eBPF : 애플리케이션 관점

1. 소개

eBPF는 리눅스 커널 내에서 실행되는 가벼우면서도 빠른 64비트 RISC 형식의 가상 머신입니다. 이 기술은 커널 내에서 신뢰할 수 없는 사용자 정의 코드를 실행할 수 있는 가장 유망한 표준으로 떠오르고 있으며, 높은 성능, 이식성, 유연성, 안전성을 보장합니다. 이러한 주요 이점과 리눅스 커널 내에 풍부한 컴파일러 및 도구 생태계가 구축되어 있기 때문에 eBPF는 산업계와 학계 양쪽에서 광범위하게 채택되고 있습니다. eBPF의 주요 활용 분야로는 모니터링 도구의 성능 향상, 다양한 새로운 보안 메커니즘 제공, 데이터 수집 도구 및 데이터 스크리닝 애플리케이션 개발 등이 있습니다. 이 리뷰에서는 기존의 eBPF 활용 사례와 추세를 조사하여 연구자와 개발자들에게 명확한 로드맵을 제공하고자 합니다. 우선 eBPF에 대한 필요한 배경 지식을 소개한 후 그 활용 사례들을 깊이 있게 다룹니다. eBPF의 잠재적 활용 사례는 매우 다양하지만, 이 리뷰에서는 네트워킹, 보안, 저장소, 샌드박싱과 관련된 네 가지 주요 응용 분야에 초점을 맞춥니다. 각 응용 분야에 대해 해결 기술과 그 원리를 분석하고 요약하여 연구자와 실무자들이 eBPF를 자신들의 설계에 쉽게 도입할 수 있도록 통찰력 있는 논의를 제공합니다. 마지막으로, 혁신적인 eBPF 기술을 완전히 활용할 수 있는 여러 흥미로운 연구 분야를 제시합니다.

최근 몇 년 동안 클라우드 컴퓨팅 패러다임은 인터넷을 통해 사용자에게 컴퓨팅 서비스(소프트웨어, 플랫폼, 하드웨어)를 온디맨드로 제공하기 위해 규모와 복잡성 측면에서 성장했습니다. 클라우드 컴퓨팅의 주요 기반 기술 중 하나인 가상화는 서비스 제공에서 리소스(CPU, 메모리 및 네트워크)의 공유를 가능하게 합니다. 이와 관련하여 컨테이너는 클라우드 서비스 제공에 있어 가장 널리 사용되는 경량 가상화 기술입니다. 또한, 네트워크 가상화는 네트워크 리소스를 효율적으로 활용함으로써 서비스 제공업체의 운영 비용을 줄이는 데 중요한 역할을 해왔습니다. 따라서 현대의 애플리케이션과 서비스는 점점 더 지리적으로 분산된 가상화 인프라와 이종 네트워킹 기술을 활용하여 다양한 네트워크 리소스 요구 사항과 성능 특성을 갖춘 서비스를 사용자에게 적절한 서비스/경험 품질(QoS/QoE)로 제공하고 있습니다.

마이크로서비스와 컨테이너화된 애플리케이션의 등장은 그들의 민첩성과 확장성 덕분에 클라우드 플랫폼에서 여러 서비스의 효율적인 호스팅과 통합을 가능하게 했습니다. 그러나 마이크로서비스의 규모, 이종성, 다양성 및 동적성은 서비스 제공업체가 정확히 모니터링하고 관리하며 문제를 해결하는 것을 복잡하고 도전적인 작업으로 만듭니다. 비슷하게, 네트워크 기능 가상화(NFV)는 네트워크 서비스(방화벽, 트래픽 로드 밸런서 등)의 빠른 배포를 지원하고 허용함으로써 네트워크 관리의 유연성을 도입합니다.

*On-demand : 공급자 중심이 아닌 수요에 따라 제품이나 서비사가 제공되는 것

가상 네트워크 기능(VNF)의 성능 특성과 NFV의 인스턴스를 모니터링하는 것은 네트워크 장애 시 원하는 성능과 복원력으로 서비스 가용성을 보장하는 것을 필수적으로 만듭니다. 네트워크 서비스는 종종 여러 컨테이너를 마이크로서비스로 인스턴스화하여 서비스 기능 체인(SFC)으로 구현됩니다. 이 SFC는 서비스 요구의 무작위적 성격으로 인해 일반적으로 적응적이고 동적입니다. 클라우드 리소스의 보안도 매우 중요하며, 악성코드, 서비스 거부(DOS) 공격, 이상 징후 감지 및 적절한 대응을 위해 심층적인 트래픽 관찰 및 분석이 필요합니다.

그러나 전통적인 모니터링 및 검사 도구는 더 이상 오늘날 끊임없이 변화하는 새로운 환경의 요구를 충족시키지 못합니다. 이는 클라우드 플랫폼에서 임의의 위치에서 동적이고 심층적인 세밀한 트래픽 검사 메커니즘이 부족하거나 이와 관련된 심각한 부하 때문입니다. 따라서 이러한 클라우드 기술은 클라우드 서비스 제공업체가 애플리케이션, 컴퓨팅 및 네트워킹 인프라 및 장치를 시스템 성능을 저해하지 않고 프로그래밍적으로 볼 수 있게 하는 새로운 세대의 확장 가능한 모니터링, 성능 측정 및 고신뢰성 감사 도구를 요구합니다. 또한 이러한 도구는 비디오 회의, 자율 주행, 클라우드 게임, 그리고 차세대 인터넷 패러다임으로 간주되는 메타버스 세계 등 점점 늘어나는 애플리케이션의 서비스 수준 요구 사항을 충족시키기 위해 쉽게 개발 및 배포될 수 있어야 합니다.

최근 몇 년간 등장한 유망한 해결책은 리눅스 커널 코드에 직접 주입되는 필터인 버클리 패킷 필터(BPF)에서 비롯되었습니다. McCanne과 Jacobson에 의해 처음으로 설명된 이 기술은 개선된 버전인 확장 BPF(eBPF)로 알려져 있으며, 커널 내에서 일반적인 처리 이벤트를 처리하는 데 넓은 범위의 능력을 갖추고 있습니다. eBPF는 일반적인 목적의 경량 커널 내 가상 머신(VM)으로, 유연성, 안전성 및 성능의 조합을 제공하여 Linux 커널 내에서 실행 시 확인 가능한 사용자 정의 코드를 개발하고 실행할 수 있는 프로그래밍 가능한 인터페이스를 제공합니다. 따라서 애플리케이션은 런타임에서 eBPF 프로그램을 사용하여 커널을 측정할 수 있으며, 커널 코드를 변경하지 않거나 런타임에서 다른 애플리케이션에 낮은 오버헤드로 영향을 미치지 않습니다. 또한, 리눅스 컨테이너는 클라우드에서의 우선적인 애플리케이션 관리 단위로 채택되었으며, 이는 이 두 가지 중요한 기술이 동일한 환경에서 매끄럽게 작동할 수 있다는 것을 의미합니다. 따라서 eBPF는 클라우드 네이티브 서비스에 대한 이상적인 모니터링 및 강제 메커니즘을 나타내며, 보안 정책, 리소스 및 네트워크 관리와 관련된 실시간 중요한 성능 지표을 수집하는 데 사용될 수 있습니다.

A. 범위 및 방법론

2014년 처음 등장한 이래로 eBPF는 보안 및 네트워킹 응용 프로그램에서의 패킷 처리, 메모리 및 저장소 액세스 지연 감소, 시스템 성능 향상과 같은 다양한 응용 분야에 대해 산업과 학계 양쪽에서 대규모로 채택되었습니다. 그러나 eBPF의 널리 확산에도 불구하고, 신흥 클라우드 응용 프로그램에서 이 혁신적인 기술의 응용 영역을 다루는 연구는 없습니다. 지금까지 2020년에 출판된 하나의 eBPF 관련 설문 조사만 있으며, 이는 빠른 패킷 처리를 위한 eXpress Data Path(XDP)와의

프로그래밍 인터페이스에 대한 기초적인 정보를 제공합니다. 따라서 eBPF의 프로그래밍 세부 사항을 이해하려면 Vieira 등 [21]의 연구를 참고하십시오. 그러나 그 이후로 eBPF 사용은 기계 학습, 숨은 채널 탐지, 침입 탐지 등 다양한 신흥 응용 분야로 확대되었습니다. 따라서 우리의 작업은 [21]에 보충적인 내용이며, 우리의 목표는 응용 프로그램 관점에서 eBPF와 그 주요 작동 원리를 연구하고 보다 넓은 관객에게 제공하는 것입니다. 저희는 이러한 eBPF 선도적인 기술이 다양한 신흥 응용 분야에 유망한 잠재력을 가질 것으로 믿습니다. 따라서 eBPF의 응용 영역을 철저히 연구하여 연구원, 기업 및 개발자들에게 eBPF의 풍경에 대한 명확한 로드맵을 제공하는 것이 중요합니다.

이 연구를 수행하기 위해 주로 Google Scholar에서 'extended Berkeley Packet Filter' 및 'eBPF'와 같은 주요 키워드를 사용하여 검색을 수행했습니다. 찾은 각 논문에 대해 먼저 초록을 확인한 다음, 논문 내용을 스캔하여 eBPF의 네트워킹, 보안, 저장소 및 샌드박싱과 같은 네 가지 주요 응용 분야 중 하나에 할당했습니다. 네트워킹 및 보안은 해당 영역에서 기존 메커니즘을 향상하거나 새로운 접근 방식을 설계하는 것을 의미하며, 저장소는 저장 작업을 가속화하고 로컬 명령을 원격 저장소로 이동시키는 것을 의미합니다. 샌드박싱은 통합된 환경에서 프로그램을 호스팅하고 최적화를 위해 그들의 성능을 분석하는 것에 관한 것입니다. 그런 다음 저희는 저자들 간에 첫 번째 라운드의 결과를 더욱 통합하여 검토할 주요 작품 세트를 확보했습니다. 이 과정은 다음과 같이 분배된 최종 46편의 논문을 결과로 가져왔습니다. 네트워킹 - 21편, 보안 - 17편, 저장소 - 4편, 샌드박싱 - 4편입니다. 또한 한 개의 주요 범주에 겹치는 22편의 교차 논문도 있었으며, 이러한 경우에는 이러한 논문들이 가장 중요한 범주에 포함되었습니다.

각 응용 프로그램 범주에 대해 우리는 솔루션 기술을 분석하고 그들의 작동 원리를 요약하여 eBPF를 사용하여 얻은 성능 향상에 대한 통찰력 있는 토론과 eBPF 응용 프로그램을 위한 포괄적인 가이드북을 제공했습니다. 마지막으로, 검토된 사용 사례를 기반으로 eBPF의 제한 사항을 가리키고 이 신흥 기술을 최대한 활용하기 위한 여러 연구 도전 과제를 강조했습니다.

B. 조직

논문의 나머지 부분은 다음과 같이 구성되어 있습니다. 섹션 II에서는 원래의 BPF와 eBPF를 설명하고 논문 전체에서 언급된 용어와 기술을 간단히 설명합니다. 섹션 III에서는 eBPF 응용 프로그램과 사용 사례를 분류합니다. 그런 다음 섹션 IV에서는 제시된 연구의 사용자 공간 관점을 논의합니다. 섹션 V에서는 동향과 미래의 연구 도전 과제를 설명합니다. 마지막으로, 섹션 VI에서 결론을 내리고 결과를 요약합니다.

2. 배경

이 섹션에서는 논문에서 설명된 기술을 더 잘 이해하기 위해 BPF, eBPF 및 관련 용어를 간략히 설명한다. 더 나은 이해를 위해 Figure 1에 일반적인 아키텍처를 보여주는데, 여기에는 주요 시스템 구성 요소가 나와 있습니다. 아래에는 하드웨어 장비(RAM, 네트워크 인터페이스 카드(NIC), CPU, SSD, GPU 등)가 있습니다. 그 위에 운영 체제가 있으며, 그 핵심 구성 요소인 커널이 있습니다(우리의 경우 리눅스 커널). 이 논문에서 이 상자를 커널 공간이라고 부르며, 이 공간에는 커널 프로세스를 포함한 모든 다른 구성 요소가 포함됩니다. 그 다음에는 사용자 공간이 오며, 여기에는 프로그램과 사용자 프로세스의 실행이 이루어집니다. 마지막으로 사용자 상자가 있으며, 이는 사용자와 시스템 간의 상호 작용을 담당합니다.

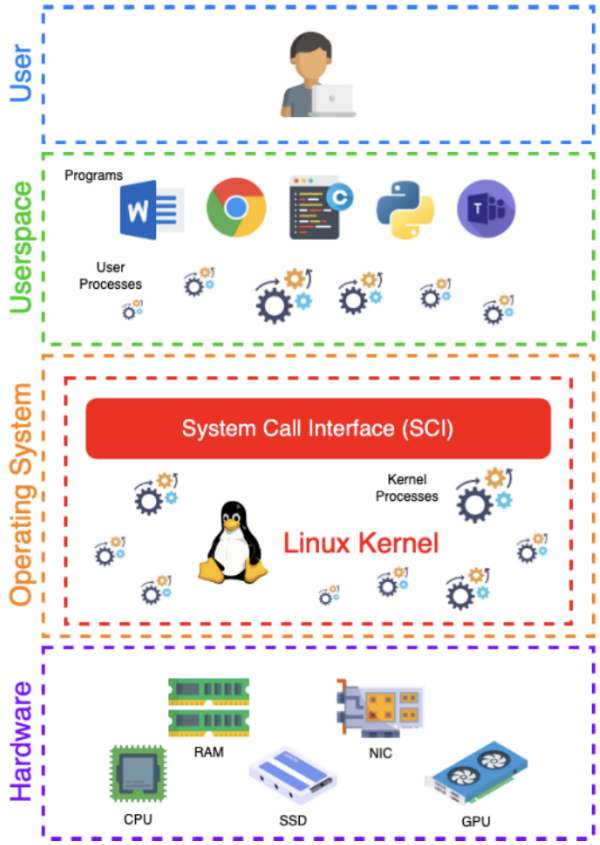


그림 1 일반 시스템 아키텍쳐

설명하는 시스템에 따라 사용자 상자와 사용자 공간의 응용 프로그램이 존재할 수 있습니다. 예를 들어, 전형적인 네트워크의 스위치는 패킷을 처리할 것입니다. 따라서 스위치는 사전에 프로그래밍된 프로세스를 실행하므로 사용자와의 상호 작용이 없습니다. 따라서 이 경우 사용자 상자(위쪽 파란색 상자)와 일반적인 응용 프로그램은 그림에서 제거됩니다.

A. BPF

BPF는 McCanne과 Jacobson에 의해 주로 패킷 검사를 위해 설계된 커널 아키텍처입니다. BPF는 당시 사용 가능한 패킷 캡처 도구와 비교하여 상당한 성능 향상을 제공했습니다. 그 목적은 커널 공간에서 패킷을 사용자 공간으로 복사하는 프로세스를 관리하고 처리하는 것이었습니다. 커널 공간은 운영 체제의 핵심으로, 운영 체제 서비스를 실행하고 관리하는 곳입니다. 반면 사용자 공간은 일반 사용자가 보고 상호 작용하는 곳입니다. 커널은 사용자 공간의 프로세스 간 상호 작용을 처리하며, BPF는 커널/사용자 공간 사이의 경계에 배치된 필터입니다. 목표는 모든 것을 사용자 공간으로 이동하는 것이 아니라 현장에서 필터링하는 것이었습니다. BPF의 기능에도 불구하고, 처음에는 사용자 공간의 네트워크 모니터링 응용 프로그램에게 패킷의 수락/허용 또는 거부/삭제를 허용하기 위해 도입되었습니다. Figure 2는 BPF가 시스템에 어떻게 배치되고 있는지를 명확하게 보여줍니다.

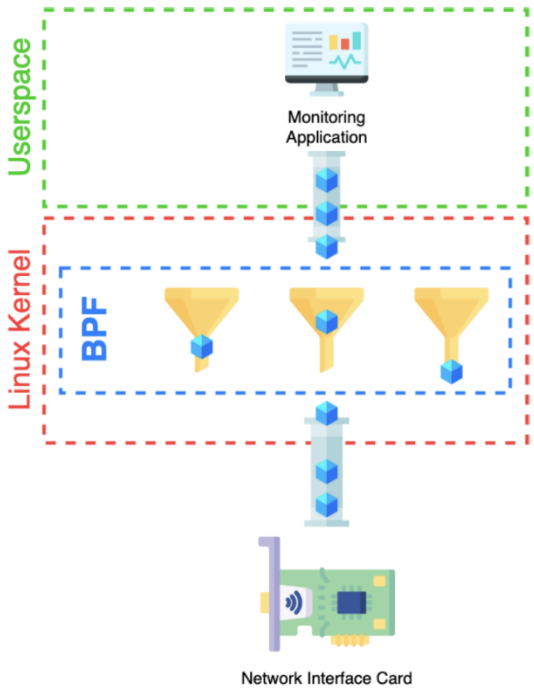


그림 2 BPF 개요

원래의 작업에서는 BPF를 사용하여 SunOS STREAMS 버퍼링 모델 (NIT) [25]보다 성능이 향상되는 것을 증명했습니다. 이는 SunOS의 설계와 같이 패킷을 필터링하기 전에 버퍼로 이동하는 오버헤드를 측정하거나 BPF의 설계와 같이 필터링한 후에 오버헤드를 측정함으로써 이루어졌습니다. 두 설계 간의 차이점은 필터링이 어디에서 이루어지는가입니다. SunOS의 설계는 필터링 전후로 두 개의 버퍼가 있습니다. 따라서 패킷은 버퍼에 저장되어 필터링 프로세스로 이동하고, 필

터링을 통과한 후에 두 번째 버퍼로 이동합니다. 따라서 원하지 않는 패킷도 저장되어 CPU 사이클을 많이 소비합니다. 반면에 BPF는 패킷 스트림에 대한 필터링 프로세스를 현장에서 수행하고 (네트워크 인터페이스에서 수신된 후), 그런 다음 버퍼에 저장합니다. 따라서 필터링 전에 원치 않는 패킷을 미리 저장하는 오버헤드가 없습니다. 저자들은 모든 패킷을 수락하는 필터링 구성을 테스트했으며(모든 것을 버퍼로 이동), 이 경우 BPF는 NIT에 비해 15배 더 적은 오버헤드를 발생시켰습니다. 그런 다음 모든 패킷을 필터링하는 옵션을 테스트했는데, 이 경우 SunOS의 설계에서 패킷이 버퍼에 저장되어 필터링되고 (삭제됨) BPF 설계에서는 패킷이 네트워크 인터페이스를 통과한 후에 필터링되므로 사전 저장 버퍼가 없습니다. 이렇게 함으로써 사전 저장 버퍼를 제거하면 BPF의 이득이 크게 증가합니다.

B. eBPF

eBPF [15]는 고전적인 패킷 필터 BPF의 확장된 버전입니다. eBPF는 커널에서 격리된 프로그램을 실행할 수 있는 가상 머신으로 정의되었습니다. 따라서 새로운 프로그램과 필터링 응용 프로그램을 만들고 이를 안전하게 호스팅할 수 있으며(가상 환경 및 샌드박싱 결과), 커널 코드를 재설계하거나 시스템을 다시 부팅할 필요 없이 직접 커널 코드에 추가할 수 있습니다. 또한, 커널 응용 프로그램에만 제한되지 않습니다. 개발자는 시스템 실행 중에 eBPF 프로그램을 실행하고 실행할 수 있습니다. 고전적인 BPF와는 달리 eBPF는 더 이상 별도의 레이어나 경계 필터가 아닙니다. eBPF는 커널에서 실행되는 작은 하위 프로세스 중 어느 것에나 부착될 수 있으며 거의 모든 곳에서 사용될 수 있습니다. Figure 4는 eBPF가 고전적인 BPF와 다른 부분을 보여주는 삽입 지점을 설명합니다. BPF의 삽입은 사용자 공간과 커널 공간 사이의 경계에만 한정되어 있지만, eBPF는 흐름의 모든 지점에 배치될 수 있습니다. eBPF는 하드웨어(저장소, NIC)와 커널 공간 사이, 커널 내부, 사용자 공간 및 커널 공간 경계, 심지어 사용자 공간에서 실행 중인 응용 프로그램에도 삽입될 수 있습니다.

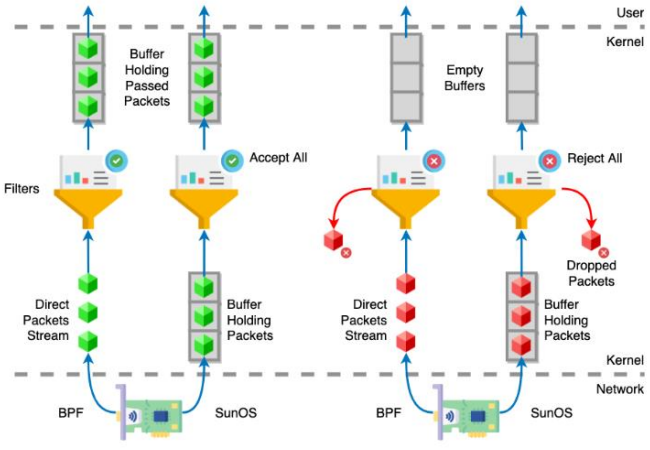


그림 3 "모두 수락"과 "모두 거부" 두 가지 테스트 케이스에서 BPF와 SunOS의 설계를 보여줍니다.

