https://doi.org/10.7840/kics.2024.49.10.1386

IoT/엣지 컴퓨팅 환경에서 경량 보안 가상화 플랫폼의 워크로드 성능 및 자원 소모 비교 분석

고 의 진•, 최 원 미•, 양 경 식°, 유 혁°°

Analysis of Workload Performance and Resource Consumption of Lightweight Secure Virtualization Platforms in IoT/Edge Computing

Eu Jin Ko•, Wonmi Choi•, Gyeongsik Yang°, Chuck Yoo°°

요 약

최근 가상머신의 높은 보안성과 컨테이너의 경량성을 결합한 경량 보안 가상화 플랫폼들이 제안되고 있다. 이 기술은 특히 IoT 장치 및 게이트웨이처럼 막대한 양의 데이터를 처리하면서도 제한된 컴퓨팅 자원을 가진 엣지 컴퓨팅 환경에서 보안과 성능을 동시에 추구할 수 있는 중요한 기술로 간주된다. 다양한 플랫폼 기술이 제안되었지만, IoT 장치들과 엣지 컴퓨팅 환경에서 이들 플랫폼을 비교하고 구체적인 워크로드 성능과 CPU 사용을 이해한 연구는 부족하다. 따라서 각 상황과 워크로드에 최적의 성능과 효율적인 자원 소모량을 파악하기 어려운 상황이다. 본 논문은 최신 경량 보안 가상화 플랫폼인 Firecracker, Kata container, Unikraft를 엣지 컴퓨팅 환경에서 비교 분석한다. 특히, 데이터 수집 및 저장에 빈번히 사용되는 Redis 프로그램을 대상으로 각 플랫폼의 초당 요청 처리량과 CPU 소모량을 세부적으로 분석하였다. 분석 결과, 엣지 컴퓨팅에서 송수신하는 메시지 크기에 따라 최적의 플랫폼이 달라지며, 플랫폼 선택에 따라 사용자 응용 프로그램이 활용 가능한 CPU 자원이 상이해짐을 확인하였다. 이 연구는 세 플랫폼에 대한 분석을 넘어서, 실제 엣지 컴퓨팅 환경에서 적합한 경량 보안 가상화 플랫폼을 선정하는 데 필요한 이해와 기준을 제공한다.

키워드 : IoT, 엣지 컴퓨팅, 보안 가상화 플랫폼, 네트워크 성능 분석
Key Words : IoT, Edge Computing, Secure Virtualization Platforms, Network Performance Analysis

ABSTRACT

Recently, lightweight and secure virtualization platforms that combine the high security of virtual machines with the lightweight nature (high performance) of containers have been widely investigated. These platforms are considered crucial for IoT devices and gateways, which create and communicate vast amounts of data with a

※ 이 논문은 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업(NRF-2021R1A6A1A13044830), 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원(NRF-2023R1A2C3004145, RS-2024-00336564), 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 ICT명품인재양성사업(IITP-2024-2020-0-01819)의 지원을 받아 수행된 연구임. 또한 본 연구는 Google Cloud Research Credits program과 LG전자 산학연구, 고려대학교 정보대학 신임교원정착연구비의 지원을 받아 수행한 연구임.
• First Author : Korea University Department of Computer Science and Engineering, ejko@os.korea.ac.kr, 학생회원; ymcui@os.korea.ac.kr
° Corresponding Author : Korea University Department of Computer Science and Engineering, g_yang@korea.ac.kr, 정회원
°° Corresponding Author : Korea University Department of Computer Science and Engineering, hxy@korea.ac.kr, 종신회원
논문번호 : 202405-108-C-RU, Received May 21, 2024; Revised June 14, 2024; Accepted June 21, 2024

limited amount of computing resources. Although various platforms have been proposed, to our knowledge, no studies have compared these platforms in IoT devices and edge computing environments in terms of understanding practical workload performance and CPU usage. Consequently, it is challenging to determine a proper platform to achieve optimal performance and efficient resource consumption. To this end, this study compares and analyzes the latest lightweight secure virtualization platforms—Firecracker, Kata container, and Unikraft—in edge computing environments. Using the Redis program, which is frequently used for data collection and storage, the study details the analysis of each platform's requests per second and CPU usage. The results show that the proper platform varies based on the message size being transmitted in edge computing, and the choice of platform affects the CPU resources available for user applications. We believe this study provides a baseline for selecting the appropriate platform in actual edge computing environments and serves as a basis for improving future lightweight, secure virtualization platforms.

