bcc(파이썬으로 개발)) [user space] 에서의 동작에서 있어서 핵심은 컴파일이라고 보면 된다.

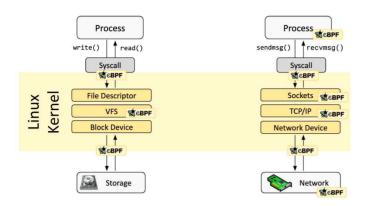
적용할 코드를 실제로 커널 단에 삽입하고 실행하는 과정에서는 ELF eBPF loader 가 관여하는데, 컴파일을 통해 생성된 eBPF ELF 파일에 내재된 일련의 메타 데이터가 제공해주는 정보를 토대로 동작한다.

[kernel space] 에서는 위 [user space] 로부터 받은 BPF 코드를 실제로 동작을 요구할 커널 요소에 적용하기 위한 일련의 검증 & 컴파일 과정을 실시한다. 커널 공간에 삽입할 코드에 대한 검증은 'Verifier'가 담당하며 관련 알고리즘은 엄격한 기준을 갖고 있다는 점이 특징이다. 이후 'JIT compiler'를 통해 kernel 단에서 작동할 수 있는 기계어로 변환하면, 실제로 eBPF 프로그램을 원하는 커널 요소(우리는 XDP 구현을 위해 Network Device Driver에 연결한다) 에 본격적으로 부착할 수 있게 된다.

● 필요하실 때 참고하시면 되겠습니다.

eBPF 프로그램이 작동되기까지의 과정은 이러하다.(논문의 내용에서 살짝 추가한 정도)
eBPF 소스코드 작성(.c) -> 컴파일 -> 오브젝트 파일(.o) -> 커널로 로드(커널로 로드한다는 것이 커널이 이를 수용했다 것은 아니다) -> Verifier(매우 엄격하게 점검) -> JIT(Kernel에서 eBPF 프로그램으로 작동할 수 있도록 컴파일, Just-In-Time 컴파일러 : 프로그램이 동작하는 와중에 컴파일을 해주는 컴파일러) -> Hook에 부착 -> Process

▶ 아래의 그림을 통해 eBPF의 다양한 활용 경로를 시각적으로 확인할 수 있음. (단, eBPF/XDP의 실질적인 동작 과정에 대한 설명보다는 eBPF의 다양한 활용 가능성을 보여주는 간단한 지표로써 보는 것이 맞아보임)



▶ XDP 란?

eXpress Data Path, eBPF 안에 포함된 프로그램 타입중의 하나로, 전통적인 Packet Data를 처리하는 것에서 벗어나 특별한 방법으로 접근하는 방식이라고 보면 된다.

Packet Data를 처리함에 있어서 운영체제의 방식은 다양한 기능을 제공하지만, 그만큼 처리 과정의 효율성과 같은 부분을 재고할 여지가 있다. 이에 따라 kernel bypass를 구현한 기능(DPDK와 같은 toolkit)이 구현된 바도 있지만, 다양한 모듈을 제공받는 방식에서 벗어나 다이나믹하게 특정 하드웨어 장치와 연결하는 방식은 그만큼 포기해야하는 기능도 있다는 점이다.

이러한 단점을 극복하면서 효율적인 Data 처리 방식을 고안한 것이 XDP 라고 보면 된다.

▶ Hook

XDP 프로그램은 네트워크 디바이스 드라이버에서 제공하는 XDP Hook에 의해 동작할 수 있다. Hook에 부착된 XDP 프로그램은 특정 이벤트(패킷이 들어올 경우)에 대해 실행된다. XDP가 특정 이벤트에 대해 실행된다고 했을때의 핵심은 바로 kernel 이 해당 Frame data를 처리하기 전의 과정이라고 생각하면 된다. 전송된 Frame data는 물리 네트워크 장치를 타고 디바이스 드라이버를 통해 커널로 전달되며, 커널은 해당 패킷을 기존의 Network stack(IP, TCP, AF_INET등..)을 통해 처리한다. 그러나 XDP 프로그램이 동작하면 디바이스 드라이버 단에서 해당 Frame data의 처리를 직접적으로 수행할 수 있다. 데이터를 처리함에 있어서 몇 가지의 정해진 방법이 있으며 이를 논문에서는 XDP의 Action 이라고 표현한다. 그리고 내가 추가로 참고한 "The eXpress Data Path: Fast Programmable Packet Processing in the Operating System Kernel" 이라는 논문에서는 "At the end of processing, the XDP program issues a final verdict for the packet" 라고 표현한다. XDP 가 패킷의 처리를 다른 커널 요소들의 개입이 있기 전에 수행할 수 있음을 말하는 것이다.

XDP 외에도 TC(Traffic Control) 계층에도 Hook이 존재한다.(아마 이번 프로젝트에서도 쓰일 것 같은 느낌이 든다.)

XDP가 eBPF 프로그램 타입중 하나이듯, TC도 그러하다. 다만 TC는 XDP보다 한 계층 위에서 작동하기에 XDP가 들어오는 패킷에 대해서는 더욱 빠른 처리가 가능하겠지만, XDP는 수신에 대해서만 작동할 수 있다. 대신 TC는 송수신 모두 다 작동이 가능하다.

▷ 논문의 방향성은 ?

기본적인 eBPF / XDP 에 대한 설명과 함께 간단한 코드 실행과 관련 프로젝트들을 나열하는 형태이다. eBPF 의 전반적인 구성(eBPF Virtual Machine 의 구성, cBPF와의 차이점, JIT과 Verifier의 동작 구성, 간단한 XDP 프로그램 사용법, MAP에 대한 기본 설명 및 사용법, 개발 툴 설명, 이외의 다양한 프로젝트 설명 등이 포함되었다.)

eBPF 핵심 키워드
eBPF > VM, Instruction set, Verifier, JIT, Map, Helperfunction
eBPF Program Type 중 핵심 등장 요소 : XDP, TC