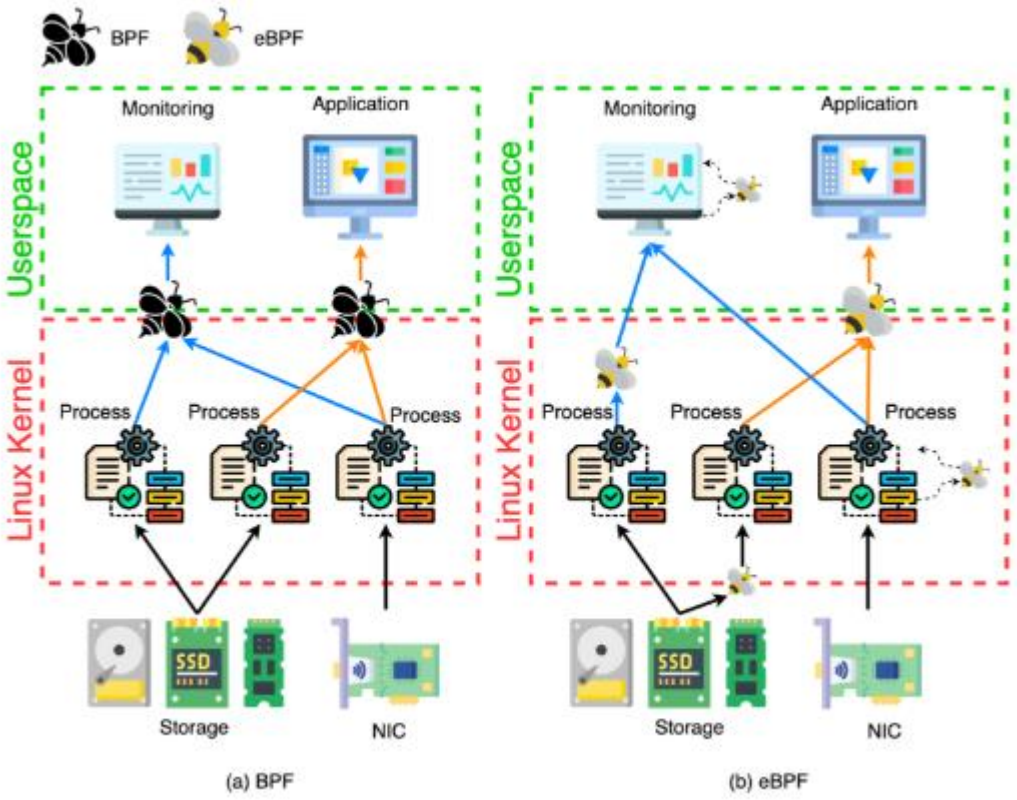


그림 4 "(a) BPF, (b) eBPF의 삽입"

eXpress Data Path (XDP) [26]는 네트워크 하드웨어를 떠날 때 패킷을 가로채기 위해 사용되는 eBPF 기반 데이터 경로입니다. 이것은 커널이 패킷 데이터에 직접 접근하기 전에 패킷을 즉시 가로챕니다. XDP는 패킷을 삭제하거나 사용자 공간으로 전달하거나 정상적인 커널 흐름(네트워크 스택)으로 전달합니다. eBPF 프로그램을 주입하는 것은 사용자 정의 코드를 첨부하거나 추적 프로브를 첨부하는 것을 의미합니다. 추적 프로브는 이름, 권한, 작동 모드, 추적 이벤트 유형 및 캡처된 데이터 양 등에 따라 다양합니다. eBPF는 주로 kprobes 및 커널 추적점 (동적 및 정적 커널 함수의 커널 공간 추적) 및 uprobes (사용자 공간 함수의 사용자 공간 추적)을 사용합니다 [27].

그림 5는 BPF와 eBPF의 아키텍처를 보여줍니다. BPF는 누산기 및 인덱스 레지스터, 암시적 프로그램 카운터 및 임시 보조 메모리로 구성됩니다. 반면, eBPF는 레지스터를 2개에서 11개로 확장했으며, 레지스터의 크기를 32비트에서 64비트로 증가시켰으며, eBPF 엔진에 512바이트 크기의 스택이 도입되었습니다. eBPF 아키텍처에 추가된 전역 데이터 맵 덕분에 데이터 크기나 구조에 대한 제한이 없어졌습니다.

*누산기: 중간 산술 논리 장치 결과가 저장되는 레지스터

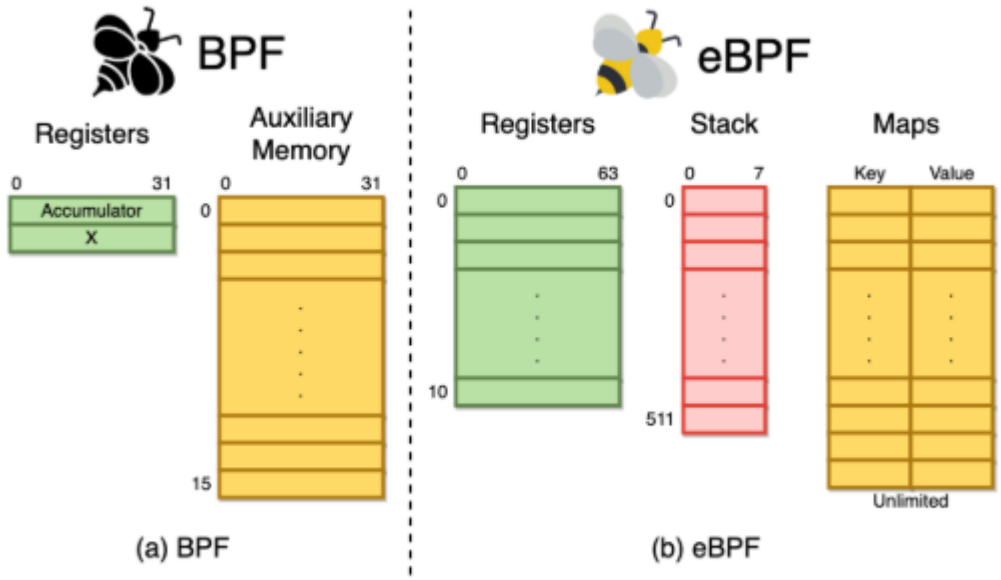


FIGURE 5. (a) BPF architecture, (b) eBPF architecture.

그림 5 (a) BPF 아키텍처, (b) eBPF 아키텍처.

새 버전은 새로운 기능을 제공하면서 요청된 후크에 첨부하기 전에 추가 요구 사항을 갖추어 프로그램의 실행 안전성, 효율성, 확장성 및 호환성을 보장합니다. 그림 6은 eBPF 프로그램을 생성하는 과정을 요약하고, 이러한 프로그램이 커널에 안전하게 주입되고 올바르게 실행되기 위해 필요한 단계를 보여줍니다. 이러한 단계는 다음과 같이 설명됩니다.

1) 검증 단계

프로그램의 안전성을 보장하기 위해 다음 기준에 따라 프로그램을 유효성 검사해야 합니다.

a) 권한: eBPF 프로그램을 보유한 프로세스는 해당 프로그램을 실행할 필요가 있으며, 프로그램이 필요하지 않을 때를 제외하고 필요한 권한을 가져야 합니다.

b) 완료까지 실행: 프로그램은 종료될 때까지 실행되어야 하며, 모든 루프는 보장된 종료 조건이 있어야 합니다.

c) 경계를 벗어난 메모리: 프로그램은 무한한 변수를 사용하거나 경계를 벗어난 메모리에 액세스하는 것을 방지해야 합니다.

d) 크기: eBPF 프로그램은 빠르고 효율적인 실행을 보장하기 위해 작은 크기로 제한되어야 합니다.

e) 유한한 복잡성: 필터링 메커니즘은 Directed Acyclic Graph (DAG)를 기반으로 하며, 이는 여러 실행 경로를 결과로 낳습니다. 검증자는 DAG의 모든 가능한 경로에 대해 제한 시간 내에 실행을 완료해야 합니다.

f) 충돌: 프로그램이 다양한 실행 경로를 시도함으로써 충돌하지 않음을 확인합니다.

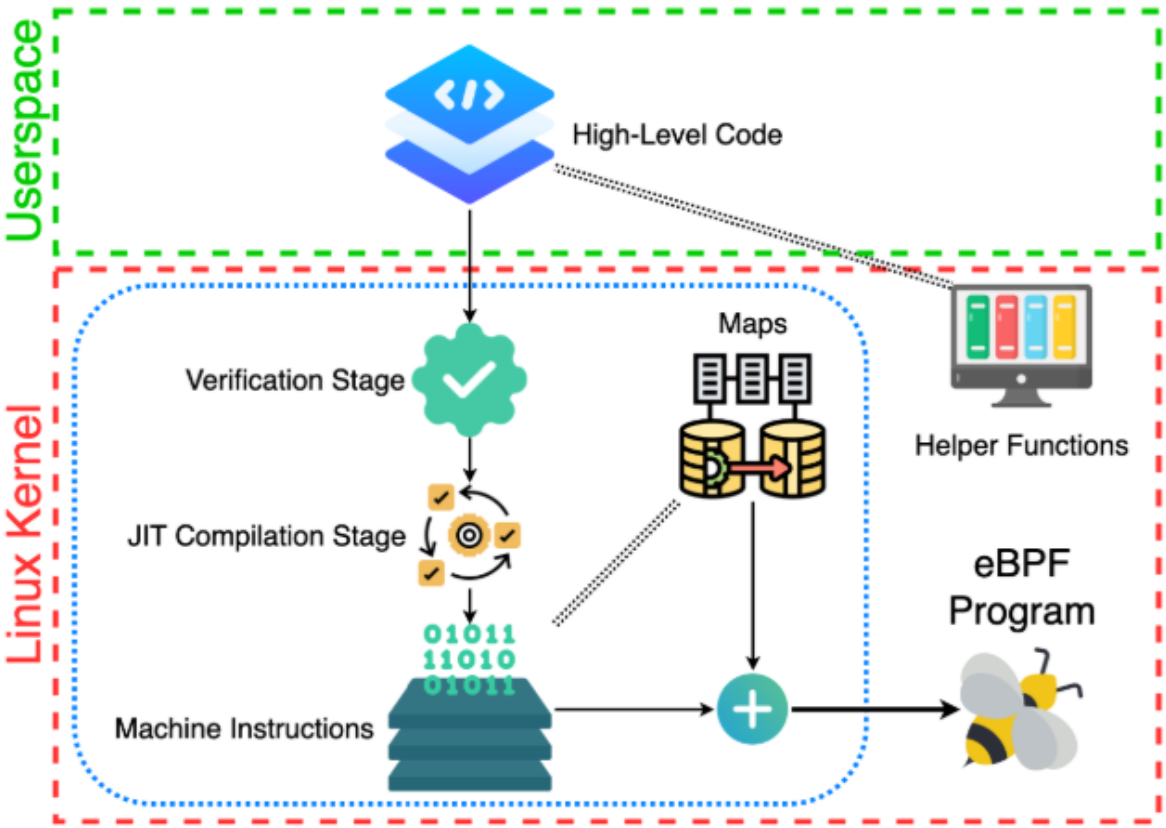


그림 6 eBPF 구성 요소 및 기능 개요

2) JIT 컴파일 단계

커널 공간 자원의 제한과 프로그램의 복잡성 제약으로 인해 Clang이나 Python으로 작성된 eBPF 프로그램은 실행 성능을 최적화하기 위해 올바른 기계 명령어로 변환되어야 합니다. 검증 단계를 거친 후 JIT 컴파일 단계는 JIT 컴파일러를 사용하여 eBPF 기반 프로그램의 원래 버전을 최적화된 기계 바이트코드 버전으로 변환하여 로드된 프로그램의 효율성을 보장합니다.

3) eBPF 맵

BPF에서 프로그램에서 사용할 수 있는 데이터 구조가 제한되어 있어 고급 응용 프로그램을 구현하는 것이 매우 어려웠습니다. eBPF의 중요한 장점 중 하나는 데이터 구조의 다양성입니다. 이 다양성은 eBPF 맵이라는 개념에 기반합니다. eBPF 맵은 eBPF 프로그램 호출 간의 상태를 저장하고 eBPF 커널 프로그램 및 커널과 사용자 공간 애플리케이션 간에 데이터를 송수신하는 방법입니다.

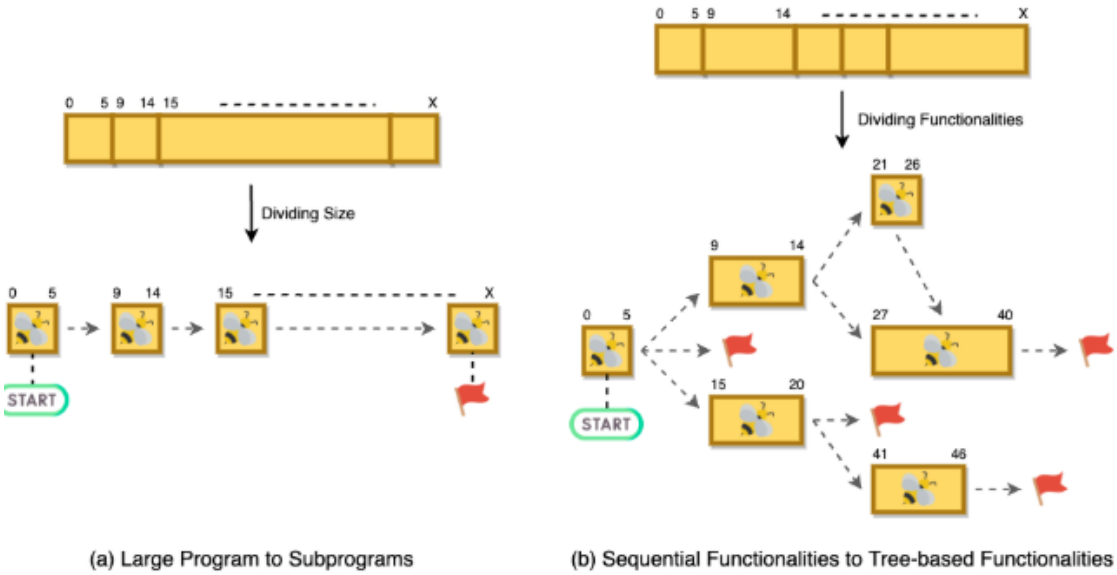
이는 임의의 구조의 키/값 쌍을 저장하는 것을 의미합니다. 사용 가능한 데이터 구조의 몇 가지 예는 해시 테이블 및 배열입니다. 데이터 구조의 완전한 목록은 Linux 커널의 공식 문서에서 확인할 수 있습니다.

4) HELPER FUNCTIONS

eBPF의 프로그래밍 가능성을 일반화하기 위해 커널과 상호 작용하기 위한 안정적인 API가 개발되었으며 이 API에는 도우미 함수가 포함되어 있습니다. 이러한 API가 필요한 이유는 eBPF 프로그램이 일반적인 커널 함수를 호출할 수 없기 때문입니다. 일반 커널 함수를 호출하면 프로그램의 커널 버전이 제한되어 프로그램의 호환성이 복잡해집니다. 가장 많이 사용되는 도우미 함수 중 일부는 난수 생성기, 시간, 데이터 및 eBPF 맵 액세스입니다.

5) TAILING AND FUNCTION CALLS

eBPF는 원래 BPF보다 큰 프로그램을 처리할 수 있지만 여전히 뚜렷한 한계점과 크기 제한이 있습니다. 이 문제를 해결하고 큰 프로그램의 성능을 향상시키기 위해 eBPF에서는 테일링(tailing)과 함수 호출이 사용됩니다. 큰 프로그램은 함수 호출을 명확하게 사용하여 여러 하위 프로그램(함수 기반)으로 분할될 수 있습니다. 각 함수는 필요할 때만 호출되는 별도의 구성 요소를 나타냅니다. 이 기능은 두 가지 방법으로 사용할 수 있습니다. 첫째, 실행 시간 및 크기 제약을 뛰어넘는 큰 프로그램은 정렬된 단계별 하위 프로그램 체인으로 생각할 수 있습니다. 둘째, 하나의 프로그램에 여러 기능을 구현하고 그 중 일부를 실행하는 대신 함수 호출을 사용하여 트리 형태의 기능을 개발할 수 있으며, 이는 프로그램 흐름에 중요한 것만 실행 명령어로 줄이면서 실행됩니다. 테일링과 함수 호출은 작은 프로그램으로부터 큰 응용 프로그램을 구축하고 실행되는 명령어의 양을 줄이므로 eBPF 프로그램의 확장성과 확장성을 향상시킵니다.



(a) Large Program to Subprograms (b) Sequential Functionalities to Tree-based Functionalities

그림 7 Tailing 및 함수 호출 사용 사례. a) 큰 프로그램을 단계적 하위 프로그램 체인으로 나누기. b) 순차적 기능을 트리 기반 구조로 나누기.

3. 응용 및 사용 사례

여러 논문들이 eBPF와 그 기능에 대해 다루었으며, 다른 논문들은 특정 사용 사례나 응용 프로그램을 논의하여 eBPF를 사용했을 때의 성능 향상을 비교했습니다. eBPF는 학계와 산업에서 널리 채택되고 있습니다. 이 섹션에서는 각 논문의 목적에 따라 이러한 주요 분야들로 논문을 나누어 설명합니다. 네트워킹 측면에서 eBPF는 주로 성능 향상, 데이터 패킷 캡처 및 필터링, 네트워크 흐름 제어에 사용됩니다. 보안에서는 기존 보안 메커니즘을 향상시키고 보안 문제를 해결하는 새로운 디자인을 만드는 것이 목표입니다. 스토리지 관련 사용 사례는 메모리 및 스토리지(하드 디스크, SSD, NVMe) 속도와 성능을 향상시켜 데이터 이동을 줄이는 데 중점을 둡니다. 로드맵의 마지막 부분은 eBPF 호스팅을 위한 더 나은 격리된 환경을 구축하는 것에 중점을 둡니다. 아래에서 각 응용 프로그램 도메인과 해당 주요 사용 사례를 설명합니다.

A. 네트워킹

네트워킹 하드웨어는 컴퓨터 네트워크에서 장치 간의 통신과 상호 작용을 담당합니다. 소규모 홈 네트워크, 회사 네트워크 또는 인터넷에서 사용 가능한 온라인 서비스 등 여러 유형의 네트워크가 있습니다. 전형적인 네트워크는 라우팅 및 스위칭 장치, 로드밸런서, 허브 및 데이터 서버로 구성됩니다. 언급된 모든 장치는 대량의 패킷을 처리합니다. 버퍼 및 메모리 제한으로 인해 네트워크에서 트래픽이 많은 경우 패킷이 삭제 될 수 있으므로 기업은 더 많고 좋은 장비를 가져야 할 수 있습니다. eBPF 기술은 추가 하드웨어 설치 없이 네트워크 기능을 개선하기 위한 새로운 기회를 제공합니다. 이 섹션에서는 eBPF를 사용하여 패킷 처리 속도를 향상시키고 네트워킹 장치의 부하를 줄이고 관리하는 데 사용되는 작업에 대해 논의합니다.

1) IPTABLES 기반 eBPF

리눅스 시스템에서 전형적인 방화벽은 IP 및 소스/목적지 포트를 기반으로 들어오는/나가는 트래픽을 처리하는 iptables에 기반합니다. 예를 들어, 보안을 강화하는 가능한 iptables 규칙 중 하나는 모든 나가는 트래픽을 허용하고 모든 들어오는 트래픽을 거부하는 것입니다. iptables는 인터넷에 보이는 IP 주소/포트 또는 로컬 네트워크 또는 서브넷에 대해 특정 호스트(IP 주소)를위한 일련의 포트를 열거나 열지 않는 데 사용됩니다. Deepak et al. [29]은 eBPF 기반 iptables 체계를 설계했습니다. 그들은 eBPF 맵에 새 규칙을 값으로 변환하여 (이전 iptables 규칙) 주어진 입력 테이블 규칙 세트를 eBPF 기반 버전으로 변환하는 일반적인 방법을 제안했습니다. 따라서 이러한 변환 모델은 이전 장치의 구성을 새로운 장치의 구성으로 마이그레이션하기위한 사용자 지정 코드를 재구현하거나 유지 관리하는 문제를 극복하는 데 도움이 되었습니다.

2) VM 간 트래픽 모니터링

모니터링 도구 및 패킷 분석기는 트래픽을 설치된 위치로 보내야 합니다. 전통적으로 각 응용 프로그램 및 도구는 별도의 VM에 별도의 서버에서 설치됩니다. 이러한 도구의 위치로 트래픽을 보내는 전통적인 방법은 Test Access Port (TAP) 장치를 사용하는 것이었습니다. 이것은 서버 상단

에 설치되는 스위치와 유사한 하드웨어 장비로 트래픽을 복제하고 한 가지 복사본을 원하는 응용 프로그램 서버로 보내고 다른 하나를 모니터링 서버로 보내는 것입니다. 현재 서버에서 여러 VM을 실행할 수 있으므로 일반 응용 프로그램 VM과 모니터링 응용 프로그램 VM, 이전 TAP는 도움이 되지 않습니다. 따라서 TAP의 새 버전인 가상 TAP (vTAP)이라는 새로운 버전이 설계되었습니다. vTAP은 서버에 설치되어 가상 스위치를 사용하여 트래픽을 복제하고 여러 VM에 보내는 것입니다. vTAP의 문제는 호스트 머신 리소스의 소비로 인한 성능 저하 일 수 있습니다. Hong et al. [30]은 eBPF 기반 vTAP를 설계하여 들어오는 트래픽을 복제하고 원하는 VM 및 모니터링 VM으로 다시 보내도록 설계했습니다. 이들의 설계를 Open vSwitch (OVS) [31]와 비교 했을 때 패킷 초당 (PPS), 수신기 처리량 (RX) 및 CPU 사용량 면에서 큰 향상이 있었습니다.

3) 부하 분산

데이터 센터의 확장성과 데이터 이동이 부하 분산의 중요성으로 이어집니다. 작은 향상도 데이터 센터의 성능을 크게 향상시킬 수 있습니다. Chen et al. [32]은 기계 학습(ML)을 기반으로 한 부하 분산기를 구축하는 가능성을 입증했습니다. ML 모델을 구축하기 위한 데이터 수집 단계는 eBPF 기능을 활용하여 실행 중인 시스템 데이터를 수집합니다. 이 작업은 리눅스의 완전 공정 스케줄러 (CFS) 대신 eBPF로 수집된 데이터를 사용하여 새로운 ML 모델을 구축하는 효율성을 보여주었습니다.

4) 네트워크 기능 가상화에서 사용되는 커널 내 처리 프레임워크

네트워크 기능 가상화(NFV)는 가상화 기술을 사용하여 네트워크를 프로그래밍 가능하게 만들어 패킷 전달 및 라우팅에 대한 네트워크 장비의 유연성을 높이는 개념입니다. Miano et al. [35]은 커널에서 실행되는 네트워크 기능을 위한 소프트웨어 프레임워크 인 Polycube를 제안했습니다. 그들의 목표는 eBPF를 사용하여 커널 내 패킷 처리 응용 프로그램에 NFV를 활용하는 것이었습니다. 가장 잘 알려진 NFV 프레임워크는 커널 바이패스 접근 방식을 사용하여 커널 내 처리 대신에 사용됩니다.

5) 네트워크 기능 가상화에서 사용되는 SRv6 기능 프레임워크

소프트웨어 정의 네트워킹(SDN)과 NFV의 발전으로 네트워크 노드에서 프로그래밍 가능한 인터페이스를 지원할 것으로 예상됩니다. NFV의 빠른 혁신 요구 사항 중 하나는 동적으로 적용되어야 하며 네트워크 운영자가 새로운 기능 통합에 유연해야 한다는 것입니다. 세그먼트 라우팅 (SRv6)은 네트워크 기능을 지원해야 하는 프로토콜 중 하나입니다. [40]에서는 SRv6에서 네트워크 기능을 구현하기 위해 eBPF 가상화를 활용하는 프레임워크가 제안되었습니다. 제안된 프레임워크는 네트워크 운영자가 자체 네트워크 기능을 eBPF 프로그램으로 인코딩하고 패킷 처리 중 자동으로 실행할 수 있도록 하는 헬퍼 기능을 제공했습니다.