Ⅰ. 서 론

최근 센서 등의 IoT 장치들은 음성, 영상, 온도, 습도, 충돌 등의 다양한 형태의 데이터를 생성하고 있다. Intel 사에 따르면, 2025년에는 컴퓨팅 인프라들이 생성하는 모든 종류의 데이터 중 55.6%에 이르는 막대한 양이 IoT 장치에서 생성될 것으로 전망된다[1]. IoT 장치들이 생성하는 다양한 형태의 데이터들은 엣지 컴퓨팅을 통해 처리된다. 구체적으로, 생성된 데이터들은 장치 근교에 배치되는 IoT 게이트웨이에 전송되어 저장되며 초기 전처리를 거친다[2]. 이후 데이터들은 엣지 클라우드 등 고성능 서버로 전송되어 인공지능 모델 활용, 통계분석, 공장 내 결함 탐지 등 다양한 응용에 활용된다[3-5]. 이때 IoT 장치 근교에 배치 되어있는 IoT 게이트웨이는 일반적으로 제한된 성능의 CPU와 메모리를 탑재한 장비를 활용한다[6].

IoT 게이트웨이는 다수의 IoT 장치들과의 통신과 그에 따라 발생하는 데이터의 관리를 안정적이고 독립적으로 수행하기 위해 웹 서버, 파일 시스템 등을 가상화하여 처리한다. 이때, IoT 게이트웨이는 한정된 컴퓨팅 자원을 지니므로, 가상 머신에 비해 상대적으로 경량화된 가상화 기술인 컨테이너를 일반적으로 사용한다[7]. 그러나, 컨테이너 가상화 기술은 가상 머신 형태의 가상화에 비해 호스트의 운영체제를 여러 컨테이너가 공유하는 구조이기 때문에, 각 컨테이너가 야기할 수 있는 결함이나 보안적 공격에 매우 취약하다는 문제점이 존재한다[8].

상기 보안 위협들에 대한 대안으로, 경량화된 보안 가상화 플랫폼들이 제안되고 있다[9,10]. 보안 가상화 플랫폼은 1) 일반적인 컨테이너 구조에 추가적인 가상화 레이어를 도입하여 보안성을 강화하거나, 2) 운영체제 수준의 고립을 제공하여 상대적으로 안전한 가상 머신을 경량화하는 접근을 취하고 있다[11-14]. 이와 같이 다양한 형태의 경량화된 보안 가상화 플랫폼이 연구되고 있으나, 기존 연구들은 대부분 컴퓨팅 자원이 매우 풍부한 클라우드의 서버를 대상으로 연구되고 있어[15-19], 제한된 컴퓨팅 자원을 지닌 IoT 장치를 대상으로 하는 플랫폼의 특징은 알려져 있지 않다.

또한, 기존 연구들은 단순한 네트워크 트래픽(netperf, iperf 등)에 대한 처리량을 통해 플랫폼을 분석하고 있어, IoT 장치가 실제 수행하는 데이터 집계 및 파일 시스템 저장 등의 네트워크 성능과 CPU 사용량에 대한 복합적인 이해는 부족하다. 요약하면, 제한된 자원을 지닌 IoT 게이트웨이에 보안 가상화 플랫폼이 활용될 때 그 구조적 차이에서 기인하는 실질적인 워크로드 성능과 CPU 사용량에 대한 명확한 분석은 제시된 바 없다.

이에 본 연구는 가장 널리 활용되고 최신의 세가지 보안 가상화 플랫폼인 Firecracker[13], Kata container[12], Unikraft[14]를 대상으로, 네트워크 성능을 비교 분석한다. 특히, 데이터를 수집하고 저장하는 Redis 응용 프로그램을 활용하여 IoT 게이트웨이의 실질적인 처리 성능인 초당 요청 처리량을 측정한다. 구체적인 세부 분석을 위해 우리는 먼저 세 가지 플랫폼의 가상화 구조와 패킷 처리 방식을 상세히 분석한다. 또한, 각 플랫폼을 다각도로 비교 분석하여 엣지 컴퓨팅의 게이트웨이에 적합한 플랫폼을 평가한다.