6)인밴드 네트워크 텔레메트리

인밴드 네트워크 텔레메트리(INT)는 옛날 알려진 네트워크 텔레메트리에 해당하는 개념의 혁신

입니다. 텔레메트리는 네트워크 상태에 대한 가시성과 의미 있는 통찰력을 얻기 위해 원격으로 네트워크 정보를 수집하고 처리하는 자동화 프로세스를 의미합니다.

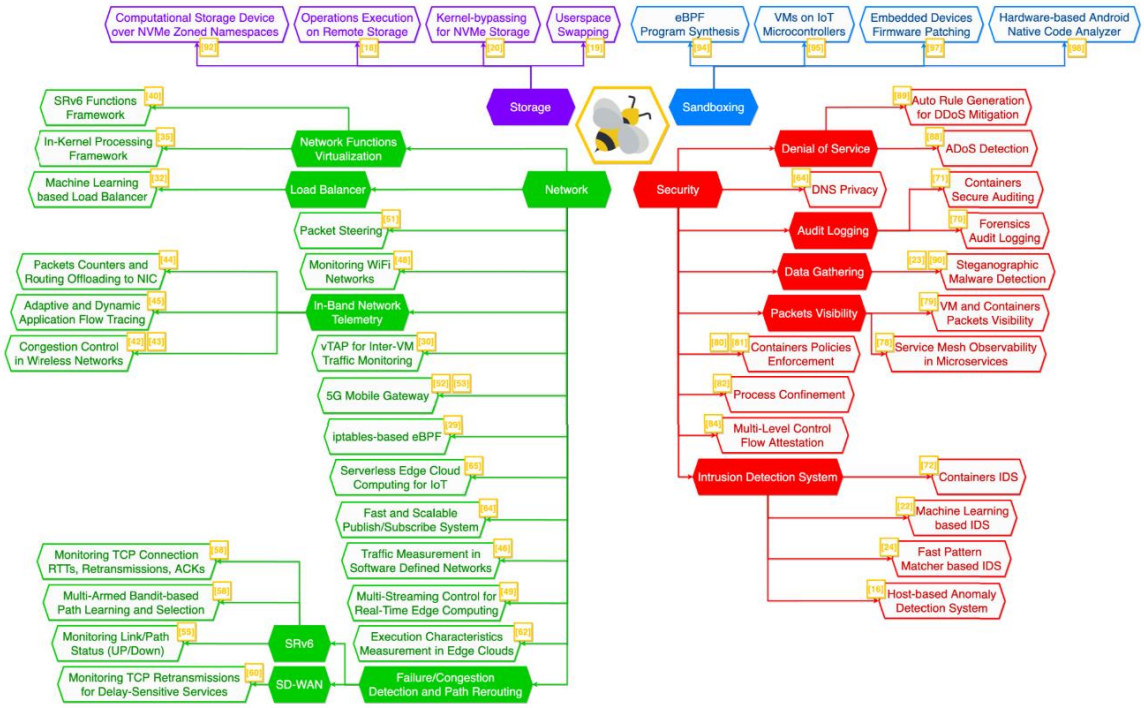


그림 8 eBPF 로드맵

INT는 패킷 수준 및 작업 수준에서 네트워크 재배치를 가능하게 합니다. Bhat 등 [42]은 무선 네트워크에서 TCP의 문제를 논의했습니다. 현재 TCP 프로토콜은 TCP 확인에 의존하며, 특히 무선 매체가 혼잡할 때 잘못된 결정을 초래할 수 있습니다. Bhat 등은 이 문제를 처리하기 위해 INT 데이터와 패킷의 윈도우 크기 사이의 관계를 사용하는 새로운 알고리즘을 설계했습니다. 이 관계를 유도하기 위해, 그들은 eBPF에 의해 추적된 INT 엔드 투 엔드 옵션의 실시간 TCP 데이터를 사용했습니다. 결과는 기존 알고리즘보다 7배 이상의 처리량 향상을 보여주었으며 링크 손실이 20% 미만으로 발생했습니다. 마찬가지로 Dong 등 [43]은 네트워크 혼잡을 조기에 감지하는 데 eBPF 기능을 시연했습니다.

SmartNIC을 통한 텔레메트리 기반의 네트워크 모니터링 응용 프로그램 설계가 가능해졌습니다. Abranches 등 [44]은 SmartNIC로 특정 정보와 계산을 오프로드하기 위해 eBPF를 기반으로 프로토 타입을 구현했습니다. eBPF 맵은 SmartNIC 통계에 액세스할 수 있습니다. 그들의 체계는 통계 카운터 위에 HyperLogLog (HLL) 알고리즘을 사용하여 고유한 흐름을 찾고 구별하기 위해 경로를 분류합니다. 저자들은 또한 IP-to-application 라우팅 테이블을 기반으로 패킷을 원하는 응용 프로그램으로 라우팅하기 위해 SmartNIC에 오프로드되고 저장되는 라우팅 유닛을 설계했습니다.

마지막으로, Sommers와 Durairajan [45]이 소개한 인밴드 네트워크 측정은 인밴드 네트워크 텔레메트리의 대안입니다. 그들은 활동적이고 동적인 인밴드 응용 프로그램 플로우 추적을 위해 ELF라는 오픈 소스 프로젝트를 개발했습니다. 현재 모니터링 도구의 주요 문제 중 하나는 네트워크 운영자가 추적할 응용 프로그램 경로를 정의해야 하므로 중간 플로우 경로 변경시 실행 가능하지 않다는 것입니다. 유사하게, 부하 분산으로 선택된 경로가 응용 프로그램의 처리량에 영향을 줄 수 있습니다. ELF는 일반적인 응용 프로그램 패킷 흐름과 함께 프로브를 첨부하고, 그런 다음 자체적으로 동적으로 조정합니다. 이는 새로운 응용 프로그램의 흐름을 구성하고 네트워크의 처리량을 측정하여 새로운 측정 프로브를 생성하고 새 경로에 삽입하기 위해 주기적으로 반환된 패킷의 사본을 분석함으로써 수행됩니다.

7) 소프트웨어 정의 네트워크에서의 트래픽 측정

최근 소프트웨어 정의 측정(SDM)은 소프트웨어 정의 네트워킹(SDN)에서의 트래픽 흐름 모니터링과 관리를 나타내기 위해 도입되었습니다. Zha 등 [46]은 Open vSwitches(OVS) [31]의 배포에 의존하는 데이터 센터를 위한 다양한 디자인과 프레임워크를 연구했습니다. 저자들은 UMON [47]의 아키텍처적 디자인을 논의했으며, 이는 OVS의 원래 아키텍처를 유지하면서 모니터링 테이블을 API를 사용하여 제어하고, 모니터링(예: 패킷/바이트 카운터)을 유저스페이스에서 전달에서 분리하는 유연성을 도입합니다. 또한, 저자들은 다섯 가지의 새로운 디자인을 제안했는데, Flow CAPture scheme(FCAP)과 Sketch based MONitoring(SMON)은 모니터링 기능을 경로상이나 경로외에 구현할 수 있는 옵션을 제공합니다. 마지막으로, eBPF를 기반으로 한 디자인을 제안했습니다. 경로는 패킷 수신부터 OVS를 빠져나가는 시간까지 사용되는 커널 경로를 의미합니다. FCAP/SMON on-path는 커널 플로우 캐시 이후에 별도의 모니터링 단계를 추가합니다. 이는 유저스페이스 API로 관리되는 커널 필터 테이블을 사용하여 원하는 패킷을 확인하고, 이후 주기적으로 모니터링 테이블을 업데이트하여 모니터링 응용 프로그램의 카운터를 업데이트합니다. 반면, 경로 외부 방식은 새롭게 추가된 링 버퍼 캐시를 사용합니다. 패킷이 수신되면 중복되어 링 버퍼로 전달되며, 이것은 모니터링 단계에서 사용됩니다. 나머지 복사본은 커널 플로우 캐시를 통해 전달되어 즉시 빠져나가며, 이는 모니터링과 전달 사이의 완전한 분리를 달성합니다. 제안된 eBPF 디자인은 사용자 스페이스의 모니터링 응용 프로그램과 상호 작용하기 위해 두 개의 eBPF 맵을 사용합니다. eBPF 프로그램은 트래픽 컨트롤(TC) 입구 및 출구에 부착되어 패킷이 수신되면 모니터링된 호스트(IP의 원하는 패킷)를 필터링하고 모니터링 응용 프로그램의 카운터를 업데이트하는 데 사용되는 두 번째 eBPF 맵을 업데이트합니다. 필터링 단계 이후에는 패킷이 OVS 데이터 경로를 진행합니다. eBPF의 주요 이점은 공유 eBPF 맵과 eBPF 프로그램의 저복잡성 때문에 모니터링된 호스트를 런타임 중에 업데이트할 수 있다는 것입니다. 결과는 eBPF 디자인이 구현 복잡성 측면에서 다른 디자인보다 우수하다는 것을 보여주었습니다. eBPF는 OVS 아키텍처와 완전히 독립적이며 런타임에서 로드되고 업데이트될 수 있으며, 다른 디자인은 업데이트시 재컴파일 및 재설치가 필요합니다. 지연시간과 메모리 소비 측면에서는, 경로 외부 디자인이 높은 메모리 소비 비용으로 이긴다는

것이지만, 링 버퍼가 없으면 eBPF가 우세합니다. 전반적으로, 각 디자인에는 장단점이 있으며 사용자가 어떤 것이 더 효과적으로 적합한지를 결정합니다.

8) WiFi 네트워크 모니터링

Sheth 등 [48]는 특히 무선 환경에서의 네트워크 작동에 대한 심층적인 통찰력 획득의 중요성을 지적했습니다. 저자들은 WiFi 네트워크를 모니터링하기 위해 eBPF를 활용하는 첫 번째 프레임워크인 FLIP을 제안했습니다. 이를 통해 두 가지 문제를 해결하였는데, 정확한 유선-무선 패킷 모니터링과 액세스 포인트 (AP) 스테이션의 에너지 소비입니다. eBPF의 타임 스탬핑 기능을 통해 패킷이 원하는 무선 액세스 채널에 연결되기 전에 여러 계층을 거치는 유선-무선 패킷 전환의 정확한 측정이 가능해졌습니다. 또한, APs 스테이션은 수면 모드로 전환하고 수면 모드에서 깨어나는 과정, 즉 깨어남 프로세스로 인해 많은 양의 에너지를 낭비합니다. 이러한 에너지 소비 문제를 해결하기 위해 저자들은 eBPF 기능을 사용하여 스테이션의 듀티 사이클 패턴을 프로파일링하는 것을 제안했습니다. FLIP은 그 동안 그 종류 중 첫 번째였지만, 결과는 표준 전력 모니터링 도구와 비교하여 6%의 에너지 소비 감소를 보여주었습니다. 그러나 이 프로토콜은 일반적인 전력 모니터링 도구와 같이 추가 하드웨어 설치가 필요하지 않습니다.

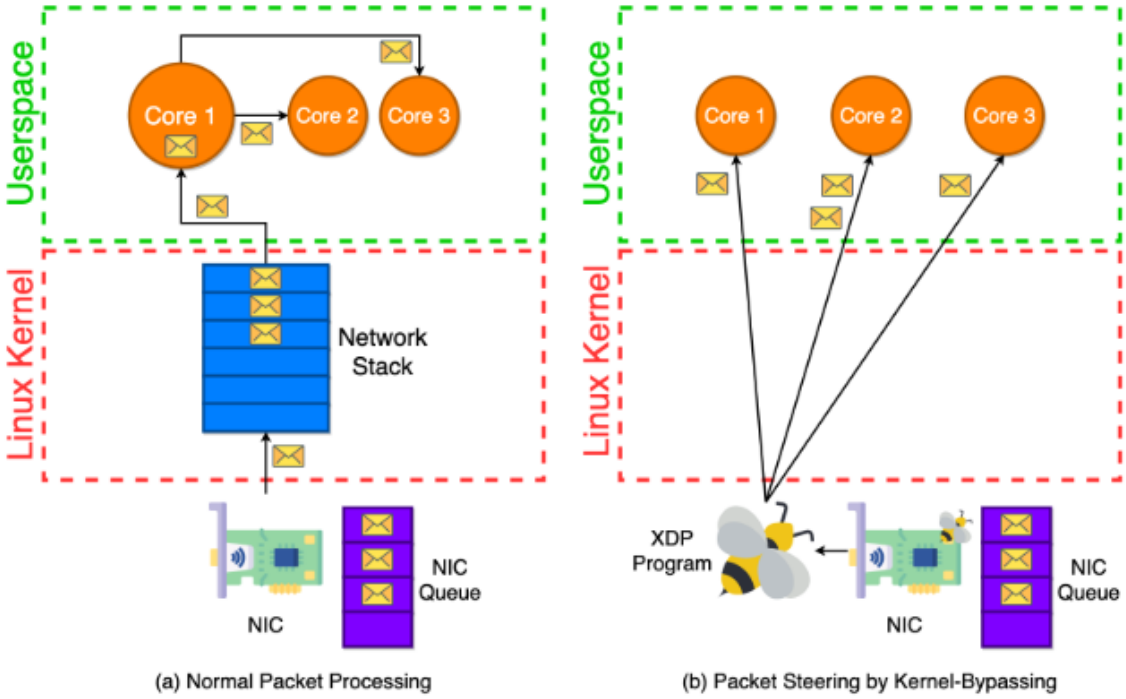


그림 10 (a) 도착한 패킷을 처리하기 위한 원래의 흐름, (b) eBPF XDP 프로그램을 사용하여 커널을 우회하여 패킷 스터링을 하는 새로운 방법입니다.

9) 다중 스트리밍 제어를 위한 실시간 엣지 컴퓨팅

데이터 다중 스트리밍은 센서, 사물인터넷(IoT) 장치, 엣지 서버 등에서 처리되어야 합니다. 가장 중요한 네트워크 관련 문제는 엣지 장치에서 계산을 위해 데이터를 복제하여 발생하는 혼잡으로 인한 용량 제한입니다. Baidya 등 [49]는 네트워크 엣지에서 처리되는 실시간 IoT 데이터 트래픽을 위해 eBPF를 사용한 계산 인식 통신 제어 프레임워크를 제안했습니다. 저자들은 네트워크 및 엣지 서버 자원 활용을 개선하고, 동적 네트워크 및 패킷 필터링 제어를 특징으로 하는 더 나은 시스템을 구현했습니다.

10) 패킷 스터링

네트워크 인터페이스 카드(NIC)의 진화로 연구자들은 응용 프로그램 수준 병렬성(ALP)을 개선하기 위한 새로운 접근 방법을 찾아야 했습니다. CPU와 NIC 작동 속도의 차이로 인해 시스템이 NIC 속도를 따라가지 못하여 지연되었습니다 [50]. CPU 속도가 NIC에서 도착하는 트래픽(예: 100 Gbps)을 처리할 수 있더라도 메모리 계층 구조와 캐시의 작은 용량이 전체 프로세스의 병목입니다. [51]에서는 응용 프로그램 수준 분할과 패킷 스터링을 결합하여 응용 프로그램 수준 병렬성을 개선하는 방안을 제안했습니다. Figure 10은 도착한 패킷을 처리하는 원래 접근 방식과 eBPF XDP 프로그램을 사용한 새로운 제안된 방식을 보여줍니다. Figure 10(a)에서는 패킷이 커널의 네트워크 스택에 저장되고 코어 1로 전달됩니다. 이 코어는 작업을 분산하기 위해 패킷을 다른 코어로 보낼 수 있는지를 결정합니다. 현재 병목은 코어 1의 속도인데, 이는 모든 패킷을 처리해야 합니다. Figure 10(b)에서 제안된 접근 방식은 커널을 우회하고 작업을 직접 사용 가능한 코어로 분산하기 위해 eBPF 기술을 사용했습니다. 이 접근 방식은 추가 평가가 필요하지만, 저자들은 이 구현이 응용 프로그램 수준 병렬성을 높이고 패킷 처리의 오버헤드를 줄이는 한 걸음으로 생각합니다.

*패킷 스터링: 네트워크에서 도착한 패킷을 적절한 처리 경로로 유도하거나 분배하는 기술을 가리킵니다. 이는 네트워크 성능을 최적화하고 네트워크 리소스를 효율적으로 활용하기 위해 중요한 역할을 합니다. 패킷 스터링은 다양한 기준에 따라 패킷을 분류하고 다른 시스템 또는 서비스로 전달하는 과정을 포함합니다. 이를 통해 네트워크 트래픽을 관리하고 최적화할 수 있습니다.

11) 5G MOBILE GATEWAY

5G 모바일 게이트웨이는 모바일 사용자 장치를 패킷 데이터 네트워크(예: 인터넷)에 연결하는 역할을 합니다. Parole 등은 모바일 패킷 코어(Mobile Packet Core, MPC)의 5G 모바일 게이트웨이에서 eBPF 통합에 대해 논의했습니다. 실험 결과는 eBPF가 엣지 컴퓨팅을 위한 작고 분산된 센터

에 대한 좋은 솔루션이 될 수 있다는 것을 보여주었습니다.

12) FAILURE/CONGESTION DETECTION AND PATH REROUTING

다수의 연구자들이 네트워크 경로의 실패와 혼잡을 감지하고, 심지어 발생하기 전에도 네트워크의 동작을 분석하고 가능한 다가오는 문제를 예측하는 영역에 대해 연구를 진행하고 있습니다. 이러한 설계들은 어떤 네트워크 매개변수를 캡처하고 사용할 것인지 선택하고, 여러 매개변수를 결합하고, 네트워크 동작을 관찰하는 것을 기반으로 합니다. 실행 단계에는 네트워크 관리자에게 경고를 보내거나, 흐름의 동적 경로 변경, 기계 학습 예측 모델을 작성하기 위한 데이터 수집 등이 포함될 수 있습니다. 실험 환경을 기준으로 사용 사례를 다중 프로토콜 레이블 스위칭(MPLS)과 소프트웨어 정의 광대역 네트워크(SD-WAN)로 나누었습니다.

a: IPv6를 사용한 세그먼트 라우팅(SRv6)

SRv6는 네트워크를 세그먼트로 분할하고 각 패킷에 완전한 라우팅 경로를 유지하는 대신 패킷 위에 라우팅 지시문(라벨)을 추가하여 기반으로 하는 새로운 라우팅 메커니즘입니다. 패킷은 방향(방문할 세그먼트)을 유지하고 네트워크를 통과합니다. 각 라우팅 지점에서 패킷은 저장된 세그먼트 방향을 사용하고 라우터는 패킷을 다음 세그먼트 라우팅 지점으로 안내합니다. 세그먼트 라우팅은 네트워크의 각 라우팅 노드가 연결된 노드에 관련된 정보만 유지함으로써 개별적으로 라우팅 노드의 부하를 줄입니다. Xhonneux 등은 SRv6에서 eBPF를 사용하여 빠른 경로 변경 서비스를 제공하기 위한 새로운 방법을 제안했습니다. 저자들은 빠른 경로 변경 서비스를 제공하기 위한 주요 세 가지 요소(장애 감지, 장애 감지 시 패킷 재경로 지정 및 백업 경로 찾기)를 논의했습니다. 그들의 설계에서, 저자들은 네트워크 노드를 마스터 노드와 슬레이브 노드 두 가지 유형으로 분류했습니다. Figure 11에서는 세그먼트 1의 모든 노드를 마스터 노드로 가정하고 나머지는 슬레이브입니다. 마스터 노드는 eBPF 맵에 상태 링크(ON/OFF)를 저장하고 슬레이브는 필요할 때마다 링크 상태를 계산합니다. eBPF를 활용한 Topology-Independent Loop-Free Alternate (TI-LFA) 메커니즘과 결합된 제안된 방법은 패킷의 라우팅 경로 설정을 복잡하게 만드는 대신 추가 응용 계층 프로토콜을 추가하는 고전적인 양방향 전달 감지 프로토콜 (BFD)과 비교하여 스루풋 향상률이 8%로 나타났습니다. (대칭 설정이 필요함)

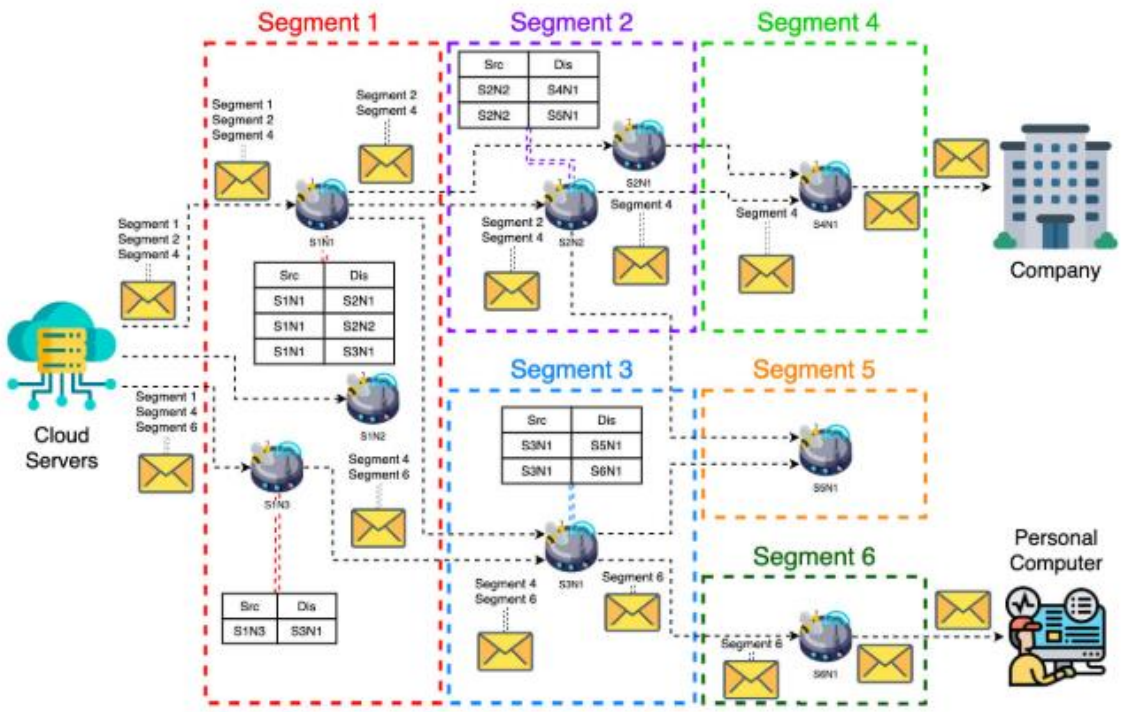


그림 11 eBPF가 포함된 세그먼트 라우팅. 세그먼트 1 라우터는 eBPF가 설치된 마스터 노드로 간주됩니다. 다른 세그먼트 라우터들은 eBPF 기반의 슬레이브 라우터입니다.