이를 통해 IoT 게이트웨이의 제한된 자원 환경에서 초당 요청 처리량 및 CPU 사용량 측면에서 상황 별 적합한 각 보안 가상화 플랫폼을 이해하고, 효율적인 보안 가상화 기술을 적용하는 데 필요한 실질적인 기준을 제시하고자 한다. 본 연구의 결과는 범용적인 환경에서의 IoT 게이트웨이의 보안성과 성능을 복합적으로 향상시키는데 필요한 후속 연구를 발굴할 기반이 될 것으

로 기대된다.

Ⅱ. 관련 연구

본 장에서는 관련 연구를 요약하며, 본 연구의 필요성을 설명한다. 표 1은 경량화된 보안 가상화 플랫폼들에 대한 기존 연구들을 요약한다. 구체적으로, 각 연구가 성능을 비교한 환경 (인프라), 분석한 플랫폼의 종류, 그리고 비교를 위해 측정한 벤치마크 (워크로드)와 그 지표를 설명한다.

먼저, 성능 분석의 환경 측면 (표의 Hardware specification)에서는 대부분의 연구가 많은 자원을 보유한 x86_64 아키텍처 기반의 Intel 서버를 대상으로 분석한다. 이에 비해 ARM 기반의 Raspberry Pi 장치에 대한 보안 가상화 플랫폼의 성능 비교 연구는 Tom Goethals[20]에서만 진행되었다. 그러나 해당 연구는 Firecracker와 Unikernel에 한정된 분석만 제공하고 있다. 따라서 본 연구는 Raspberry Pi를 활용하여, 다수의 IoT 게이트웨이가 사용하는 ARM 코어 기반의 한정된 리소스 환경에서 Firecracker, Kata container, unikernel을 복합적으로 비교 분석한다.

둘째, 워크로드 측면 (표의 Analysis target)에서는 기존 연구들이 대부분 iperf3 또는 netperf등을 사용하여 시간당 송수신량(throughput)을 측정한다. 그러나 실제 IoT 게이트웨이에서는 네트워크의 송수신뿐만 아니라 데이터 저장 등 다양한 연산이 수행된다. 즉, 기존 연구들은 엣지 컴퓨팅의 IoT 게이트웨이가 실제로 처리하는 연산의 특징을 명확히 드러내지 못한다. 표에 나타난 Tom Goethals[20]의 연구는 응용 수준의 네트워크 서비스인 REST에서 데이터를 송수신하는 성능을 측정하지만, 이는 여전히 다른 연구의 송수신량과 유사한 특징과 경향을 보인다. 따라서 본 연구는 엣지 컴퓨팅 환경에서 IoT 데이터를 저장하는 데 널리 사용되는 Redis 캐시 데이터베이스 시스템을 활용하여 네트워크를 거쳐 송수신된 IoT 장치의 데이터가 저장되는 데까지의 성능을 비교한다[21].

셋째, 기존 연구들이 분석한 성능 지표를 살펴보면 (표의 Analysis target 및 CPU), 기존 연구들은 대부분 네트워크 관점에서 처리량만을 측정하며, 구체적인 워크로드의 성능, 예를 들어 초당 요청 처리량 (RPS) 등에 대한 분석은 부족하다. 특히, 기존 연구는 모두 CPU 자원 소모량을 분석하지 않는데, IoT 게이트웨이는 한정적인 CPU 자원을 지니므로 네트워크 성능을 달성하기 위해 효율적인 CPU 자원 사용이 필요하다. 그러나, 보안 가상화 플랫폼 구조의 차이로 인해 각 요소에서 CPU 자원 사용량이 상이하고 그에 따라 네트워크 성능에 차이가 발생할 수 있다. 따라서, 본 연구는 기존 연구가 분석하지 않은 CPU 자원 사용의 효율성을 분석한다. 구체적으로, CPU 사용량을 사용자 (user), 커널 레벨 (system), 패킷 처리 루틴 (softirq) 등으로 세분화하여 프로파일링 한다. 이러한 분석 방법은 보안 가상화 플랫폼 영역에서는 시도되지 않았으나, 데이터센터를 위한 가상화 플랫폼에서는 세부 분석을 위해 중요하게 활용되는 방식이다[22]. 따라서, 본 논문은 CPU 사용량의 세분화 프로파일링을 통해 세 플랫폼에 대한 다각도의 분석을 제공한다.

Ⅲ. 보안 가상화 플랫폼 구조 분석

본 장에서는 Firecracker, Kata container, 그리고 Unikraft의 구조를 소개하고 각각의 네트워크 동작 방식을 분석한다. 이 분석은 다음 장에서 수행할 다각도의

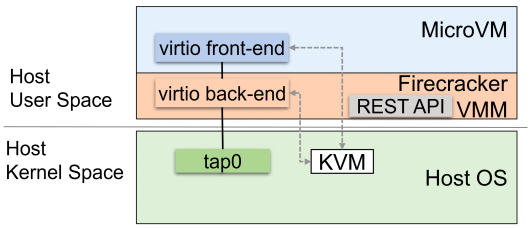
표 1. 관련 논문 비교
Table 1. Related work comparison

Related work	Hardware specification	Platform			Analysis target		CPU
		Firecracker	Kata	Unikernel	Workload	Metric	
Anjali, T. Caraza-Harter, et al. [15]	Intel server	O	X	X	iperf3	Network throughput	X
G. Li, et al. [16]	Intel server	O	O	X	iperf3	Network throughput	X
X. Wang, et al. [17]	Intel server	X	O	X	netperf	Network throughput	X
G. Li, et al. [18]	Intel server	O	O	X	iperf3	Network throughput	X
T. Goethals, et al. [20]	Intel server, Raspberry pi	O	X	O	REST service	Requests per second	X
This study	**Raspberry pi**	**O**	**O**	**O**	**Redis**	**Requests per second**	**O**

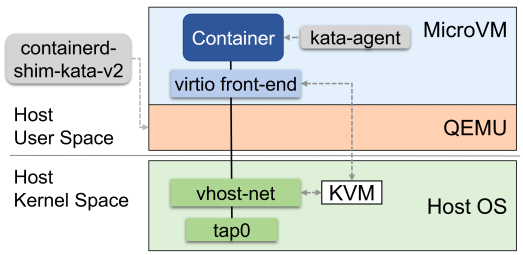
실험 결과를 해석하는 기반이 된다. 각 플랫폼이 가상화하여 생성하는 인스턴스를 경량 가상머신 또는 MicroVM이라고 지칭한다.

3.1 Firecracker

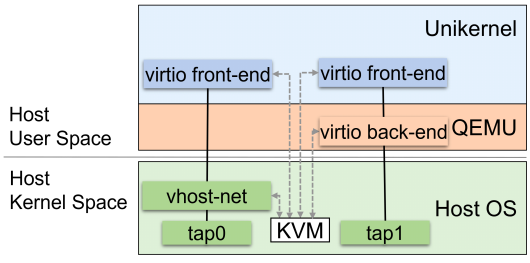
Firecracker[13]는 Amazon에서 제안한 보안 가상화 플랫폼으로, 현재 AWS Lambda에서 사용되고 있다. 그림 1a는 Firecracker의 구조를 보여준다. Firecracker는 운영체제 커널에서 응용 프로그램을 구동하는데 필수적인 기능만 남기고 나머지를 걷어내어 경량 가상머신 (MicroVM)을 생성한다. MicroVM을 실행하기 위해 Firecracker는 사용자 레벨에서 동작하는 Firecracker Virtual Machine Monitor (VMM)를 사용하며, REST API를 통해 MicroVM의 동작을 제어한다. Firecracker VMM은 일반적인 리눅스에 탑재된 가상화 방식인 QEMU[23]와 유사하게 KVM[24]을 이용하여 하드웨어의 동작을 에뮬레이트하지만, 안정성이 높은



(a) Firecracker



(b) Kata container



(c) Unikraft

그림 1. 가상화 플랫폼의 구조 비교
Fig. 1. Comparison of Virtualization Platform Architectures

Rust 언어를 활용하고, 네트워크와 블록 디바이스, 시리얼 포트, 키보드 컨트롤러 등 IO 동작을 위해 최소한의 기능만 제공하여 보안적 위협과 공격이 가능한 영역(attack surface)을 최소화한다.