Jadin 등 [58]은 TCP 프로토콜을 기반으로 한 SRv6에서 장애를 감지하고 경로를 재지정하는 대안적인 방법을 제안했습니다. 이를 TCP 경로 변경기 (TPC)라고 합니다. 저자들은 작업을 두 가지로 나누었습니다. 첫째, 먼 링크 장애에서의 복구로, 이는 발신자와 수신자 양쪽에서의 경로 장애를 감지하고, 이를 사용하여 대체 경로를 선택합니다. 그리고 둘째, 경로의 성능 특성을 모니터링하여 최저 지연 경로를 동적으로 선택합니다. 경로 장애로부터 복구하기 위해서는 발신자와 수신자 양쪽에서 경로를 별도로 추적해야 합니다. 발신자는 카운터 기반 형식 또는 시간 기반 형식 중 하나에 따라 작동하도록 구성될 수 있습니다. 첫 번째 유형에서 (도 12(a) 참조), 발신자로부터 재전송 제한 시간 메시지(RTO)의 수를 유지하고, 네트워크 디자이너가 사전에 구성한 특정 수를 초과하면 다른 경로를 사용하여 재전송합니다. 시간 기반 형식에서 (도 12(b) 참조), 실제 수 대신 RTO 메시지의 총 시간을 유지합니다. 수신자 측에서는 양쪽 모두에서 동작하여(도 12(c) 참조), 이쪽(수신자)에서 발신자 쪽으로 보낸 중복된 확인 응답의 수를 계산하고 경로 변경의 트리거로 사용합니다. 패킷이 수신자에 도달하면 수신자는 발신자에 대해 ACK로 응답하여 다음 패킷으로 진행하지만, 동일한 패킷이 수신자에 도달하면 경로의 문제로 인해 ACK가 발신자 끝점에 도달하지 않은 것입니다. 경로 선택은 다중 암 기계 (MAB) 접근 방식을 기반으로 하며, 이 방식은 새로운 경로의 탐색과 현재 경로의 이용을 균형 있게 조정합니다. 이는 선택된 경로의 영향을 잡아내는 보상 함수에 따라 이루어집니다(수집된 성능 통계에 따라). 사용 가능한 경로는 누적된 보상 값을 기준으로 내림차순으로 저장되어 가장 낮은 지연 경로를 반영합니다. 전반적으로, 제안된 방법은 경로 장애의 감지를 처리하고, 가장 낮은 지연 경로 목록을 기반으로 다른 경로로 재지정합니다.

TPC는 커널 공간의 격리된 eBPF VM에서 실행되며, 경로 헤더와 연결을 두 개의 다른 eBPF 맵에 저장합니다. 설정 구성(경로 및 요구 사항)은 TPCDaemon에 의해 수행됩니다. 이는 주요 코드를 호스팅하는 eBPF VM과 eBPF 맵에 대한 인터페이스로 작동하는 사용자 공간 프로그램입니다.

*TPCDaemon은 TPC의 설정을 관리하고 제어하는 역할을 담당하는 프로그램입니다. 이는 사용자 공간에서 실행되며, eBPF 맵 및 eBPF VM과 상호 작용하여 TPC의 동작을 설정하고 모니터링합니다. TPCDaemon은 주로 TPC의 설정 구성, 경로 및 요구 사항 설정, 그리고 eBPF 맵 및 VM과의 인터페이스 역할을 수행하여 TPC 시스템의 원활한 운영을 지원합니다.

*TCP 경로 변경기(TCP Path Changer)의 약자입니다. 이는 SRv6에서 경로의 장애를 탐지하고 다른 경로로 다시 라우팅하는 기능을 제공하는 대체 접근 방식으로 제안되었습니다. TPC는 TCP 프로토콜을 기반으로 작동하며, 네트워크에서 발생하는 경로 문제를 감지하고 다른 경로로의 전환을 처리합니다.

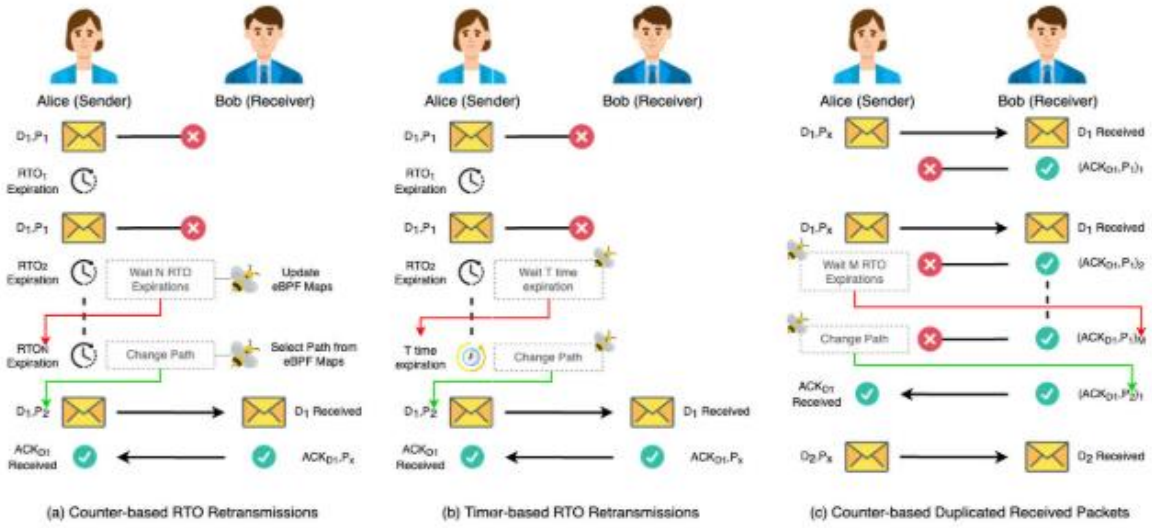


그림 12 (a) 카운터 기반 송신자, (b) 시간 기반 송신자, (c) 카운터 기반 수신자.

b: 소프트웨어 정의 광역 네트워킹 (SD-WAN)

소프트웨어 정의 광대역 네트워킹(SD-WAN)은 지리적으로 떨어진 넓은 영역에서의 소프트웨어 정의 네트워킹(SDN)을 가리킵니다. Troia et al. [60]은 ONOS(Open Network Operating System) [61] 기반의 WAN 컨트롤러에서 TCP 재전송 횟수를 캡처하고 기록하기 위한 새로운 방식을 개발했습니다. 이 방식은 eBPF를 기반으로 하며 커널 에이전트(Kernel Agent)와 캡처된 흐름 정보를 저장하는 버퍼, 그리고 각 서비스 흐름별로 재전송 횟수에 대한 사전 정의된 임계값 테이블을 가지는 모니터링 응용 프로그램으로 구성됩니다. 이 설계의 주요 목표는 네트워크 장애(즉, 혼잡, 링크 중단)로 인한 다운타임을 최적화하고 최소화하여 서비스의 탄력성을 향상시키는 것입니다. 저자들은 해당 시스템이 다른 시스템에 비해 CPU 사용량이 낮다고 주장하지만 실제 비교 자료는 제시되지 않았습니다. 그러나 이 방식은 실시간 세그먼트 손실을 감지할 수 있으며 Linux 운영 체제의

eBPF에 의존하므로 사전 설치된 소프트웨어가 필요하지 않습니다.

13) 엣지 클라우드에서 실행 특성 측정

Gowtham 등 [62]은 MessTool 프레임워크를 설계하여 클라우드 개발에서 분산 애플리케이션의 종단 간 동작을 프로파일링합니다. 이 프레임워크의 목적은 응용 프로그램 공급 업체 요구 사항을 고려하여 클라우드 인프라의 성능을 측정하여 지정된 목적을 위한 특수화된 클라우드 환경을 설계하는 데 도움을 주는 것입니다. MessTool은 커널의 이벤트 추적 및 타임 스탬핑 기능을 활용하여 실행 경로상의 이벤트의 실행 특성을 측정합니다.

14) 빠르고 확장 가능한 발행/구독 시스템

게시/구독 아키텍처 [63]는 사용자 관심을 기반으로 알림 서비스를 제공하는 데 사용될 수 있습니다. 발행자는 알림을 생성하고, 구독자는 관심 있는 내용을 수집합니다. 이 아키텍처는 브로커라고 하는 중간 장치에 기초하며, 알림과 구독자를 매칭하는 전달식 테이블을 구현합니다. 이 아이디어의 응용 중 하나는 IoT에서 찾을 수 있습니다. IoT에서 센서는 데이터를 수집하기 위해 사전 프로그래밍되며, 이러한 데이터는 브로커에게 전송됩니다. 브로커는 구독자의 관심에 따라 데이터를 전달합니다. Tatarski 등 [64]은 이러한 매칭 및 전달 프로세스의 오버헤드를 줄이기 위해 커널 수준에서 실행되는 공간에 맞추기 위한 발행/구독 방식의 혁신적인 아키텍처를 제안했습니다. 저자들은 브로커를 기존의 엣지 브로커 (CEB)와 eBPF 기반 클라우드 브로커 (eBPF-CB)로 분할하는 방법을 제안했습니다. 이로 인해 데이터가 이미 사전 처리되었으므로 클라우드 브로커의 목표는 패킷 전달만을 수행합니다. eBPF의 역할은 패킷을 수신하자마자 이를 가로채고 목적지 IP를 복제하고 교체하여 해당 구독자에게 전달하는 것입니다. 저자들은 사용자 공간 브로커와 eBPF-CB에 대해 초당 전달된 알림을 평가했습니다. 결과적으로 후자가 전자보다 20~25배 더 나은 성능을 보였습니다. 이 아이디어의 전체적인 목적은 원래의 브로커 작업을 처리와 전달로 분할하는 것이었습니다. 저자들은 전달 작업만을 다루었으며, 이 접근 방식은 전달 속도와 설계의 확장성을 크게 향상시켰지만, 새로운 처리 단계를 도입했기 때문에 전체 경로 지연시간이나 전력 소비와 엣지 브로커 및 클라우드 브로커의 수명에 미치는 영향은 평가하지 않았습니다.

*엣지 클라우드(Edge Cloud)는 클라우드 컴퓨팅의 확장으로, 데이터 및 서비스를 사용자와 더 가까운 네트워크 엣지에 위치한 데이터 센터에서 제공하는 것을 의미합니다. 이는 사용자 디바이스와의 통신 레이턴시를 줄이고 데이터 처리를 가속화하여 실시간 및 지연 시간에 민감한 응용 프로그램에 대한 서비스 품질(QoS)을 향상시키는 데 기여합니다. 엣지 클라우드는 주로 스마트 시티, 스마트 팩토리, 자율 주행 자동차 및 IoT(IoT) 디바이스와 같은 분야에서 사용됩니다.

*엣지 브로커는 엣지 컴퓨팅 환경에서 중간 장치로서 동작하는 소프트웨어 컴포넌트입니다. 엣지 브로커는 주변 장치에서 생성된 데이터를 수집하고 처리하여 클라우드로 전송하거나, 클라우드에서 주변 장치로 데이터를 전송하는 역할을 합니다. 이러한 역할은 주로 엣지 디바이스와 클

라우드 간의 데이터 흐름을 조정하고 최적화하는 데 사용됩니다. 엣지 브로커는 네트워크 트래픽을 최적화하고 데이터 처리 성능을 향상시키는 데 도움이 됩니다.

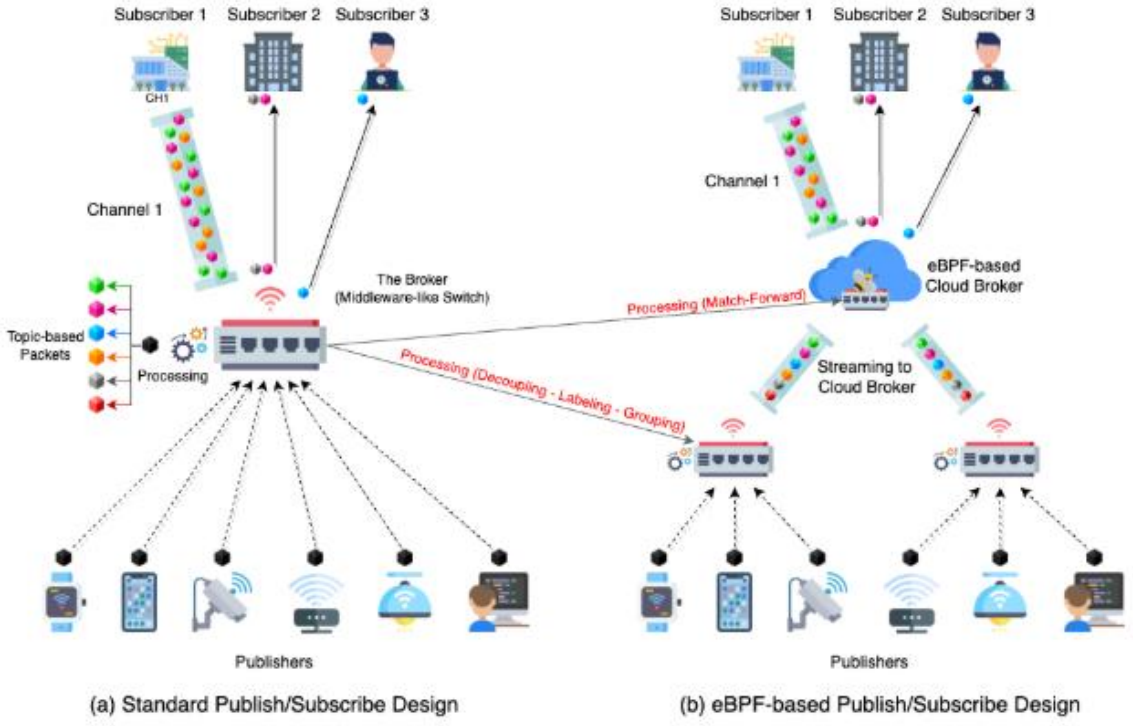


그림 13 (a) 표준 게시/구독 디자인, (b) 제안된 eBPF 기반 게시/구독 디자인입니다.

15) IoT에서의 서버리스 엣지 클라우드 컴퓨팅

서버리스 엣지 클라우드 (SEC)는 클라우드 아키텍처의 계산을 엣지에 더 가까운 곳으로 이동시키는 새로운 클라우드 패러다임입니다. 클라우드 아키텍처의 주요 장점은 낮은 지연 시간과 중앙 집중식 계산이며, 이는 사용 가능한 자원으로 쉽게 관리되고 확장될 수 있습니다. 그러나 클라우드 계산은 제공된 서비스의 품질을 유지하는 데 드는 높은 비용으로 고생합니다. SEC의 주요 목표는 동일한 서비스 품질을 유지하면서 비용을 줄이는 것입니다. 왕 등 [65]은 IoT 트래픽을 위한 새로운 SEC 기반 설계를 제안했습니다. 이 설계는 두 가지 주요 문제인 i) SEC 설계의 cold-start 문제와 ii) Knative 큐 프록시의 오버헤드를 해결하기 위해 Knative 프레임워크를 사용했습니다. IoT 통신에 사용되는 표준 프로토콜은 Message Queueing Telemetry Transport (MQTT)이며, 이는 그림 13(a)의 게시/구독 구조를 기반으로 합니다. 반면 Knative는 통신에 HTTP를 사용합니다. 따라서 저자들이 처음으로 소개한 것은 표준 MQTT 프로토콜을 HTTP 프로토콜로 변환하여 Knative가 처리할 수 있도록하는 MQTT-to-HTTP 어댑터입니다. cold-start 문제는 서비스가 활동이 없는 긴 기간 동안 슬립 모드로 들어가면, 요청이 도착하고 서비스가 다시 기능을 활성화 할 때 높은 응답 대

기 시간이 발생하는 문제입니다. 저자들은 XGBoost를 기반으로 한 요청 도착 예측 구성 요소를 설계했습니다. 저자들은 이벤트 기반 프록시 (eProxy)를 설계하기 위해 eBPF를 활용했습니다. eProxy는 CPU가 아무런 사이클도 낭비하지 않고 유휴 상태에 머무를 수 있도록 합니다. 이것은 이더넷 인터페이스 위에 배치되며, 따라서 사전 정의된 이벤트 (예 : 패킷 도착, 패킷 체인)에서 eProxy가 트리거되고 CPU가 처리를 시작합니다. 결과는 큐 프록시를 eProxy로 교체함으로써 CPU 오버헤드를 최대 37% 절약할 수 있다고 보여줍니다. 그러나 새로운 중간 구성 요소 (MQTT-to-HTTP 어댑터 및 MQTT 브로커)로 인해 도입되는 지연 시간을 연구하기 위해 더 많은 실험이 필요합니다. 이 부분은 향후 작업으로 남겨졌습니다.

a: INSIGHTS • 네트워킹 작업의 주요 초점은 성능입니다. 시스템 이용률 (CPU, 메모리, 레지스터, 네트워크 스택 등), 처리량, 처리 능력, 에너지 소비, 사용 가능한 자원의 확장성 및 확장 가능성의 형태로 나타날 수 있습니다. • 핵심은 eBPF의 구성 요소가 어떤 용도로 사용되는지입니다. eBPF 프로브에서의 피드백은 정보 수집 (자원 사용, 패킷 정보)에 사용될 수 있습니다. eBPF 맵은 사용자 공간 또는 커널 공간에서 효율적으로 액세스할 데이터를 저장하거나 그들 사이에서 데이터를 교환할 수 있는 지점으로 사용될 수 있습니다. • 수집된 데이터는 프로파일링 (응용 프로그램, 사용자, 장치), 패킷 헤더 수정 (흐름 방향 변경), 패킷 복사 (분석 및 중복을 위해 복제) 및 수집된 정보를 기반으로 연결된 장치의 동작 변경 (유휴 장치의 전원 차단, 네트워크 프로토콜 제어)에 사용됩니다. 이러한 기능은 더 나은 부하 분산, 혼잡 제어, 과부하 장치의 자원 사용량 감소, 유휴 장치의 에너지 소비 감소에 사용될 수 있습니다. 표 1은 네트워킹 도메인과 관련된 논의된 연구들의 요약과 그들의 주요 측면을 지적하며, 제안된 설계의 한계와 미래 탐구를 위해 남은 영역을 보여줍니다. 첫 번째 열은 eBPF를 활용한 연구를 가리키며, 두 번째 열은 eBPF 프로그램이 저장되고 실행되는 위치를 정의합니다 (커널 공간 (KS), 사용자 공간 (US), 네트워크 인터페이스 카드 (NIC), 저장 서비스 (SS)). 세 번째 열은 eBPF 프로그램의 주요 역할을 정의하고, 네 번째 열은 연구의 주요 측면을 요약하며, 다섯 번째 열은 저자가 언급한 가능한 미래 방향에 대한 도전 과제를 보고합니다.

TABLE 1. Summary of eBPF use-cases in the networking domain.

Ref.	Place	Role	Key Aspects	Open Challenges and Future Work
[29]	KS	Storing the value-to-action records	Iptables rules translator to eBPF iptables version	N/A
[30]	KS	Duplicating and redirecting packets to the monitoring VM	Better CPU utilization and higher throughput Extendibility and scalability	XDP integration More testing scenarios
[32]	KS	Data collection of runtime kernel statistics and decisions	Zero impact on system performance High accuracy	Applying deep reinforcement learning techniques for training Hardware resources usage measurement
[35]	KS	Building the data plane for NFs	Higher throughput Better CPU utilization	Analysis of chain NFs and the removal of redundant features Polycube's scalability
[40]	KS	Encoding NFs as eBPF code	Generic version	Overhead and throughput issues
[42], [43]	KS	Collecting real-time data for TCP behavior and network parameters	Higher throughput and consistent data transfer rates with lower packet retransmission rates Real time modification to TCP window size	Investigating more detection perspectives
[49]	KS	Real-time decisions of packet forwarding and cloning policy	Better network and system resource utilization Dynamic network and packet filtering control	N/A
[51]	KS / NIC	Packet queueing and steering	Transparent NIC-CPU scheme to improve request-level parallelism	Performance evaluation
[52], [53]	KS	Network functions creation	Higher packet processing rate Scalability with number of cores No rigid machine resource partitioning	Deep packet inspection and charging Exploiting run-time injection capabilities Studying cross-modules optimization mechanism
[55]	KS	Storing and detecting a link status changes (UP/DOWN) for rerouting	No additional application layer Performance enhancements and robustness of the slave	Overhead reduction Improving performance of the master peer
[58]	KS	Detecting path failure and rerouting Ordering paths and selecting lowest-delay paths	Sender-to-Receiver path independent of Receiver-to-Sender Path ordering/selection based on MAB	Rerouting for better bandwidth utilization Collecting measurement from production traffic
[62]	KS	Event tracing and timing performance	Accurate profiling of execution characteristics of distributed edge applications Targeting user defined events	Profiling real-time applications dynamic configurations of TSN networks Optimizing complex wireless communications with software-based real-time requirements
[44]	KS, NIC	Offloading statistics packets counters and packet-to-application routing tables to the SmartNIC	Traffic accounting, detection of Half-Open TCP connections and DNS flow analysis Offloading IP-to-Application routing tables to NIC Scalability with the amount of resources available	Limitations of SmartNIC XDP offloads Stateful operations offloading to NIC
[45]	KS	Active and dynamic application flow tracing	Dynamic application flow alteration detection and analysis Dropping probe responses within XDP	Scalability with number of destinations and programmability and user control of ELF Noise in the latency measurements
[48]	KS	Monitoring wired-to-wireless packets switching, and tracking energy consumption of APs stations duty-cycle	Reducing energy consumption and number of awakening times No extra hardware equipment needed in the AP stations	Reacting to WiFi network dynamics changes
[64]	KS	Packets IP alteration and packets replication and forwarding	Higher forwarding speed Less overhead	Delivery guarantees (Reliability) Scalability of the eBPF-CB
[65]	KS	Event-based queuing on top of the ethernet interface	Integrating Knative with MQTT XGBoost-based traffic prediction Designing event-based queue Low CPU overhead	Additional Latency of new components Evaluating XGBoost-based prediction component Trying different prediction schemes
[46]	KS	Managing and storing monitored hosts IPs and collect their flow stats counters	Low implantation complexity and CPU overhead Independent of the hosting system architecture	N/A
[60]	KS	Counting number of TCP retransmissions	Low CPU usage Real-time detection of segment losses Not limited to SD-WAN traffic	Limited to TCP traffic Studying other link down causes Missing base reference comparison

B. 보안

eBPF의 능력을 활용할 수 있는 중요한 영역은 데이터 및 네트워크 보안입니다. 공격자가 서버에 있는 자원과 데이터에 비특권적 액세스를 얻을 수 있기 때문에 보안 침해로 인해 데이터 수정 또는 파괴, 기밀 사용자 또는 회사 데이터 유출, 마침내는 부정 광고 및 사용자들이 제공되는 서비스에 대한 신뢰 손실이 발생할 수 있습니다.

eBPF의 보안에서의 중요성은 패킷이 시스템에 들어가기 전이나 커널의 네트워크 스택에 도달하기 전에 패킷 분석을 가능케 하는 고속 처리입니다. 더구나 eBPF는 머신 러닝 기반 보안 응용

프로그램의 훈련 데이터 세트를 구축할 때 효율적인 데이터 수집에 사용되는 견고한 도구입니다. 또한 eBPF는 내장 검증기를 갖고 있어 이미 신뢰할 수 없는 오픈 소스 코드에 신뢰성을 제공할 수 있습니다. 다음은 이 연구에서 검토한 보안 관련 연구입니다.