Firecracker는 네트워크 통신을 위해 virtio 네트워크 드라이버[25]를 사용한다. virtio는 가상화 환경에서 I/O 처리를 위한 split I/O 드라이버 구조로, MicroVM에 위치한 virtio front-end와 Firecracker VMM에 위치한 virtio back-end로 구성되어 있는데, virtio front-end와 virtio back-end는 virtqueue 공유 메모리를 생성하여 패킷을 비동기적으로 전달한다. Firecracker의 구체적인 패킷 처리 과정은 다음과 같다. 1) MicroVM의 애플리케이션이 네트워크 패킷을 전송하면, 이 패킷은 먼저 virtio front-end로 전달된다. 2) virtio front-end는 패킷을 virtqueue에 저장한다. 3) virtqueue에 패킷이 저장되면, Firecracker VMM의 virtio back-end가 이를 감지하고 패킷을 가져온다. 4) Firecracker VMM은 패킷을 물리 네트워크 인터페이스로 전달하여 외부 네트워크와 통신한다.

상기 과정에서 MicroVM 과 KVM, KVM 과 Firecracker VMM, 그리고 Firecracker VMM과 호스트 TUN/TAP 디바이스 사이에서 빈번한 모드 스위칭이 발생한다. 이러한 과정은 패킷 전송 시의 보안 공격에 대응할 수 있는 계층이 늘어나는 이점이 있지만, 호스트 사용자 공간과 커널 공간사이에서의 메모리 복사가 이루어지기 때문에, I/O 요청을 처리하는 데 높은 병목을 야기한다.

3.2 Kata container

Kata container[12,26]는 Alibaba Cloud가 제안한 기법으로 자사 클라우드의 Elastic Container Instance (ECI)를 제공할 때 사용된다. 그림 1b와 같이 Kata container는 컨테이너를 실행할 때 이를 MicroVM 내에서 실행하도록 하여, 기존 컨테이너들이 호스트 자원을 공유함으로써 발생할 수 있는 보안 취약점을 개선한다. 특히 Kata container의 MicroVM은 컨테이너 실행 시 필요한 최소한의 메모리 풋프린트를 달성하기 위해 최적화된 게스트 커널을 활용하여 운영체제를 포함한 가상화에서 발생하는 병목을 개선한다. MicroVM을 실행하기 위해 Kata container는 QEMU 에뮬레이터와 KVM을 활용하며, 컨테이너의 생성과 실행은 각각 containerd-shim-kata-v2 런타임과 MicroVM 내부의 kata-agent를 통해 수행된다. 이러한 구조는 컨테이너의 격리 수준을 높여 보안을 강화하고, 동시에 경량화된 가상화를 통해 성능을 개선한다.

Kata container는 네트워크 통신을 위해 vhost 네트워크 드라이버[24]를 사용한다. 구체적으로 MicroVM내에는 virtio front-end을 사용하며, 호스트 커널 내에서는 vhost-net을 backend로서 활용한다. 해당 구조에서 패킷 처리과정은 다음과 같다. 1) MicroVM 내에서 실행되는 컨테이너화된 응용 프로그램이 패킷을 생성하면, 해당 패킷은 virtio front-end로 전송된다. 2) virtio front-end는 패킷을 virtqueue에 저장한 후 KVM에 트랩을 발생시킨다. 3) Firecracker와 비교하면, KVM은 QEMU를 거치지 않고 대신 패킷 처리 요청을 대기 중인 vhost-net에게 전달한다. 4) 이때, 패킷은 게스트 메모리에서 Direct Memory Access (DMA)를 통해 vhost-net이 접근할 수 있는 호스트 OS의 메모리로 복사된다. 5) 이후, vhost-net은 별도의 스레드를 사용하여 패킷을 읽은 후 MicroVM을 위한 네트워크 인터페이스 (tap0)에 전송한다.

이 과정을 앞서 설명한 Firecracker와 비교해 보면, Kata container는 MicroVM과 KVM 사이에서만 모드 스위칭이 발생하고 별도의 메모리 복사 작업이 발생하지 않기 때문에, Firecracker에 비해 상대적으로 적은 I/O 처리 병목을 야기한다.

3.3 Unikraft

Unikraft[14]는 Xen과 Linux 재단이 주도한 경량 보안 가상화 플랫폼으로, 단일 주소공간을 가진 unikernel을 가상화된 인스턴스로 제공한다 (그림 1c). Unikraft의 가상화된 인스턴스는 응용 프로그램 실행에 필요한 라이브러리와 커널 모듈만 포함하며, 특히 응용 프로그램의 실행에 필요하지 않은 시스템 콜의 호출을 원천적으로 차단하여 보안성을 향상한다. Unikraft는 unikernel을 생성 및 실행하기 위해 QEMU 기반의 KVM 하이퍼바이저를 사용한다.