1) DNS 개인 정보 보호

도메인 이름 시스템(DNS)은 전 세계 인터넷의 뿌리로, 인터넷 흐름, 노드의 연결 및 사용자와의 인터넷 상호 작용을 제어하는 기초입니다. DNS 레이어에 액세스하는 것은 사용자 행동에 대한 깊은 통찰력을 제공합니다. 단일 사용자 트래픽을 다른 DNS 제공자로 분산시키면 사용자의 개인 정보를 유지할 수 있습니다. [69]에서는 시스템에서 실행되는 각 응용 프로그램의 DNS 흐름을 변경하여 사용자 개인 정보 보호를 강화하기 위한 새로운 체계가 제안되었습니다. eBPF는 사용자에게 자신의 DNS 네트워크 트래픽에 대한 통찰력을 제공하여 사용할 DNS 서버를 선택하고 어떤 경로를 따를지 결정하는 데 도움을 줍니다. 제안된 솔루션은 표준 방법과 비교하여 사용자를 데이터 유출과 개인 정보 공격으로부터 보호합니다.

2) 포렌식 분석에서의 감사 로깅

포렌식은 공격자의 지문을 발견하는 것으로, 연기된 조사 활동입니다. 포렌식 분석의 두 가지 중요한 요구 사항은 시스템 활동 기록과 데이터를 간결하게 제공하는 것입니다. Rohit [70]은 런타임 오버헤드를 단 1%로 유지하는 효율적인 감사 로깅을 위한 eBPF 기반 프레임워크를 설계했습니다. 감사 시스템의 강점은 eBPF의 기본 구성에서 추적된 커널 활동과 사용자의 런타임 정의된 계측 코드의 두 가지 주요 소스로부터 수집된 데이터를 제공하는 것입니다. 제시된 감사 시스템은 생성된 감사 로그 크기를 줄이고, 실행 중 오버헤드를 낮추며, 커널에 추가 모듈을 설치할 필요가 없으며, 다중 처리 환경에서 사용할 수 있도록 만들어져 기존 감사 도구의 한계를 극복하는 데 도움이 되었습니다.

3) 컨테이너 보안을 위한 안전한 감사

[71]의 저자들은 컨테이너 보안을 위한 배치 가능한 안전한 감사 도구인 saBPF를 제안했습니다. 그들의 디자인은 saBPF가 전통적인 감사 도구와 동일한 성능 수준을 유지하도록 리눅스 보안 모듈 (LSM) [72]과 비교 평가되었습니다. 저자들은 eBPF 프레임워크를 확장하여 컴파일 시간에 감사 규칙 구성을 사용자 정의할 수 있도록하여 각 컨테이너가 자체 구성된 규칙 및 감사 정책을 갖도록 했습니다. 각 개별 컨테이너는 호스트 시스템 성능에 영향을 주지 않고 독립적으로 안전한 감사 도구를 배포할 수 있습니다. 저자들은 실행 시간이 아닌 컴파일 시간에 구성을 수행하는 효과를 강조했으며, 이는 실행 시간 감사 복잡성을 최소화하고 전반적인 성능을 향상시켰습니다.

4) 머신 러닝 기반 침입 탐지 시스템

Bachl 등 [22]은 eBPF를 활용한 커널에서 실행되는 기계 학습 기반 침입 탐지 시스템을 개발했습니다. 그들의 솔루션은 이전 패킷의 컨텍스트를 고려하여 패킷이 악성인지 여부를 결정하기 위해 의사 결정 트리를 사용합니다. 정확한 결과를 얻기 위해, 그들은 자신들의 IDS를 사용자 공간 응용 프로그램 및 eBPF 프로그램으로 구현했습니다. 그들의 결과는 커널 IDS가 사용자 공간 버전보다 처리된 패킷 수가 20% 증가했다는 것을 보여주었습니다.

5) 컨테이너 IDS

[73]에서 제안된 좋은 사용 사례 중 하나는 컨테이너 보안에 특화된 침입 탐지 시스템을 개발하기 위해 eBPF의 모니터링 능력을 활용하는 것입니다. 이 논문은 기술이 복잡한 컨테이너 및 응용 프로그램 수준 컨텍스트를 검색하고 기존 런타임 보안 도구의 성능 영향을 줄이는 방법을 보여줍니다.

6) 호스트 기반 이상 징후 탐지 시스템

ebpH [16]은 Somayaji의 pH 디자인 [74]의 eBPF 기반 버전입니다. 두 알고리즘의 아이디어는 시스템의 각 실행 파일에 대한 프로필을 작성하는 것입니다. 이 프로필은 시스템 프로세스의 정상 동작 기록을 설정합니다. 시스템 호출이 설정된 프로필을 위반할 때마다 ebpH는 사용자에게 경고를 표시하고 사용자가 적절한 조치를 취할 수 있도록 해당 시스템 호출을 지연시킵니다. 성능 측면에서, eBPF 덕분에 ebpH 감지는 시스템에 무시할 정도의 오버헤드를 초래하지만 안전성 보장을 유지했습니다. ebpH의 코드는 오픈 소스로 제공되어 개인이 자신의 장치에 호스트 기반 이상 징후 탐지 시스템을 구현하는 데 적합한 프로젝트입니다.

7) 빠른 패턴 매처 기반 IDS

Wang과 Chang [24]은 eBPF를 기반으로 한 Snort [75]와 유사한 IDS를 설계했습니다. Snort는 사용자 공간에서 실행되어 침입 탐지 및 방지 도구로 설계되었지만, Wang과 Chang의 구현은 커

널에서 실행됩니다. 그들의 접근 방식은 사용자 공간의 제어 프로그램을 추가하여 eBPF 프로그램의 실행을 구성하고, 커널 레벨에서 실행되는 eBPF 기반 빠른 패턴 매처 (FPM)를 추가하는 것입니다. FPM 엔진은 Aho-Corasick (AC) 알고리즘 [76]을 기반으로 하며, 키워드에서 상태 머신을 구성하고, 그런 다음 가장 긴 일치 패턴을 한 번에 찾아냅니다. eBPF의 제한 조건에 따르면, 실험 결과는 동일한 테스트 조건에서 Snort에 대한 그들의 디자인의 효율성을 증명했습니다.

8) 마이크로서비스에서의 서비스 메시 관찰 가능성

컨테이너에서 마이크로서비스의 발전으로 인해 동일한 서비스로 이동하는 트래픽을 조정하는 문제가 발생했습니다. 표준 경우에는 호스트 장치가 조정자가 되어 관리 응용 프로그램이 개발되는 문제가 발생했습니다. 서비스 메시라는 디자인 패턴 중 하나는 서비스 코드 수정 없이 트래픽 관리, 관찰 가능성 및 보안을 허용하기 위해 마이크로서비스 상단에 투명하게 배치되는 레이어(제어 플레인)를 갖도록 하는 것입니다. 현대 도구의 문제는 이벤트 및 로그를 수집하기 위해 정적 및 유연하지 않은 메트릭을 사용하는 것입니다. Levin은 ViberProbe [78]를 (작성 시점 기준으로) 첫 번째로 확장 가능한 eBPF 기반 동적 및 적응형 마이크로서비스 메트릭 수집 프레임워크로 제안했습니다. eBPF는 애플리케이션별 구성을 할당하고 그들의 메트릭을 수집합니다. 수집된 데이터는 서비스 패턴 분석을 기반으로 새로운 구성을 생성한 다음 그림 14(b)에 표시된 것처럼 배포되고 분배될 수 있습니다.

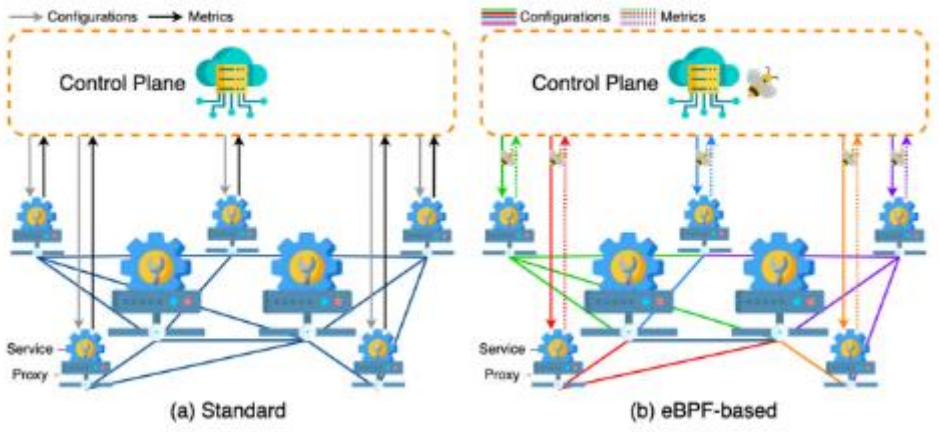


그림 14 서비스 메쉬 개요.

*메트릭: 측정 항목이나 평가 지표를 의미

9) VM 및 컨테이너 패킷 가시성

Deri 등 [79]는 VM 및 컨테이너로 들어오거나 나가는 데이터 암호화로 인한 패킷 가시성 문제를 고려했습니다. 이 문제를 해결하는 표준 방법은 VM 또는 컨테이너 당 패킷 분석기를 배치하는 것인데, 이는 실제 시나리오에서 실행할 수 없습니다. 저자들은 네트워크 활동을 모니터링함으로써 운영 체제 수준에서 통신 가시성을 활성화하는 새로운 도구를 설계했습니다. 이는 패킷이 시스템 경계를 넘어가는 것을 모니터링하는 일반 IDS/IPS와는 달리 커널 프로브와 추적점을 사용합니다.

10) 컨테이너 정책 강제 실행

컨테이너는 호스트 자원을 공유하며, 컨테이너가 호스트와 상호 작용하는 방법은 시스템 호출(syscalls)을 보내는 것입니다. 응용 프로그램을 컨테이너화하면 호스트 자원의 재사용이 가능해지며, 이러한 방식으로 컨테이너 당 정책이 필요하여 호스트 시스템에 배포된 응용 프로그램에서 다양한 보안 프로토콜을 지원할 수 있습니다. Bélair 등 [80]은 SNAPPY라는 Safe Namespaceable And Programmable PolicY를 제안했습니다. SNAPPY 프레임워크는 Linux Security Model (LSM) 훅을 위한 네임스페이스별 정책의 정의를 허용했습니다. 논문의 주요 기여는 설계된 네임스페이스 정책_NS를 사용하여 신뢰할 수 없는 프로세스가 커널로 eBPF 기반 정책을 강제하는 것을 허용하고, 런타임에 로드할 수 있는 동적 eBPF 도우미를 구현한 것입니다. 저자들은 또한 SNAPPY가 관련 컨테이너 엔진 (예: Docker)과 통합될 수 있다는 것을 시연했습니다. Findlay 등 [81]도 비슷한 작업을 수행하여 BPFcontain을 제안했는데, 이는 컨테이너를 제한하고 최소 권한 액세스를 보장하기 위한 유연한 정책 언어입니다.

11) 프로세스 제약

프로세스 제약은 오픈 소스 코드와 신뢰할 수 없는 코드가 실행되고 무단으로 시스템 리소스 및 다른 응용 프로그램에 액세스하는 것을 제한하는 것으로 정의됩니다. 제약 프로세스는 특히 클라우드에서 시스템에 더 많은 오버헤드를 추가합니다. 목표는 오버헤드를 최소화하는 것입니다. Findlay 등 [82]는 eBPF를 기반으로 한 bpfbox라는 프로토타입 제약 애플리케이션을 제시했습니다. 그들의 프로토타입은 AppArmor [83]와 비교하여 적은 오버헤드를 보였습니다. bpfbox의 강력함을 증명한 결과는 시스템 보안과 관련된 미래 응용 프로그램에 대한 여러 기회를 열었습니다.

12) 다중 수준의 제어 흐름 인증

시스템 인증은 시스템 구성의 무결성과 시스템 실행 동작의 정확성을 보장하는 과정입니다. 원격 인증에서는 서드파티가 장치가 사전 정의된 구성에 따라 작동하고 있는지를 검증해야 합니다. 인증서는 시스템의 신뢰성의 증명으로 사용됩니다. Papamartzivanos 등 [84]는 표준 원격 인증 솔루션의 한계를 극복하기 위해 하이브리드 제어 흐름 인증 (CFA) 프레임워크를 제안했습니다. 이는 eBPF 및 Intel Processor Trace 기술 (Intel PT) [85]을 사용한 다중 수준 모니터링 및 보고를 기반으로 합니다. 다중 수준 설계의 주요 아이디어는 먼저 시스템을 eBPF를 사용하여 모니터링하는 것입니다 (저자들은 이를 고수준 모니터링이라고 부릅니다). 그리고 의심스러운 활동이 감지되고 경보가 발생하면 Intel PT를 사용하여 깊이 있는 조사를 진행합니다. 다시 말해, eBPF는 소프트웨어 명령과 실행 흐름을 추적하는 데 첫 번째 수준의 인증으로 사용됩니다. 실패할 경우 두 번째 수준의 포렌식 분석이 시작됩니다. 여기서는 Intel PT를 사용하여 프로그램 및 데이터 메모리에 액세스하여 공격을 감지합니다. 결과는 eBPF 추적이 잘 맞고 전반적인 오버헤드를 추가하지 않는다는 것을 보여주었습니다. 반면, 특정 문제를 대상으로 작은 규모에서도 Intel PT는 추가적인 오버헤드를 추가할 것이며, 따라서 CFA의 가장 효율적인 확장 가능한 솔루션은 다중 수준 추적 및 보고를

사용하는 것입니다. 모니터링 및 조사 깊이는 수준에 관련이 있습니다. 두 수준 시스템의 경우, 낮은 수준 (즉, 첫 번째 수준)은 적은 오버헤드를 가지지만 조사 깊이가 적습니다. 반면, 높은 수준 (즉, 두 번째 수준)은 더 많은 오버헤드를 생성하지만 정확하고 정밀한 조사 이미지를 제공합니다.

13) 비대칭 DoS 탐지

서비스 거부 (DoS) 공격은 시스템을 중단시키고 무용한 트래픽으로 채우거나 사용 가능한 리소스를 점유하는 집중적인 계산을 요청하여 일반 사용자가 시스템 서비스에 액세스할 수 없도록하는 것을 목표로합니다. 공격 방지는 공격의 효과를 줄이거나 공격을 미리 방지하는 것입니다. DoS 공격을 분류하는 한 가지 방법은 공격 대상 리소스에 사용된 리소스의 수와 유형을 관찰하는 것입니다. 그들이 동일한 양과 유형일 때, 그것은 대칭 DoS로, 사용된 리소스가 대상 리소스보다 적거나 다른 유형이면 비대칭 DoS (ADoS)라고합니다. Findlay 등 [87]은 고유 한 특성으로 인해 ADoS의 경우 클래식 방법으로 DoS 공격을 식별하는 것이 더 이상 적용되지 않습니다.

FINELAME [88]은 감지를 위한 주요 관심사로 ADoS 공격을 고려하는 모니터링 도구입니다. eBPF를 사용하여 세 가지 구성 요소가 소개되었습니다. 즉, i) 수신 요청 처리를 담당하는 기능에 응용 프로그램 수준 프로브를 부착, ii) 커널/사용자 공간의 데이터 리소스를 모니터링하고, iii) 수집된 데이터를 사용하여 이상한 요청 패턴을 감지하는 반지도 학습 모델을 구축합니다. FINELAME은 ReDoS, Billion Laughs 및 SlowLoris 공격에 노출되어도 거의 100%의 정확도로 공격과 합법적인 요청을 구별했습니다.

14) DDoS 완화를 위한 자동 규칙 생성

분산 서비스 거부 (DDoS)는 감염된 컴퓨터 (좀비)를 사용하여 공격자 컴퓨터 (마스터)에 의해 제어되는 컴퓨터를 사용하여 희생 시스템으로 무용한 트래픽을 보내어 리소스를 넘치게합니다. 이는 추적할 수 없는 분산 기계가 공격 시작자의 원천을 조사하기 위한 추적점을 사용하지 않습니다. 공격자는 제어된 서버 (CS)로 명령을 보내서 감염된 컴퓨터 (좀비라고 함)에 이 명령을 전달합니다. 감염된 컴퓨터는 대량의 무용한 트래픽을 희생자에게 보내어 사용 가능한 리소스를 모두 소비하게 만듭니다. 평균 사용자가 영향을받은 기계로 안전한 패킷을 보낼 때, 리소스가 부족하여 패킷이 삭제됩니다.

Wieren [89]는 DDoS 완화 시스템을 구축하기 위해 eBPF를 사용한 자동 규칙 생성에 대해 논의했습니다. 그림 15에서와 같이 유효한 및 유효하지 않은 트래픽이 XDP 필터로 전달될 때, 수용된 트래픽은 이상 탐지 시스템 (ADS)에 의해 조사됩니다. ADS는 합법적인 트래픽을 통과시키고 이상이 감지되면 패킷의 사본이 규칙 생성 시스템에 전송됩니다. 새로운 필터 세트가 생성되어 XDP 필터에 추가되고 이러한 프로세스가 반복됩니다. 이 연구는 100%의 필터링 정확도를 얻는 잠재

력을 보여주었습니다.

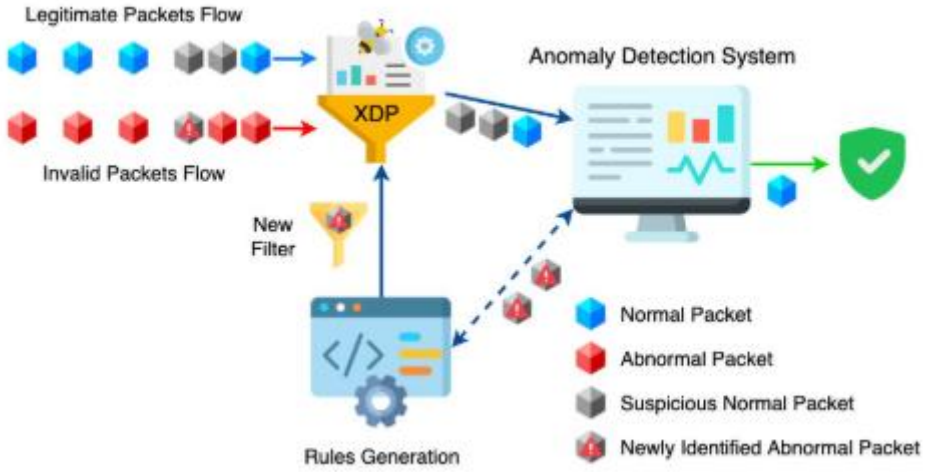


그림 15 DDos 공격에 대한 자동 규칙 생성.

15) 스테가노그래피적 악성코드 탐지

스테가노그래피는 비밀로 생각되는 이미지, 파일 또는 코드 내부에 메시지를 숨기는 과정입니다. 스테가노그래피적 악성코드는 임베드된 정보가 악성코드인 스테가노그래피적 방식입니다. Caviglione 등 [23]은 최소한의 오버헤드로 스테가노그래피적 악성코드(stegomalware)를 감지하는 가능성을 보여주었습니다. 그들의 결과는 가장 간단한 eBPF 코드가 머신 러닝 기반 도구와 같은 고급 기술로 전송되어 스테고말웨어를 감지하기 위해 분석될 수 있는 유용한 데이터를 제공할 수 있음을 보여주었습니다. 마찬가지로, Carrega 등 [90]은 위협으로 작용할 수 있는 숨겨진 정보를 탐지하기 위한 일반화된 모델을 구축하기 위해 eBPF를 사용한 데이터 수집 개념을 논의했습니다.

a: 통찰력

• 보안은 네트워크 시스템의 중요한 요구 사항입니다. 다양한 수준의 네트워크 및 시스템 스택에서 방어 메커니즘과 제어를 구현함으로써 무단 액세스와 데이터 또는 리소스의 남용을 방지할 수 있습니다. 여기서 연구된 사용 사례는 다양한 시스템 수준에서 필터링 규칙 및 정책을 적용하고, 패킷 흐름을 변경하고, 분리된 마이크로서비스 간의 트래픽을 모니터링하고 조정하며, 보안 모델을 구축하기 위한 데이터 수집입니다.

• 수집된 데이터는 의심스러운 패킷, 비정상적인 행동, 프로필 위반 및 감사 및 머신 러닝 기반 보안 모델의 훈련을 감지하는 정확하고 효율적이며 신뢰할 수 있는 보안 도구를 개발하는 데 사용됩니다.

• 수집된 데이터로 구성된 필터링 규칙 및 정책은 커널, NIC, 컨테이너/VM 당, 컨테이너/VM 사이, 심지어 함수 기반 레벨(커널에서 함수 호출 전, 후 및 내부)에 적용될 수 있으며, 낮은 오버헤드 프로파일을 유지합니다.

• 분리된 마이크로서비스(지리적으로 분리된 응용 프로그램 또는 서버)의 트래픽을 모니터링하고 조정하는 것은 수집된 데이터 및 패턴을 분석하여 새로 조정된 메트릭 및 구성을 제공하는 데 도움이 됩니다.

토론된 연구의 주요 측면을 요약하고 제안된 설계의 한계와 미래 연구 분야를 가리키는 테이블 2를 참조하십시오.

C. 스토리지

이 하위 섹션은 저장 장치와 그 성능을 향상시키기 위한 방법과 관련된 연구를 요약할 것입니다. eBPF는 메모리 처리 속도를 높이거나 저장 장치에 있는 데이터를 원격으로 실행하는 도구로 사용될 수 있습니다.

1) NVMe 저장소를 위한 커널 바이패스

새로운 비휘발성 메모리 익스프레스(NVMe) 저장 장치의 문제 중 하나는 커널 저장 경로가 액세스 지연의 절반을 담당하기 때문에 발생하는 오버헤드입니다. Zhong 등 [20]은 커널 바이패스 개념을 적용하여 사용자 공간과 커널 공간 사이의 경계를 넘어가는 추가 지연을 피하고 저장 장치에 있는 데이터에 대한 의존적인 요청의 액세스와 탐색을 처리하는 eBPF 프로그램을 설계했습니다. 그들의 벤치마크는 지연 시간을 절반으로 줄이고 입력/출력 작업 수(IOPS)를 2.5배로 늘릴 수 있다는 것을 보여주었습니다.

2) 원격 저장 장치에서 작업 실행

NVM 저장 장치 기술의 급속한 발전과 개선(예: Samsung PM1743 [91]의 13 GB/초(초당 기가바이트)의 읽기/쓰기 속도)은 메모리에서 저장소로의 속도 격차를 줄이게 되었습니다. 반면에 네트워크 장치(예: 이더넷, NIC)는 느린 속도로 개선되고 있습니다. 예를 들어, 전형적인 이더넷 NIC는 초당 1 Gb(기가비트)의 트래픽을 처리할 수 있는데, 이는 저장 장치의 속도(예: 읽기 속도가 초당 7 GB(56 Gb)이고 쓰기 속도가 초당 5 GB(35 Gb)인 Samsung 980 Pro)와 극명한 차이가 있습니다. 이러한 성능 격차로 인해 "데이터 이동 벽"이라는 문제가 발생했습니다.

Kourtis 등 [18]는 eBPF를 사용하여 저장소에서 원격 작업 실행을 지원하기 위한 근거리 메모리 컴퓨팅을 논의하는 프로토타입을 제안했습니다. 저자들은 저장 장치를 프로그래밍하는 데 직면한 세 가지 문제에 중점을 두었습니다: 모든 도메인 및 응용 프로그램을 지원하기 위해 확장성과 안정성 사이의 균형을 유지해야 하며, 실행되는 확장 프로그램은 최소한의 오버헤드와 가능한 최대의 효율성을 가져야 합니다. 또한, 확장 프로그램의 호환성을 강제하여 모든 도메인 및 응용 프로그램을 지원해야 합니다. 원격 실행의 두 가지 경우를 테스트했습니다. 원격 숫자 증가 작업은 네트워크 작업 요청을 절반으로 줄여 지연 시간을 절반으로 줄일 수 있습니다. 이진 트리로 색인된 데이터에서 검색 작업을 오프로딩하면 1 TB의 데이터 크기에서 지연 시간을 86%로 줄일 수 있습니다.

TABLE 2. Summary of eBPF use-cases in the security domain.

Ref.	Place	Role	Key Aspects	Open Challenges and Future Work
[69]	KS	Finding the amount of leaked information to the DNS and alternating packets DNS destinations	Enforce application specific DNS servers User controllability of DNS network traffic	Further studies on DNS (encrypted traffic, filtration and alteration, queries control) Integrability of DoH and DoT
[70]	KS	Tracking Kernel activities at specific tracepoints	Less overhead Small file sizes generated	Supporting all system calls Selective logging and audit logging mechanisms
[71]	KS	Extending and customizing original eBPF framework with containers-level requirements	Attaching on the intersection between namespaces and reference monitor	cgroup lookup overhead
[22]	KS	Implementing Decision Tree-based Machine Learning model in kernel	Performance increase Higher packet inspection rate	Evaluation against other works Alternative machine learning techniques (RF, DNN)
[73]	KS	Capturing Kernel activities	Defining boolean expressions with limitless complexity File integrity monitoring and suspicious process execution detection	Expanding event types Performance evaluation of host overhead
[16]	KS	Building profiles for executables and detect profiles violations	eBPF version Somayaji's pH	Security analysis, integrability, usability and response automation
[78]	KS	Gathering nodes metrics	Dynamic metrics collection Offline pattern analysis Support horizontal autoscaling	Combined online and offline techniques Effects of unknown patterns Integration with existing microservices management systems
[79]	KS	Capturing the starting parameters and ending values of network events-based functions	Running as an active tool with 0.01% event losses Function level policy enforcements	Handling large number of rules Enabling loops like commands
[80]	KS	Allowing containers to push their own designed policies as eBPF code to the host kernel	Per-container policies enforcements Innovating additional eBPF helpers Additional overhead of 0.1%	Validating the new eBPF helpers Helpers' recursion issue Transformation of LSM modules into SNAPPY
[81]	KS	Enforcing the policy on the confined application	Per-container policies enforcements New YAML-based policy configuration language Adding audit logging capabilities	New eBPF helpers to move process groups into per-container new namespaces The removal of the Daemon Integration with existing container solutions
[82]	KS	Enforcing policies on userspace and kernel space functions	Audit data enabled integrable with audit2allow-like	Unified solution for process confinement for known technologies Extendibility to container level policies
[88]	KS	Data gathering for a semi-supervised learning model of resource utilization	Live detection and resource usage monitoring to user/kernel space Attaching application-level interposition probes to request processing functions	Applying more complex machine learning techniques
[89]	KS	Generating traffic filtering rules based on network packets parameters	Hybrid signature and anomaly-based mitigation system Auto XDP rule generation, injection and updating	Packet filtering algorithms and rules limitations Investigating DDoS protections services using content delivery networks (CDNs)
[23], [90]	KS	Data gathering for colluding applications and covert channels	Colluding application and IP-v6 capable covert channels data gathering enabled	Code inspection and deep inspection of execution patterns Extending approach for other threats
[24]	KS	Storing rules information and actions and applying fast pattern matching	High throughput Fast pattern matching algorithm based on Aho-Corasick algorithm Supporting multi-core execution	Handling large eBPF programs Implementing all Snort features Perfectly utilizing all cores
[84]	KS	Monitoring system configurations and executed software commands	Integrating eBPF with Intel PT Scalability with low overhead and precise monitoring	Limited to systems occupied with Intel PT Investigating pure software-based attestation solutions

3) NVMe 존드 네임스페이스를 통한 계산 저장 장치

계산 저장 장치(CSD)는 근처 메모리 컴퓨팅 및 데이터 이동 문제 해결을 위한 대체 접근 방식입니다. CSD의 주요 목표는 사용자가 정의한 프로그램과 작업을 저장 장치에 가능한 가까운 곳에서 실행할 수 있는 프로그래밍 가능한 인터페이스를 제공하여 데이터 이동 비용을 줄이는 것입니다. Lukken 등 [92]이 제시한 제안된 디자인 중 하나는 사용자 공간에서 실행되는 eBPF의 도움을 받아 NVMe 존드 네임스페이스 위에서 작동하는 Zoned Computational Storage Device(ZCSD)입니

다. ZCSD는 여전히 진행 중인 작업입니다. 그러나 초기 프로토타입은 소프트웨어 오버헤드가 낮은 성능 향상의 잠재력을 보여 주었습니다.

4) 사용자 공간 스와핑

Zhong 등 [19]은 페이지 폴트(누락된 페이지) 및 오류(예: 페이지 검색 오류, 손상된 페이지 등)를 처리하기 위해 사용자 공간에서 실행되는 LightSwap라는 새로운 스와핑 방식을 제안했습니다. 이 방법은 느린 커널 데이터 경로를 피하고 스레드 컨텍스트 전환 비용을 줄이기 위해 커널 공간에서 처리되는 것 대신에 사용자 공간에서 실행됩니다. LightSwap은 로컬 및 원격 메모리(NIC 사용)를 사용하여 데이터를 저장합니다. 커널 및 사용자 스택의 읽기/쓰기 지연 분석 결과 대부분의 벌점은 커널 스택에서 발생하며, 이는 사용자 스택보다 최대 10배 느릴 수 있습니다. 이러한 이유로 스왑 작업을 커널 공간에서 처리하는 대신 사용자 공간으로 이동하도록 제안했습니다. LightSwap은 1) 경량 스레드(LWT)의 사용 및 전통적인 스왑 메커니즘과의 통합, 2) eBPF 기술을 사용하여 eBPF 맵에 기본 스레드 컨텍스트와 페이지 폴트 컨텍스트(LWT 폴트에 의한)를 저장하고, 3) 새로운 try-catch 예외 프레임워크에 기반합니다. 페이지 폴트를 처리하기 위해 eBPF 맵은 사용자 공간과 커널 공간 간의 컨텍스트(스레드, LWT 페이지 폴트)를 공유하는 공유 장소로 사용될 수 있습니다. 페이지 폴트가 발생하면 현재 LWT가 유지될 것이며(현재 컨텍스트를 저장함), 페이지가 가져와질 때까지 대기한 후 실행이 다시 시작됩니다. LWT 폴트 처리(페이지 스왑)는 비동기적으로 수행되므로 다른 LWT가 폴트를 해결하는 동안 실행될 수 있습니다. 저자들은 페이지 폴트 처리 지연 시간을 3-5배로 줄이고 처리량을 40% 향상시킬 수 있는 것을 보여주어 Infiniswap [93]과 비교하여 LightSwap이 가능한 페이지 폴트 처리를 감소시킬 수 있다는 것을 보였습니다.

a: 인사이트

• eBPF는 다양한 유형의 데이터 구조 및 저장 기술(SSD, NVMe 등)의 효율적인 액세스 메커니즘을 제공하고 간단한 명령 및 작업을 로컬 저장소에서 직접 실행하거나 원격 저장소로 보낼 수 있도록 확장됩니다.

• eBPF를 활용하여 저장소에 블록 및 페이지를 명확하게 볼 수 있도록 데이터 구조를 최적화함으로써 사용자는 데이터에 효율적으로 액세스할 수 있습니다. 이는 데이터 검색 시간과 응용 프로그램이 데이터를 가져오기 위해 수행하는 요청 수를 줄입니다. 이로 인해 더 적은 지연, 시간당 더 많은 액세스 및 저장소에서 발생하는 오버헤드가 줄어듭니다.

• eBPF 헤더로 요청을 캡슐화하면 안전성 보장이 증가하고 커널과 저장소 사이에 배치 및 유지할 수 있습니다. 이로써 사용자 정의 프로그램이 커널을 우회하고 독립적인 요청 및 의존적인 요청 체인의 오버헤드를 무시할 수 있습니다.

• 마지막으로 사용 사례는 eBPF를 활용하여 로컬 저장소 설계 및 원격 위치(분리된 하드웨어)에 저장되는 데이터에 대한 요청(단일, 반복, 청크, 종속적 및 독립적)의 대안적 솔루션을 탐색합니다.

표 3은 논의된 연구의 요약과 주요 측면, 제안된 디자인의 제한 사항 및 탐색할 남은 미래 분야를 보여줍니다.

D. SANDBOXING

프로그램이 배포 제약 조건을 충족하는지 확인하는 과정을 샌드박싱이라고 합니다. 컴파일러는 코드가 환경 규칙과 제약 조건을 준수하는지 확인하기 위해 코드를 다시 컴파일합니다. eBPF에서는 정적 검증 및 JIT 컴파일 단계가 샌드박싱 프로세스를 나타내며, 이러한 단계는 섹션 II에서 설명한대로 바이트 코드를 출력합니다. 샌드박싱은 eBPF 실행의 효율성을 보장하면서 형식적으로 올바르고 안전한 보장이 포함된 최적화된 바이트 코드 버전을 생성합니다. 여기에는 eBPF를 사용한 프로그램 샌드박싱에 대한 몇 가지 연구가 제시됩니다.

1) eBPF 프로그램 합성

최근 Xu 등 [94]의 연구에서는 eBPF 바이트 코드를 최적화하는 중요성과 잠재력을 보여 주었습니다. 저자들은 프로그램 합성 최적화된 eBPF 바이트 코드 컴파일러인 K2를 개발했습니다. 그들의 결과는 바이트 코드 크기를 6-26% 감소시키고, 코어 당 초당 패킷 개수를 기준으로 0-4.75% 더 나은 처리량을 얻고, 평균 패킷 처리 지연 시간을 1.36-55.03% 낮출 수 있다는 것을 보여 주었습니다. K2는 eBPF 정적 검증기와 독립적으로 작동하며 코드에 대한 두 가지 다른 세트의 확인이 있습니다. K2는 코드를 다양한 출력으로 컴파일하고 이를 eBPF 검증기에 전달하여 검증기에서 거부된 것을 제외합니다. 그런 다음 검증을 통과한 버전의 성능을 확인하고 그들의 지연 시간, 크기 및 처리량을 비교합니다. 저자들에 따르면, K2 컴파일에 더 많은 시간을 할애할수록 최적화된 결과를 얻을 수 있습니다.

TABLE 3. Summary of eBPF use-cases in the storage domain.

Ref.	Place	Role	Key Aspects	Open Challenges and Future Work
[20]	KS	Traversing persistent application-defined data structures in the Kernel	New high-level fast storage accessing library New eBPF compatible with NVMe hook	Exploring caching and scheduler policies Safety guarantees of the new NVMe hook
[18]	SS	Shipping the execution of simple codes within the network request contents of remote storage to be directly executed on the storage	Supporting eBPF appcode using NBD protocol Running verification and execution in storage devices	Unsupported loops operations and clang loop unrolling issues Proving total program (program termination) Supporting complicated protocols
[19]	KS	Storing faulting LWT context and restoring the default context	User-space page fault handling mechanism Novel try-catch framework to deal with paging errors Latency improvements and throughput increase	Overcoming storage stack overhead Use of LWT in distributed share memory systems
[92]	US	Support programmability with ZNS SSDs	Low footprint Easy programmability, extensibility, and analysis	Integration with libnvme and zoned-enabled file system for storage Supporting different hardware and additional data structures

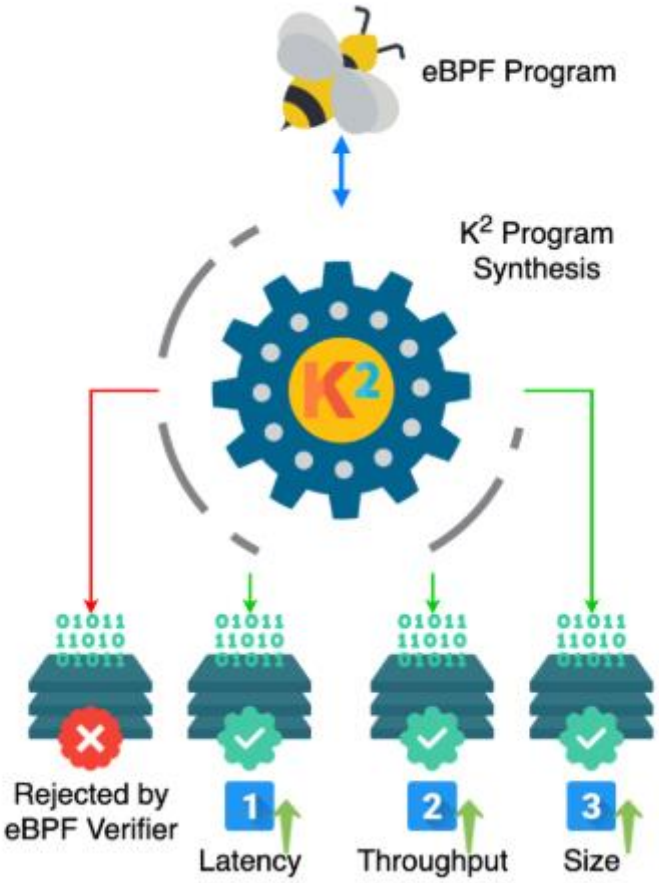


그림 16 eBPF 프로그램 합성

D. SANDBOXING

프로그램이 배포 제약 조건을 충족하는지 확인하는 과정을 샌드박싱이라고 합니다. 컴파일러는 코드가 환경 규칙과 제약 조건을 준수하는지 확인하기 위해 코드를 다시 컴파일합니다. eBPF에서는 정적 검증 및 JIT 컴파일 단계가 샌드박싱 프로세스를 나타내며, 이러한 단계는 섹션 II에서 설명한대로 바이트 코드를 출력합니다. 샌드박싱은 eBPF 실행의 효율성을 보장하면서 형식적으로 올바르고 안전한 보장이 포함된 최적화된 바이트 코드 버전을 생성합니다. 여기에는 eBPF를 사용한 프로그램 샌드박싱에 대한 몇 가지 연구가 제시됩니다.

1) eBPF 프로그램 합성

최근 Xu 등 [94]의 연구에서는 eBPF 바이트 코드를 최적화하는 중요성과 잠재력을 보여 주었습니다. 저자들은 프로그램 합성 최적화된 eBPF 바이트 코드 컴파일러인 K2를 개발했습니다. 그들의 결과는 바이트 코드 크기를 6-26% 감소시키고, 코어 당 초당 패킷 개수를 기준으로 0-4.75% 더 나은 처리량을 얻고, 평균 패킷 처리 지연 시간을 1.36-55.03% 낮출 수 있다는 것을 보여 주

었습니다. K2는 eBPF 정적 검증기와 독립적으로 작동하며 코드에 대한 두 가지 다른 세트의 확인이 있습니다. K2는 코드를 다양한 출력으로 컴파일하고 이를 eBPF 검증기에 전달하여 검증기에서 거부된 것을 제외합니다. 그런 다음 검증을 통과한 버전의 성능을 확인하고 그들의 지연 시간, 크기 및 처리량을 비교합니다. 저자들에 따르면, K2 컴파일에 더 많은 시간을 할애할수록 최적화된 결과를 얻을 수 있습니다.

2) IoT 마이크로컨트롤러의 가상 머신

인터넷 익스플로러(이하 IoT) 기기와 저전력 소비 마이크로컨트롤러의 급속한 증가는 마이크로컨트롤러 프로그램을 실행하는 동적이고 확장 가능한 격리된 환경의 등장을 가능하게 했습니다. Zandberg와 Baccelli에 따르면 [95] 마이크로컨트롤러 응용 프로그램을 위한 적합한 격리된 환경을 구축하기 위한 요구 사항은 다음과 같습니다: 메모리 풋 프린트, 메모리 보호를 위한 추가 하드웨어 설치, 실행 속도 저하에 대한 관용 가능한 코드 실행 속도 저하 및 응용 프로그램 업데이트를 위한 전송되는 데이터 양. 저자들은 eBPF를 이용하여 마이크로컨트롤러 프로그램을 호스팅하기 위한 작은 가상 머신을 빌드하는 새로운 흥미로운 접근법을 논의했습니다. 그들은 rBPF를 WASM3 [96]과 네이티브 C 구현과 비교하여 ROM 및 RAM 사용량, 코드 크기 및 실행 시간에 대해 비교했습니다. 초기 결과는 rBPF가 RAM 및 ROM 사용량에서 WASM3과 네이티브 C 모두를 15차수로 이겼으며 1.3백만명의 명령을 초당 실행할 수 있었습니다. 그러나 코드 크기와 실행 시간의 경우 rBPF가 WASM3과 네이티브 C보다 큰 차이로 뒤처지지만, 이러한 오버헤드는 마이크로컨트롤러가 계산 집중적인 사용 사례를 위해 만들어지지 않았기 때문에 무시될 수 있습니다.

3) 임베디드 디바이스 펌웨어 패칭

eBPF의 일반성과 저 실행 요구 사항 (eBPF를 사용하면 eBPF 기반 프로그램을 실행하기 위해 시스템을 다시 부팅하거나 재시작할 필요가 없음)은 취약점이 존재하는 경우 시스템 흐름을 리디렉션하여 적절한 조치가 취해질 때까지 패치를 실행하는 데 적합한 솔루션으로 만들었습니다. [97]에서는 임베디드 디바이스를 위한 펌웨어 핫 패칭 프레임워크를 만들기 위한 eBPF의 사용 사례를 제시했습니다. 공급 업체는 취약점이나 버그가 발견되면 여러 임베디드 디바이스에 동시에 패치를 적용하는 데 어려움을 겪고 있습니다. 이 문제는 다양한 임베디드 디바이스에 대한 일반적인 펌웨어 패치 솔루션을 찾는 중요성을 높였으며 여기에 eBPF의 역할이 있습니다. 저자들은 RapidPatch를 개발했으며, 이는 이질적인 임베디드 디바이스에 대한 일반적인 패치를 생성하여 패치 개발 및 배포 과정을 가속화할 수 있는 펌웨어 패치 프레임워크입니다.

4) 하드웨어 기반 안드로이드 네이티브 코드 분석기

Zhou 등[98]은 현대 안드로이드 시스템의 내장 트레이스 마이크로셀 (ETM) [99]과 eBPF의 내장된 기능 및 구성 요소를 활용한 새로운 하드웨어 기반 안드로이드 네이티브 코드 분석기인

NCScope를 제안했습니다. 이 연구에서는 1) 금융 앱에서의 자가 보호 (SP)를 통해 응용 프로그램의 실행 구조와 데이터를 구현하고 보호하는 앱의 비율을 확인하고, 2) 악성 앱에서의 반 분석 (AA) 방어를 통해 코드 분석 소프트웨어에 의해 감지

되지 않고, 3) 메모리 훼손 감지 및 4) 네이티브 코드 함수의 성능 비교 등 네 가지 사용 사례를 제시했습니다. 기존 솔루션에서의 문제점은 불완전한 명령어 추적, 높은 오버헤드입니다. 이는 악성 앱에서의 반 방어 메커니즘을 트리거로 사용할 수 있으며, 실제 시스템이 아닌 에뮬레이터에서 수행된 테스트와 분석 때문에 신뢰할 수 없는 분석입니다. ETM은 목표 주소를 기반으로 명령어 추적에 사용되어 앱의 동작을 추적하고, eBPF는 실행 시간에 함수 호출의 매개 변수를 기반으로 앱 메모리 데이터를 수집합니다. 결과는 NCScope가 시스템에 최대 40배까지 느려질 수 있는 다른 솔루션과 비교하여 최소한의 오버헤드를 소개한다는 것을 보여주었습니다. 또한 900개의 금융 앱 (은행, 지갑, 암호화폐, 결제 앱 등) 중 26.8%가 코드에 SP 메커니즘을 구현했으며, 450개의 테스트된 악성 앱 중 78.3%가 감지되지 않도록 AA 방어를 가지고 있습니다. 또한 수집된 데이터는 메모리 버그를 진단하고 네이티브 코드 함수의 성능을 평가하는 데 유용한 정확한 정보를 제공할 수 있습니다.

a: 통찰 • 샌드박싱 연구의 주요 목표는 시스템 사양 및 요구 사항과는 독립적으로 다양한 기기를 위한 재최적화된 코드 (eBPF 바이트 코드, 패치)를 호스팅하고 프로그램 및 그들의 숨겨진 기능을 분석하기 위해 eBPF 가상화를 사용하여 더 효율적인 환경(통합된 보안)을 만드는 것입니다. 테이블 4는 샌드박싱 관련 연구를 요약합니다.

TABLE 4. Summary of eBPF use-cases in the sandboxing domain.