Unikraft는 네트워크 통신을 제공하기 위해 lwIP 경량 TCP/IP 스택 라이브러리를 활용한다[27]. Unikraft의 호스트 영역에서는 virtio back-end와 vhost-net 드라이버를 모두 활용하는데, Unikraft의 경량화된 패킷 처리를 구현하기 위해 unikernel은 일반적인 리눅스의 net_device API 대신 uknetdev API를 구현하여 사용한다. 구체적인 패킷 처리 과정은 다음과 같다. 1) unikernel 내에서 실행되는 응용 프로그램이 패킷을 생성하면, 해당 패킷은 virtio front-end로 전송된다. 2) virtio front-end는 패킷을 큐 (virtqueue)에 저장하고 KVM에 트랩을 발생시킨다. 3) 이후의 패킷 처리는 Unikraft를 구성하는 두 가지 드라이버, vhost-net과 virtio 드라이버 중 어떤 것이 활용되는지에 따라 달라진다.

vhost-net 드라이버를 사용하는 경우, Kata container와 유사하게 KVM은 vhost-net에게 인터럽트를 통해 패킷 처리 요청을 보낸다. 이후 vhost-net은 별도의 스레드로 패킷을 네트워크 인터페이스 (tap0)에 전송한다. 즉, vhost-net 드라이버가 동작하는 Unikernel은 Unikernel과 KVM 사이 모드 스위칭만 발생하므로, Firecracker에 비해 낮은 병목이 발생한다.

반면 virtio 드라이버를 사용하는 경우, Firecracker와 비슷하게, KVM은 QEMU에게 패킷 처리 요청을 보낸다. QEMU의 virtio back-end는 sys_sendmsg 시스템 콜을 통해 패킷을 네트워크 인터페이스 (tap1)로 전송한다. 이때, Firecracker와 마찬가지로 패킷 전송 시 unikernel과 KVM, KVM 과 QEMU, 그리고 QEMU과 호스트 TUN/TAP 디바이스 사이에서 빈번한 모드 스위칭이 발생한다. 즉, 상대적으로 vhost-net 드라이버를 사용하는 상황과 비교해서는 I/O 요청 시 메모리 복사로 인한 병목이 존재한다는 단점이 있다.

Ⅳ. 보안 가상화 플랫폼에 따른 네트워크 성능 및 CPU 사용량 분석

4.1 실험 환경

본 연구의 실험을 위해 ARM Cortex-A72 1.5GHz CPU와 32GB 메모리를 탑재한 Raspberry Pi 4 Model B를 이용하여 IoT 게이트웨이의 성능과 자원 사용량을 평가한다. 해당 IoT 게이트웨이에 IoT 장치로부터 데이터를 송신하는 과정을 구현하기 위해, Raspberry Pi에 1 GbE로 서버머신을 연결한다. 서버 머신은 x86_64 기반의 CPU (Intel i7-3770K 3.50GHz)를 탑재하고 있다. Raspberry Pi는 우분투 20.04 버전과 리눅스 커널 5.10 버전을 사용하며, 서버 머신은 우분투 18.04 버전과 리눅스 커널 5.4 버전을 사용한다. 상기 구성은 IoT 게이트웨이의 성능과 자원 소모량을 평가하기 위해 다른 연구에서도 활용된다[20].

IoT 게이트웨이인 Raspberry Pi 위에, 보안 가상화 플랫폼을 구동하고 MicroVM을 하나 생성하여 그 내부에 실험 대상 워크로드인 Redis[28] 서버를 실행한다. Redis는 엣지 환경에서 가장 널리 사용되는 데이터베이스 응용 프로그램 중 하나로, IoT 장치에서 전송한 대량의 데이터를 빠르게 저장하고 처리할 수 있기 때문에 netperf나 iperf보다 본 실험의 워크로드로 적합하다[21]. 서버 머신에서는 IoT 게이트웨이로 데이터를 전송하는 redis-benchmark를 실행한다. 해당 벤치마크는 Redis를 향해 데이터 저장 (SET) 또는 전송 (GET) 요청을 생성한다. 실험을 위해 1회 저장 또는 전송 시의 메시지

크기를 32B, 64B, 1024B, 4KB, 16KB로 변화시킨다. 또한, SET과 GET 요청을 각각 100,000번 생성한다. 안정적인 실험값 측정을 위해 각 실험은 최소 3회 이상 수행하고, 평균값을 표시한다.