Ref.	Place	Role	Key Aspects	Open Challenges and Future Work
[94]	KS	Executing the old and the new eBPF programs to check their cost functions	Novel synthesis-based eBPF bytecode optimization compiler Adapting stochastics synthesis techniques for BPF New equivalence and safety checker for BPF programs	Scaling to larger programs optimization Proper cost measurement metrics Syncing K2 checker with kernel checker
[95]	KS	Hosting process in an isolated unified environment	Process isolation for low-power microcontroller Better memory usage and utilization	Improving execution time overhead and reducing power consumption and program size Sandboxing guarantees of rBPF
[97]	KS	Distributing and containerizing the firmware patches	Low execution requirements needed Generating device specific code with additional verification step Redirecting execution flow to the desired patch without modifying the ROMs/firmware	Patching large sized features Guaranteeing the patch to be bug-free Distributing the patch on non-connected devices
[98]	KS	Collects and inspect function calls parameters and memory data	Integrating eBPF with ETM Extremely less overhead Undetectable by anti-analysis apps	Integrating with automated testing tools Evaluating more apps Extending the set of SP and AA behavior detection rules

4. 사용자 공간

사용자 공간 사용 사례는 대부분의 설계가 시스템의 흐름을 중단하지 않고 가능한 최초의 단계에서 데이터 패킷을 처리하는 데 초점을 맞추기 때문에 매우 제한적입니다. 예를 들어, XDP를 사용하면 네트워크 스택에 도달하기 전에 도착한 패킷을 처리할 수 있습니다. 또한, XDP를 통해 패킷을 커널을 거치지 않고 사용자 공간에 전달하여 처리할 수 있습니다. 그럼에도 불구하고, 사용자 공간 처리 연구와 관련된 일반적인 아이디어를 추출하고 일반화하는 것이 유익할 수 있습니다.

특히, 검토된 작업의 목적과 구현을 조사한 후, 우리는 이러한 기술적 해결책이 eBPF와 상호 작용하는 관점에서 다른 분류 접근 방식을 제시하기로 결정했습니다. 그림 17에서 보여진 것처럼, 우리는 또한 연구를 eBPF 주입 모드, 위치, 유형 및 eBPF 맵 모드 및 유형과의 상호 작용에 따라 분류했습니다. 그러나 이 분류는 사용자 공간 수준에서의 eBPF 사용을 밝히기 위해 목적을 가진 것으로 범위가 더 제한적입니다. 따라서 이는 네트워킹, 보안, 저장 및 샌드박싱 수준에서의 eBPF 사용을 밝히는 우리의 보다 일반적인 접근 방식만큼 유익하지 않을 수 있습니다. 그러나 우리는 완전성을 위해 이를 여기에 포함시켰습니다.

A. eBPF 주입

eBPF 주입은 시스템 위치에 eBPF 코드를 첨부하는 프로세스를 의미합니다. 주입 모드는 i) 동적/자동으로 eBPF 프로그램이 언제 eBPF 코드를 첨부할지 결정하는 경우와 ii) 사용자가 제어하고 첨부할 것을 결정하는 경우로 나뉩니다. 우리의 관측 결과, 주입 위치에는 i) 네트워크 하드웨어를 떠나 커널 공간과 상호 작용하는 패킷의 첫 번째 지점인 eXpress Data Path (XDP), ii) 커널 공간의 커널 스택, iii) 사용자 공간 및 iv) 원격 저장소와 같은 원격 하드웨어가 포함됩니다. 그런 다음 주입되는 코드와 추적 지점의 형태를 살펴보았는데, 이를 유형이라고 합니다. 우리는 이러한 유형을 i) 특정 작업을 수행하기 위해 사용자 정의된 코드를 나타내는 eBPF 프로그램, ii) 커널 함수를 추적하는 커널 프로브 (예: kprobe 및 커널 추적 지점), 그리고 iii) 사용자 공간 함수를 추적하는 사용자 공간 프로브 (예: uprobes)로 그룹화했습니다. 다음은 서로 다른 eBPF 주입 모드, 위치 및 유형의 조합의 예입니다:

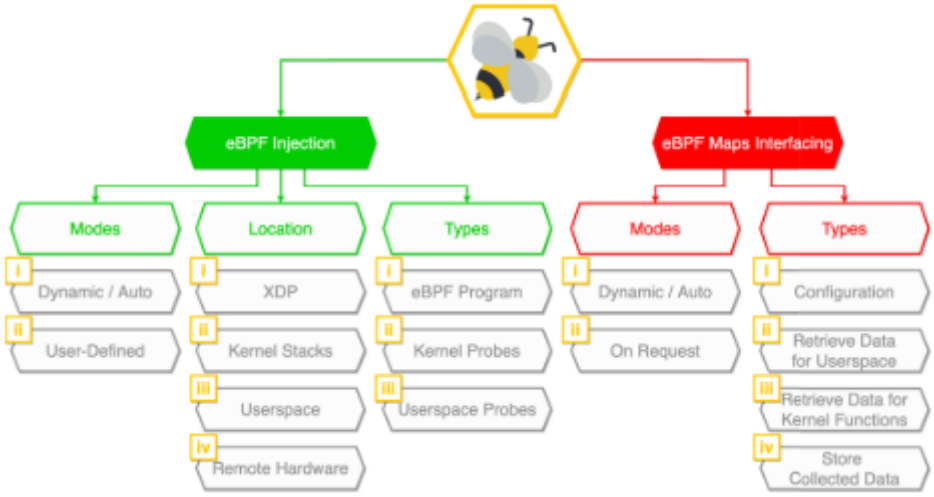


그림 17 eBPF에 대한 사용자 공간 관계의 유형

1) eBPF 디자인은 커널 프로브에서 수집된 데이터를 분석하여 자동 생성된 필터링 규칙을 호스팅하기 위해 XDP (위치)를 사용합니다.

2) 응용 프로그램 개발자는 사용자 공간 (위치)에 사용자 정의된 uprobes (유형)를 첨부하여 사용자 공간 함수의 흐름을 추적합니다.

3) 데이터 센터에서는 단순한 명령 (예: 증분 작업) (유형)을 원격 저장소 (위치)에서 실행할 수 있습니다.

B. eBPF 맵 인터페이싱

eBPF 맵은 주로 데이터를 저장하는 데 사용됩니다. 그들은 i) 값이 사용자의 간섭 없이 업데이트되거나 검색되는 자동/동적 모드 및 ii) 사용자가 수집하거나 업데이트하려는 데이터를 이벤트를 트리거하여 수집하는 요청 모드로 두 가지 모드를 갖습니다. 인터페이싱의 유형은 다음과 같이 네 가지입니다: i) 필요한 양과 유형의 데이터를 사전에 정의하고 정의하기 위해 eBPF 맵을 구성하는 것, ii) 사용자 공간으로 저장된 데이터를 검색하여 사용자에게 표시하고 시각화하거나 사용자 공간에서 추가로 처리하는 것, iii) 커널 공간 함수에 대한 저장된 데이터의 검색 및 iv) 수집된 데이터의 저장 모든 eBPF 추적 지점 유형에서입니다. 어떤 면에서는 모든 설계가 eBPF 맵을 처리해야 하므로 모든 설계에는 어떤 형태의 사용자 공간 상호 작용이 있습니다. 다음은 서로 다른 인터페이싱 모드 및 유형의 조합의 예입니다:

a) XDP에 배치된 eBPF 프로그램은 다양한 IP 주소의 패킷 수를 계산하고 이를 이상 행동 분석 도구로 전달합니다.

b) 네트워크 관리자가 정의한 왕복 시간 제한을 초과하는 패킷을 재전송하기 위해 저장된 패킷을 사용하는 TCP 모니터링 eBPF 기반 프로그램이 개발되었습니다.

c) 특정 애플리케이션 흐름의 트래픽을 캡처하여 다양한 구현 방법으로 성능을 분석하는 최적

화 도구입니다.

우리가 본 바에 따르면, 모든 연구는 주로 uprobes를 통해 사용자 공간 함수 사용을 추적하거나 시각화, 추가 분석 및 처리를 위해 eBPF 맵에서 데이터를 검색하는 것과 관련되어 있습니다. 일부 연구에서는 사용자 공간 범위를 완전히 지정하고 커널 공간과 구분합니다. 그러나 대부분의 작업에서는 사용자 공간의 역할이 eBPF 맵에서 데이터를 검색하는 데 있습니다.

V. 미래 연구 방향

주요 장점으로 인해 eBPF는 다양한 신생 응용 프로그램에 대해 산업 및 학계에서 거대한 관심을 끌었습니다. 수많은 eBPF 사용 사례를 분석하면서 일부 제한 사항이 발견되었으며, 이는 이 혁신적인 기술의 능력을 향상시키고 전체 잠재력을 실현하기 위한 흥미로운 미래 연구 기회를 제공합니다.

eBPF 코드 안전성 속성은 커널 내에서 실행되기 전에 정적 분석기에 의해 확인되고 검증됩니다. 그러나 정적 체커는 중간 정도의 크기의 코드조차도 검증할 수 없으며, 이는 허용 가능한 프로그램의 복잡성을 제한합니다. 더 나아가 성능 향상을 위해 간결한 eBPF 코드를 생성하는 것은 번거로운 작업입니다. 공식적인 방법은 이러한 문제를 해결하기 위해 최전선에 있습니다. Nelson 등은 [100]에서 상당한 크기의 코드의 정확성을 검증할 수 있는 자동화된 증명 전략을 보고했습니다. 마찬가지로, Vishwanathan 등은 [101]에서 산술 연산자 (덧셈, 뺄셈)의 음성 및 최적성을 증명하고, tnums(삼 상태 숫자) 영역에서 더 나은, 정확하고 빠른 곱셈 알고리즘을 제안했습니다. 음성 보증을 갖춘 eBPF 프로그램의 개선된 검증은 모든 취약성을 제거하고 어떤 종류의 공격으로부터 보호하는 데 중요합니다. 최근 Demoulin 등은 [94]에서 공식적인 정확성과 안전성 보증을 갖춘 강력한 프로그램 합성 기반 컴파일러를 제안하여, 중요한 기여를 했습니다. 그러나 성능과 음성 보증을 갖추고 큰 프로그램을 빠르게 검증하고 최적화하기 위해 이 분야에서 더 많은 후속 연구가 필요하며, 리눅스 도구 체인과 통합하여 eBPF 능력을 확장하여 리소스 제한된 저전력 IoT 기기와 같은 보다 넓은 응용 프로그램에 풍부한 기능을 제공하는 것이 목표입니다.

대부분의 eBPF 작업이 보안 보증에 기반을 두었지만, 일부 취약성과 문제가 발견되었으며, 공격자가 그것들을 악용하기 전에 해결해야 합니다. eBPF 기반 응용 프로그램을 보안하는 것은 eBPF 검증과는 관계없이 주요 관심사입니다. 이러한 eBPF 기반 프로그램은 검증기를 우회하고 시스템을 해치는 데 악용될 수 있는 새로운 기능을 소개합니다. 예를 들어, SNAPPY [80]가 오용되면, 악의적인 관리자가 악의적인 커널 모듈을 삽입하여 DoS 공격을 유도할 수 있습니다. 다른 작업에서는 개발자의 역할이 eBPF 기반 프로그램이 버그 없이 작동하고 적절히 작동하는 것을 보장하는 데 중요하다고 강조합니다. I/O의 적절한 모니터링은 Cryptolockers(랜섬웨어)에 대한 효과적인 지표가 될 수 있습니다. IoT 장치의 배터리를 고갈시키거나 DoS 공격이 발생할 수 있으며, 이는

eBPF 기반 프로그램의 실행 시간 경계를 놓치는 결과일 수 있습니다. 또한 eBPF 보안의 가장 중요한 측면은 eBPF 검증기 자체의 정확성과 보안 보증을 보장하는 것입니다. 잘못된 ALU32 경계로의 업데이트는 경계 외 코드 실행을 초래하고, 따라서 DoS 공격을 유발하며, 이는 로컬 권한 상승(LPE)에 사용될 수 있습니다. eBPF 검증기에 관한 여러 취약성이 CVE 데이터베이스에 나열되어 있으며, 이는 검증기를 재검토하고 그의 견고성과 정확성을 보장하는 중요성을 강조합니다.

eBPF는 보안 분석을 위한 강력한 데이터 수집 도구이며, 호스트 기반 [16], 기계 학습 [22], IPv6 트래픽 흐름 내의 숨겨진 채널 탐지 [23]와 같은 여러 분야에서 침입 탐지에 사용되었습니다. [22]에서는 흐름 기반 의사 결정 트리를 사용하여 기계 학습 모델을 개발하여 상당한 성능 이점을 제공했습니다. 다른 복잡한 기계 학습 모델, 예를 들어 랜덤 포레스트 또는 딥 뉴럴 네트워크를 eBPF를 사용하여 조사하는 것은 중요하고 흥미로운 방향입니다. 이는 부하 분산 [32], 포렌식 분석을 위한 컨테이너 감사 메커니즘 [71] 등의 eBPF 기반 응용 프로그램에 대한 기계 학습 기반 응용 프로그램의 새로운 지평을 열 것입니다. 암호 잠금기와 같은 보다 넓은 범위의 위협을 탐지하고 에너지 소비, RAM 사용 패턴 등의 다른 성능 메트릭을 분석하기 위한 eBPF의 기계 학습 기반 응용 프로그램에 대한 새로운 방향을 탐색하는 것이 중요합니다.

네트워킹은 eBPF 응용 분야 중 하나입니다. 최근 eBPF는 표준 도메인 이름 시스템(DNS), DNS over TLS 및 DNS over HTTPS 통신의 개인 정보 보호를 강화하기 위해 낮은 오버헤드로 사용되었습니다. 그러나 [69] 및 [105]의 저자들이 지적한 바와 같이 암호화된 DNS 트래픽을 분석하기 위해서는 더 많은 연구가 필요합니다. 마찬가지로, [42]의 저자들은 TCP의 혼잡 제어를 위해 실시간 인밴드 네트워크 텔레메트리 정보를 얻기 위해 eBPF를 사용했습니다. 이런 선도적인 작업은 [42]에서 보고한 바와 같이 흐름 제어, TCP 공정성 등을 탐색하기 위해 여러 흥미로운 방향을 엽니다. 최근에는 대부분의 인터넷 트래픽을 위해 WiFi 링크를 사용하는 추세입니다. 최근 eBPF를 WiFi 액세스 포인트에 적용하여 에너지 소비 및 데이터 패킷 전환 지연을 결정하는 것은 또 다른 흥미로운 연구 기회를 제공합니다.

저장 지연 시간 및 I/O 작업은 고성능 컴퓨터 시스템에서 병목 현상이 될 수 있습니다. 따라서 eBPF는 신흥 저장 장치의 저장 액세스 지연 시간을 줄이거나 시스템 호출의 성능 비용을 줄이는 데 사용되었습니다. [20] 및 [106]에서 언급된 것처럼 이러한 흥미로운 작업은 압축 및 중복 제거를 포함한 다른 주요 저장 작업을 커널로 오프로드하는 등의 추가 탐색 가능성을 제공합니다. 메모리 내 처리를 위해 비휘발성 메모리 장치를 기반으로 하는 프로그래밍 가능한 계산 저장소의 또 다른 유망한 방향은 CPU 기반 시스템의 성능 병목 현상을 극복하기 위해 [107]입니다. [107]에서 강조된 바와 같이 인기있는 응용 프로그램 수준 데이터 구조 (B+ 트리, 해시 테이블 등)에 대한 내부 데이터 처리를 위한 내부 지원을 제공하기 위한 미래 확장이 가능합니다. 또한, 비전 논

문에서 저자들이보고한 것처럼, 주요 응용 분야에 대한 eBPF 기반 프로그래밍 가능한 저장소 서비스는 신흥 엣지 컴퓨팅 패러다임을 위해 탐구되고 있습니다. 이 저장소 서비스는 복제, 일관성, 가비지 수집과 같은 응용 프로그램별 사용자 정의를 가능하게 하며, 일부 계산을 저장소 서비스로 오프로드하여 자율 주행, 증강 현실 및 실시간 비디오 분석과 같은 혁신적인 응용 프로그램의 낮은 대기 시간 요구 사항을 충족시킵니다.

클라우드 서비스가 보급되면서 데이터 센터는 그들의 대규모 에너지 소비로 인해 탄소 과도로 변모되고 있으며, 그 결과 기후에 미치는 영향으로 인해 전 세계적인 우려를 불러일으킵니다. 따라서 데이터 센터의 탄소 발자국을 줄이는 것은 중요한 설계 목표이며, 이는 하드웨어 및 소프트웨어 수준에서 해결할 수 있습니다. 최근 [109]에서 보고된 소프트웨어 중심 접근 방식에서는 성능 및 탄소 배출량 모두를 세밀한 기준으로 응용 프로그램 개발자에게 가시적으로 표시하기 위해 시스템 API를 통해 고도의 리소스 사용을 선택하여 정보에 기반한 트레이드오프를 할 수 있어야 합니다. 우리는 eBPF가 세밀한 수준에서 에너지 및 자원 사용을 추적하여 서비스 수준 협약 (SLA) 내에서 적절한 리소스 선택을 자동으로 수행함으로써 데이터 센터의 탄소 발자국을 줄이는 데 중요한 역할을 할 수 있다고 기대합니다.

마지막으로, 통합된 성능 및 평가 메트릭 및 테스트 정책이 eBPF의 다양한 사용 사례를 올바르게 평가하는 데 필요할 수 있습니다. eBPF는 아직 활발히 개발 중이며, 새로운 기능이 정기적으로 발표되고 있으며, 이는 가까운 미래에 새로운 응용 프로그램이 등장하는 길을 열 것입니다.

5. 결론

클라우드 서비스의 현격한 성장으로 리소스 프로비저닝, 트래픽 엔지니어링, 보안 및 엄격한 QoS/QoE 요구 사항에 대한 새로운 세대의 깊은 모니터링 및 검사 도구가 필요합니다. 이런 관점에서 eBPF는 클라우드 서비스에서 이러한 도전에 대처하기 위한 유망한 후보 기술 중 하나로 등장했습니다. 신흥 클라우드 응용 프로그램에서의 eBPF의 보급을 인식하여, 우리는 이 혁신적인 기술의 응용 프로그램 풍경을 제시했습니다. 주요 목표는 연구 및 제품 개발자들을 위한 eBPF 응용 프로그램의 사용 사례 중심의 포괄적인 안내서를 제공하는 것이었습니다. 우리는 eBPF에 대한 배경 지식으로 시작하여, 그 주요 특징과 능력을 강조했습니다. 그런 다음, 네트워킹, 보안, 저장 및 샌드박싱 분야와 관련된 eBPF의 네 가지 주요 응용 프로그램 도메인을 조사했습니다. 각 응용 프로그램 도메인에 대해, 필수적인 특성과 장단점을 갖춘 다양한 사용 사례를 논의하고 요약했습니다. 마지막으로, 이 기술이 제공하는 거대한 잠재력을 활용하기 위한 몇 가지 흥미로운 연구 기회를 제안했습니다. 특히, 대규모 eBPF 바이트 코드의 빠른 검증 및 최적화, 보안 및 네트워크 관리를 위한 eBPF 기반 기계 학습 응용 프로그램, 신흥 엣지 컴퓨팅 패러다임을 위한 eBPF 기반 프로그래밍 가능한 저장소 서비스, 데이터 센터의 탄소 발자국 줄이기 등 네 가지 유망한 분야를 식

별했습니다. 이 연구가 다채로운 eBPF 풍경에서의 추가 발전을 영감을 주어 클라우드 응용 프로그램의 발전에 기여할 것으로 기대합니다.

REFERENCES

[1] B. Varghese and R. Buyya, "Next generation cloud computing: New

trends and research directions," Future Gener. Comput. Syst., vol. 79,

pp. 849–861, Feb. 2018.

[2] A. Bhardwaj and C. R. Krishna, "Virtualization in cloud computing:

Moving from hypervisor to containerization—A survey," Arabian J. Sci.

Eng., vol. 46, no. 9, pp. 8585–8601, Sep. 2021.

[3] E. Casalicchio and S. Iannucci, "The state-of-the-art in container technologies: Application, orchestration and security," Concurrency Comput., Pract. Exper., vol. 32, no. 17, Sep. 2020, Art. no. e5668.

[4] B. Yi, X. Wang, S. K. Das, K. Li, and M. Huang, "A comprehensive

survey of network function virtualization," Comput. Netw., vol. 133,

pp. 212–262, Mar. 2018.

[5] A. Kiani and N. Ansari, "Profit maximization for geographically dispersed green data centers," IEEE Trans. Smart Grid, vol. 9, no. 2,

pp. 703–711, Mar. 2018.

[6] M. Waseem, P. Liang, and M. Shahin, "A systematic mapping study

on microservices architecture in DevOps," J. Syst. Softw., vol. 170,

Dec. 2020, Art. no. 110798.

[7] N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi,

R. Mustafin, and L. Safina, "Microservices yesterday, today, and tomorrow," Present Ulterior Softw. Eng., pp. 195–216, 2017, doi: 10.1007/978-

3-319-67425-4_12.

[8] M. Waseem, P. Liang, M. Shahin, A. D. Salle, and G. Márquez, "Design,

monitoring, and testing of microservices systems: The practitioners' perspective," J. Syst. Softw.,

vol. 182, Dec. 2021, Art. no. 111061.

[9] S. Yang, F. Li, Z. Zhou, X. Chen, Y. Wang, and X. Fu, "Online

control of service function chainings across geo-distributed datacenters," IEEE Trans. Mobile Comput., early access, Dec. 15, 2021, doi:

10.1109/TMC.2021.3135535.

[10] F. K. Parast, C. Sindhav, S. Nikam, H. I. Yekta, K. B. Kent, and S. Hakak,

''Cloud computing security A survey of service-based models," Comput.

Secur., vol. 114, Mar. 2022, Art. no. 102580.

[11] M. Kleehaus and F. Matthes, "Challenges in documenting microservicebased IT landscape: A survey from an enterprise architecture management

perspective," in Proc. IEEE 23rd Int. Enterprise Distrib. Object Comput.

Conf. (EDOC), Oct. 2019, pp. 11–20.

[12] S. Karumuri, F. Solleza, S. Zdonik, and N. Tatbul, "Towards observability data management at scale," ACM SIGMOD Rec., vol. 49, no. 4,

pp. 18–23, Mar. 2021.

[13] H. Ning, H. Wang, Y. Lin, W. Wang, S. Dhelim, F. Farha, J. Ding, and

M. Daneshmand, "A survey on metaverse the state-of-the-art, technologies, applications, and challenges," 2021, arXiv2111.09673.

[14] S. McCanne and V. Jacobson, "The BSD packet filter A new architecture

for user-level packet capture," in Proc. USENIX winter, vol. 46, 1993,

pp. 1–11.

[15] eBPF. Accessed: Oct. 20, 2021. [Online]. Available: https://ebpf.io/

[16] W. Findlay. (Apr. 2020). Host-Based Anomaly Detection With Extended

BPF. httpswww.cisl.carleton.ca~willwrittencourseworkundergrad-ebpHthesis.pdf

[17] N. Hedam, "EBPF-from a programmer's perspective," EasyChair, London, U.K., Tech. Rep. 5198, 2021.

[18] K. Kourtis, A. Trivedi, and N. Ioannou, "Safe and efficient remote

application code execution on disaggregated NVM storage with eBPF," 2020, arXiv:2002.11528.

[19] K. Zhong, W. Cui, Y. Lu, Q. Liu, X. Yan, Q. Yuan, S. Luo, and K. Huang, ''Revisiting swapping in user-space with lightweight threading," 2021, arXiv:2107.13848.

[20] Y. Zhong, H. Wang, Y. J. Wu, A. Cidon, R. Stutsman, A. Tai, and J. Yang, ''BPF for storage: An exokernel-inspired approach," in Proc. Workshop Hot Topics Operating Syst., Jun. 2021, pp. 128–135.

[21] M. A. M. Vieira, M. S. Castanho, R. D. G. Pacífico, E. R. S. Santos, E. P. M. C. Júnior, and L. F. M. Vieira, "Fast packet processing with eBPF and XDP: Concepts, code, challenges, and applications," ACM Comput. Surv., vol. 53, no. 1, pp. 1–36, Jan. 2021.

[22] M. Bachl, J. Fabini, and T. Zseby, "A flow-based IDS using machine learning in eBPF," 2021, arXiv:2102.09980.

[23] L. Caviglione, W. Mazurczyk, M. Repetto, A. Schaffhauser, and M. Zuppelli, "Kernel-level tracing for detecting stegomalware and covert channels in Linux environments," Comput. Netw., vol. 191, May 2021, Art. no. 108010.

[24] S.-Y. Wang and J.-C. Chang, "Design and implementation of an intrusion detection system by using extended BPF in the Linux kernel," J. Netw. Comput. Appl., vol. 198, Feb. 2022, Art. no. 103283.

[25] Sunos 4.1.1 Reference Manual, S. M. Inc, Mountain View, CA, USA, Oct. 1990.

[26] BIBentryALTinterwordspacing T. Hoiland-Jorgensen, J. D. Brouer, D. Borkmann, J. Fastabend, T. Herbert, D. Ahern, and D. Miller, "The express data path Fast programmable packet processing in the operating

system kernel," in Proc. 14th Int. Conf. Emerg. Netw. EXperiments Technol., New York, NY, USA: Association for Computing Machinery, 2018,

pp. 54–66, doi: 10.11453281411.3281443.

[27] J. Evans, "Linux tracing systems & how they fit together,"

Tech. Rep., 2017. [Online]. Available: https://jvns.ca/blog/2017/

07/05/linux-tracing-systems/

[28] Kernel Development Community. Accessed: Oct. 23, 2021. [Online].

Available: httpswww.kernel.org

[29] A. Deepak, R. Huang, and P. Mehra, "eBPFXDP based firewall and

packet filtering," in Proc. Linux Plumbers Conf., 2018, pp. 1–5.

[30] J. Hong, S. Jeong, J.-H. Yoo, and J. W.-K. Hong, "Design and implementation of eBPF-based virtual tap for inter-VM traffic monitoring," in

Proc. 14th Int. Conf. Netw. Service Manage. (CNSM), 2018, pp. 402–407.

[31] Open Vswitch. Accessed: Oct. 25, 2021. [Online]. Available:

httpwww.openvswitch.org

[32] J. Chen, S. S. Banerjee, Z. T. Kalbarczyk, and R. K. Iyer, "Machine

learning for load balancing in the Linux kernel," in Proc. 11th ACM

SIGOPS Asia–Pacific Workshop Syst., Aug. 2020, pp. 67–74.

[33] M. Chiosi et al., "Network functions virtualisation An introduction, benefits, enablers, challenges and call for action," in Proc. SDN OpenFlow

World Congr., vol. 48, 2012, pp. 1–16.

[34] European Telecommunications Standards Institute (ETSI).

Accessed: Nov. 3, 2021. [Online]. Available: httpswww.

etsi.orgtechnologiesnfv

[35] S. Miano, F. Risso, M. V. Bernal, M. Bertrone, and Y. Lu, "A framework

for eBPF-based network functions in an era of microservices," IEEE

Trans. Netw. Service Manage., vol. 18, no. 1, pp. 133–151, Mar. 2021.

[36] Data Plane Development Kit. Accessed: Jul. 10, 2022. [Online].

Available: httpswww.dpdk.org

[37] L. Rizzo, "Netmap a novel framework for fast packet IO," in Proc. 21st

USENIX Secur. Symp. (USENIX Secur.), 2012, pp. 101–112.

[38] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme,

J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, and M. Casado, "The

design and implementation of open vSwitch," in Proc. 12th USENIX

Symp. Netw. Syst. Design Implement. (NSDI), May 2015, pp. 117–130.

[Online]. Available: httpswww.usenix.orgconferencensdi15technicalsessionspresentationpfaff

[39] C. Systems. Segment Routing. Accessed: Mar. 7, 2022. [Online].

Available: httpswww.segment-routing.net

[40] BIBentryALTinterwordspacing M. Xhonneux, F. Duchene, and

O. Bonaventure, "Leveraging ebpf for programmable network functions

with ipv6 segment routing," in Proc. 14th Int. Conf. Emerg. Netw. Exp.

Technol., New York, NY, USA: Association for Computing Machinery,

2018, pp. 67–72, doi: 10.11453281411.3281426.

[41] L. Tan, W. Su, W. Zhang, J. Lv, Z. Zhang, J. Miao, X. Liu,

and N. Li, "In-band network telemetry: A survey," Comput.

Netw., vol. 186, Feb. 2021, Art. no. 107763. [Online]. Available:

httpswww.sciencedirect.comsciencearticlepiiS1389128620313396

[42] R. V. Bhat, J. Haxhibeqiri, I. Moerman, and J. Hoebeke, "Adaptive

transport layer protocols using in-band network telemetry and eBPF," in

Proc. 17th Int. Conf. Wireless Mobile Comput., Netw. Commun. (WiMob),

Oct. 2021, pp. 241–246.[43] X. Dong and Z. Liu, "Multi-dimensional detection of Linux network

congestion based on eBPF," in Proc. 14th Int. Conf. Measuring Technol.

Mechatronics Autom. (ICMTMA), Jan. 2022, pp. 925–930.

[44] M. Abranches, O. Michel, E. Keller, and S. Schmid, "Efficient network monitoring applications in the kernel with eBPF and XDP," in Proc. IEEE Conf. Netw. Function Virtualization Softw. Defined Netw. (NFV-SDN), Nov. 2021, pp. 28–34.

[45] J. Sommers and R. Durairajan, "Elf High-performance in-band network measurement," in Proc. TMA, 2021. [Online]. Available: https://ix.cs.uoregon.edu/~ram/papers/TMA-2021.pdf

[46] Z. Zha, A. Wang, Y. Guo, and S. Chen, "Towards software defined measurement in data centers: A comparative study of designs, implementation, and evaluation," IEEE Trans. Cloud Comput., early access, Jun. 10, 2022, doi: 10.1109/TCC.2022.3181890.

[47] A. Wang, Y. Guo, F. Hao, T. V. Lakshman, and S. Chen, "Umon Flexible and fine grained traffic monitoring in open vswitch," in Proc. 11th ACM Conf. Emerg. Netw. Exp. Technol., 2015, pp. 1–7, doi: 10.11452716281.2836100.

[48] J. Sheth, V. Ramanna, and B. Dezfouli, "FLIP: A framework for leveraging eBPF to augment WiFi access points and investigate network performance," in Proc. 19th ACM Int. Symp. Mobility Manage. Wireless Access, Nov. 2021, pp. 117–125.

[49] S. Baidya, Y. Chen, and M. Levorato, "EBPF-based content and computation-aware communication for real-time edge computing," 2018, arXiv:1805.02797.

[50] A. Kaufmann, S. Peter, N. K. Sharma, T. Anderson, and A. Krishnamurthy, "High performance packet processing with flexnic," in Proc. 21st Int. Conf. Architectural Support Program. Lang. Operating

Syst., New York, NY, USA: Association for Computing Machinery,

2016, pp. 67–81, doi: 10.11452872362.2872367.

[51] P. Enberg, A. Rao, and S. Tarkoma, "Partition-aware packet steering

using XDP and eBPF for improving application-level parallelism," in

Proc. 1st ACM CoNEXT Workshop Emerg. Netw. Comput. Paradigms,

2019, pp. 27–33, doi: 10.11453359993.3366766.

[52] F. Parola, "Prototyping an eBPF-based 5g mobile gateway," M.S. thesis,

Dept. Comput. Eng., Politecnico di Torino, Turin, Italy, 2020.

[53] F. Parola, F. Risso, and S. Miano, "Providing telco-oriented network

services with eBPF: The case for a 5G mobile gateway," in Proc. IEEE

7th Int. Conf. Netw. Softwarization (NetSoft), Jun. 2021, pp. 221–225.

[54] C. Filsfils, N. K. Nainar, C. Pignataro, J. C. Cardona, and P. Francois,

''The segment routing architecture," in Proc. IEEE Global Commun.

Conf. (GLOBECOM), Dec. 2014, pp. 1–6.

[55] M. Xhonneux and O. Bonaventure, "Flexible failure detection and fast

reroute using eBPF and SRV6," in Proc. 14th Int. Conf. Netw. Service

Manage. (CNSM), 2018, pp. 408–413.

[56] A. Bashandy, C. Filsfils, B. Decraene, S. Litkowski, P. Francois,

D. Voyer, F. Clad, and P. Camarillo. (Oct. 2018). Topology Independent Fast Reroute Using Segment Routing. Internet Engineering

Task Force, Internet-Draft Draft-Bashandy-RTGWG-Segment-RoutingTI-LFA-05. [Online]. Available: httpsdatatracker.ietf.orgdochtmldraftbashandy-rtgwg-segment-routing-ti-lfa-0

[57] D. Katz and D. Ward, Bidirectional Forwarding Detection (BFD),

document RFC 5880, Jun. 2010. [Online]. Available: httpsrfceditor.orgrfcrfc5880.txt

[58] M. Jadin, Q. De Coninck, L. Navarre, M. Schapira, and O. Bonaventure,

''Leveraging eBPF to make TCP path-aware," IEEE Trans. Netw. Service

Manage., vol. 19, no. 3, pp. 2827–2838, Sep. 2022.

[59] S. Bubeck and N. Cesa-Bianchi, "Regret analysis of stochastic and nonstochastic multi-armed bandit problems," 2012, arXiv:1204.5721.

[60] S. Troia, M. Mazzara, M. Savi, L. M. M. Zorello, and G. Maier, ''Resilience of delay-sensitive services with transport-layer monitoring in SD-WAN," IEEE Trans. Netw. Service Manage., vol. 19, no. 3, pp. 2652–2663, Sep. 2022.

[61] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow, and G. Parulkar, "Onos towards an open, distributed sdn os," in Proc. 3rd Workshop Hot Topics Softw. Defined Netw., 2014, pp. 1–6, doi: 10.11452620728.2620744.

[62] V. Gowtham, O. Keil, A. Yeole, F. Schreiner, S. Tschoke, and A. Willner, ''Determining edge node real-time capabilities," in Proc. IEEE/ACM 25th Int. Symp. Distrib. Simul. Real Time Appl. (DS-RT), Sep. 2021, pp. 1–9.

[63] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, "The many faces of publishsubscribe," ACM Comput. Surv., vol. 35, no. 2, pp. 114–131, Jun. 2003, doi: 10.1145857076.857078.

[64] M. Tatarski, H. Parzyjegla, P. Danielis, and G. Muhl, "Fast publishsubscribe using Linux eBPF," Tech. Rep., 2022. [Online]. Available: http://dx.doi.org/10.15496/publikation-67449

[65] I.-C. Wang, S. Qi, E. Liri, and K. K. Ramakrishnan, "Towards a proactive lightweight serverless edge cloud for Internet-of-Things applications," in Proc. IEEE Int. Conf. Netw., Archit. Storage (NAS), Oct. 2021, pp. 1–4.

[66] Knative Framework. Accessed: Dec. 5, 2021. [Online]. Available: httpsknative.devdocs

[67] M. B. Yassein, M. Q. Shatnawi, S. Aljwarneh, and R. Al-Hatmi, "Internet of Things: Survey and open issues of MQTT protocol," in Proc. Int. Conf.

Eng. MIS (ICEMIS), May 2017, pp. 1–6.

[68] T. Chen and C. Guestrin, "Xgboost A scalable tree boosting system,"

in Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining,

2016, pp. 785–794, doi: 10.11452939672.2939785.

[69] S. Rivera, V. K. Gurbani, S. Lagraa, A. K. Iannillo, and R. State, "Leveraging eBPF to preserve user privacy for DNS, DoT, and DoH queries,"

in Proc. 15th Int. Conf. Availability, Rel. Secur., Aug. 2020, pp. 1–10.

[70] R. Aich, "Efficient audit data collection for Linux," M.S. thesis, Dept.

Comput. Sci., Stony Brook Univ., New York, NY, USA, 2021.

[71] S. Y. Lim, B. Stelea, X. Han, and T. Pasquier, "Secure namespaced kernel

audit for containers," in Proc. ACM Symp. Cloud Comput., Nov. 2021,

pp. 518–532.

[72] BIBentryALTinterwordspacing C. Wright, C. Cowan, S. Smalley,

J. Morris, and G. Kroah-Hartman, "Linux security modules General

security support for the Linux kernel," Linux security modules: General

security support for the Linux kernel," in Proc. Found. Intrusion

Tolerant Syst., [Organically Assured Survivable Inf. Syst.], 2002.

[Online]. Available: httpswww.usenix.orgconference11th-usenixsecurity-symposiumlinux-security-modules-general-security-supportlinux

[73] G. Fournier, S. Afchain, and S. Baubeau, "Runtime security monitoring

with eBPF," in Proc. 17th Symp. la Sécurité des Technol. del'Inf. et de la

Commun. (SSTIC), 2021, pp. 1–23.

[74] A. B. Somayaji, "Operating system stability and security through process

homeostasis," Univ. New Mexico, Albuquerque, NM, USA, 2002.

[75] Cisco. Accessed: Dec. 10, 2021. [Online]. Available: httpswww.snort.org

[76] A. V. Aho and M. J. Corasick, "Efficient string matching An aid to bibliographic search,"

Commun. ACM, vol. 18, no. 6, pp. 333–340, Jun. 1975,

doi: 10.1145360825.360855.

[77] Istio. The Istio Service Mesh. Accessed: Dec. 15, 2021.

[Online]. Available: httpsistio.iolatestaboutservice-mesh

[78] J. Levin and T. A. Benson, "ViperProbe: Rethinking microservice observability with eBPF," in Proc. IEEE 9th Int. Conf. Cloud Netw. (CloudNet),

Nov. 2020, pp. 1–8.

[79] L. Deri, S. Sabella, and S. Mainardi, "Combining system visibility and

security using eBPF," in Proc. 3rd Italian Conf. Cyber Secur., in CEUR

Workshop Proceedings, vol. 2315, P. Degano and R. Zunino, Eds. Pisa,

Italy: CEUR-WS.org, Feb. 2019.

[80] M. B'elair, S. Laniepce, and J.-M. Menaud, SNAPPY Program. KernelLevel Policies for Containers.hskip 1em plus 0.5em minus 0.4emrelax New York, NY, USA Association for Computing Machinery, 2021,

pp. 1636–1645, doi: 10.11453412841.3442037.

[81] W. Findlay, D. Barrera, and A. Somayaji, "BPFContain: Fixing the soft

underbelly of container security," 2021, arXiv:2102.06972.

[82] W. Findlay, A. Somayaji, and D. Barrera, "BPFbox Simple precise process confinement with eBPF," in Proc. ACM SIGSAC Conf. Cloud Comput. Secur. Workshop, 2020, pp. 91–103, doi: 10.11453411495.3421358.

[83] C. Cowan, S. Beattie, G. Kroah-Hartman, C. Pu, P. Wagle, and V. Gligor,

''SubDomain parsimonious server security," in Proc. 14th Syst.

Admin. Conf. (LISA 2000). Dec. 2000, pp. 1–20. [Online]. Available:

httpswww.usenix.orgconferencelisa-2000subdomain-parsimoniousserver-security

[84] D. Papamartzivanos, S. A. Menesidou, P. Gouvas, and T. Giannetsos,

''Towards efficient control-flow attestation with software-assisted multilevel execution tracing," in Proc. IEEE Int. Medit. Conf. Commun. Netw.

(MeditCom), Sep. 2021, pp. 512–518.

[85] Intel Platform Analysis Technology. Accessed: Jan. 5, 2022. [Online].

Available: httpswww.intel.comcontentwwwusendevelopertopictechnologyplatform-analysis-technologyoverview.html

[86] A. Chen, A. Sriraman, T. Vaidya, Y. Zhang, A. Haeberlen, B. T. Loo,

L. T. X. Phan, M. Sherr, C. Shields, and W. Zhou, "Dispersing asymmetric ddos attacks with splitstack," in Proc. 15th ACM Workshop Hot Topics

Netw., 2016, pp. 197–203, doi: 10.11453005745.3005773.[87] A. Praseed and P. S. Thilagam, "Modelling behavioural dynamics for

asymmetric application layer DDoS detection," IEEE Trans. Inf. Forensics Security, vol. 16, pp. 617–626, 2021.

[88] H. M. Demoulin, I. Pedisich, N. Vasilakis, V. Liu, B. T. Loo, and

L. T. X. Phan, "Detecting asymmetric application-layer Denial-ofService attacks in-flight with FineLame," in Proc. USENIX Annu. Tech.

Conf. (USENIX ATC), Jul. 2019, pp. 693–708. [Online]. Available: httpswww.usenix.orgconferenceatc19presentationdemoulin

[89] H. V. Wieren. (Nov. 2019). Signature-Based DDoS Attack Mitigation

Automated Generating Rules for Extended Berkeley Packet Filter and

Express Data Path. [Online]. Available: httpessay.utwente.nl80125

[90] A. Carrega, L. Caviglione, M. Repetto, and M. Zuppelli, "Programmable

data gathering for detecting stegomalware," in Proc. 6th IEEE Conf.

Netw. Softwarization (NetSoft), Jun. 2020, pp. 422–429.

[91] SAMSUNG. Accessed: Jan. 18, 2022. [Online]. Available:

httpswww.samsung.comsemiconductorssdpm1743

[92] C. Lukken, G. Frascaria, and A. Trivedi, "ZCSD: A computational storage device over zoned namespaces (ZNS) SSDs," 2021,

arXiv:2112.00142.

[93] J. Gu, Y. Lee, Y. Zhang, M. Chowdhury, and K. G. Shin, "Efficient

memory disaggregation with Infiniswap," in Proc. 14th USENIX

Symp. Netw. Syst. Design Implement. (NSDI). Boston, MA, USA:

USENIX Association, Mar. 2017, pp. 649–667. [Online]. Available:

https://www.usenix.org/conference/nsdi17/technicalsessions/presentation/gu

[94] Q. Xu, M. D. Wong, T. Wagle, S. Narayana, and A. Sivaraman, "Synthesizing safe and efficient kernel extensions for packet processing," in

Proc. ACM SIGCOMM Conf., Aug. 2021, pp. 50–64.

[95] K. Zandberg and E. Baccelli, "Minimal virtual machines on IoT microcontrollers: The case of Berkeley packet filters with RBPF," in Proc. 9th

IFIP Int. Conf. Perform. Eval. Modeling Wireless Netw. (PEMWN). IEEE,

2020, pp. 1–6.

[96] E. Ronen and A. Shamir, "Extended functionality attacks on IoT devices:

The case of smart lights," in Proc. IEEE Eur. Symp. Secur. Privacy

(EuroS&P), Mar. 2016, pp. 3–12.

[97] M. Salehi and K. Pattabiraman, "Poster AutoPatch: Automatic hotpatching of real-time embedded devices," in Proc. ACM SIGSAC Conf. Comput. Commun. Secur., Nov. 2022, pp. 3451–3453. [Online]. Available:

httpswww.usenix.orgconferenceusenixsecurity22presentationhe-yi

[98] H. Zhou, S. Wu, X. Luo, T. Wang, Y. Zhou, C. Zhang, and H. Cai,

''Ncscope Hardware-assisted analyzer for native code in Android apps,"

in Proc. 31st ACM SIGSOFT Int. Symp. Softw. Test. Anal., 2022,

pp. 629–641, doi: 10.11453533767.3534410.

[99] Embedded Trace Macrocell Architecture Specification.

Accessed: Jan. 28, 2022. [Online]. Available: httpsdeveloper.arm.comdocumentationihi0014q

[100] L. Nelson, J. Van Geffen, E. Torlak, and X. Wang, "Specification and

verification in the field Applying formal methods to BPF just-in-time

compilers in the Linux kernel," in Proc. 14th USENIX Symp. Operating

Syst. Design Implement. (OSDI), 2020, pp. 41–61.

[101] H. Vishwanathan, M. Shachnai, S. Narayana, and S. Nagarakatte,

''Semantics, verification, and efficient implementations for tristate numbers," 2021, arXiv:2105.05398.

[102] K. Zandberg and E. Baccelli, "Femto-containers Devops on microcontrollers with lightweight virtualization & isolation for iot software modules," 2021, arXiv:2106.12553.

[103] V. Palmiotti. (Mar. 2022). Kernel Pwning With eBPF a Love Story.

[Online]. Available: httpswww.graplsecurity.compostkernel-pwningwith-ebpf-a-love-story#toc-4

[104] Common Vulnerabilities and Exposures. Accessed: Feb. 6, 2022.

[Online]. Available: httpscve.mitre.orgcgi-bincvekey.cgikeyword=eBPF

[105] J. Steadman and S. Scott-Hayward, "DNSxP: Enhancing data exfiltration

protection through data plane programmability," Comput. Netw., vol. 195,

Aug. 2021, Art. no. 108174.

[106] L. Gerhorst, B. Herzog, S. Reif, W. Schröder-Preikschat, and T. Hönig,

''AnyCall: Fast and flexible system-call aggregation," in Proc. 11th Workshop Program. Lang. Operating Syst., Oct. 2021, pp. 1–8.

[107] G. Frascaria, A. Trivedi, and L. Wang, "A case for a programmable edge

storage middleware," 2021, arXiv:2111.14720.[108] C. Lukken and A. Trivedi, "Past, present and future of computational

storage A survey," 2021, arXiv:2112.09691.

[109] T. Anderson, A. Belay, M. Chowdhury, A. Cidon, and I. Zhang,

''Treehouse A case for carbon-aware datacenter software," 2022,

arXiv:2201.02120.