실험의 비교 군으로 Raspberry Pi 위에 Firecracker (v1.1.1), Kata container (v3.2.0), 그리고 Unikraft (v10)를 구동하여 위 실험을 진행한다. 그중 Unikraft는 네트워크 전송을 위한 드라이버로 vhost와 virtio를 지원하기 때문에, 두 가지 방식을 모두 실험하며 본 논문과 그래프에서는 각각 UK-vhost, UK-virtio로 표기한다. 더불어 경량 가상화 플랫폼과 일반적인 가상화 방식의 성능과 자원 소모량을 분석하고자, 경량 가상화 플랫폼 대신 일반적인 리눅스 위에 QEMU를 기반으로 가상머신을 구동하고 위의 실험을 수행한다. 해당 가상머신은 패킷 처리 시 virtio를 활용한다. 본 방식은 그래프와 논문에서 LinuxVM으로 지칭한다. 각 실험마다 1) 초당 요청 처리량 (requests per second, RPS)과 2) CPU 사용량을 측정한다. RPS는 redis-benchmark[29]로 측정하며, CPU 사용량은 mpstat[30]을 이용하여 측정한다.

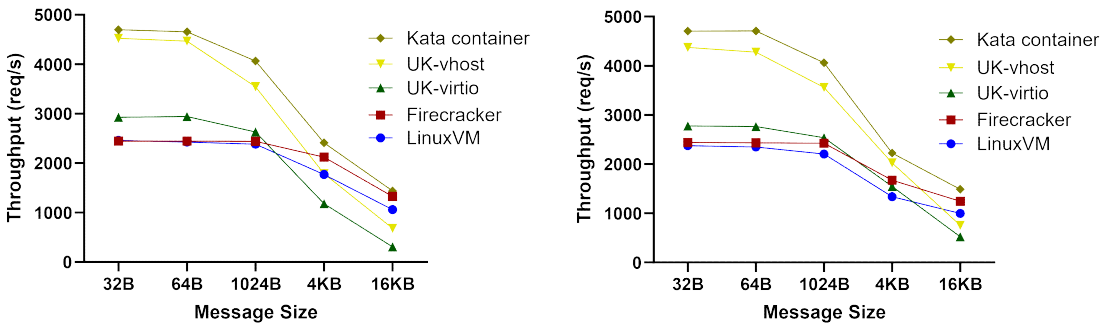
4.2 초당 요청 처리량 (RPS)

그림 2a와 그림 2b는 각각 보안 가상화 플랫폼에 따른 Redis의 GET, SET 요청에 대한 평균 RPS를 나타낸다. 먼저, 실험 결과에서 단일 요청의 크기 (그래프의 x축)가 최대 전송 단위 (MTU, 1500B) 이하인 32B부터 1024B (1KB)까지의 구간에 대해 결과를 분석한다.

결과를 살펴보면 GET과 SET 명령 모두에서 Kata container와 UK-vhost가 상대적으로 높은 RPS를 나타내고, UK-virtio, Firecracker, LinuxVM은 낮은 RPS 값을 나타낸다. 예를 들어, 64B 메시지를 기준으로 살펴보면, Kata container와 UK-vhost는 나머지 세 플랫폼보다 GET과 SET 요청에 각각 평균적으로 75.11%와 78.61% 더 높은 RPS를 나타낸다. 이는 각 플랫폼이 네트워크 패킷 처리를 수행하는 데 사용하는 드라이버에 기인한다. 앞서 III장에서 살펴본 것처럼, 높은 RPS를 보이는 Kata container와 UK-vhost는 패킷 처리에 vhost를 사용한다. 반면, 나머지 세 플랫폼은 virtio를 사용한다. vhost는 패킷 처리 시 네트워크 디바이스에서 호스트를 우회하여 가상화된 게스트 인스턴스로 패킷을 직접 전달하므로, 호스트를 거치는 virtio에 비해 패킷 당 낮은 처리 지연을 보이고, 이로 인해 초당 처리량이 증가한다. 이러한 결과는 플랫폼에 따른 패킷 처리 드라이버의 선택이 RPS에 큰 영향을 미친다는 점을 시사한다.

다음으로, 32B부터 1024B (1KB) 메시지 크기에서 동일한 패킷 처리 드라이버를 사용하는 플랫폼 간의 RPS를 분석한다. vhost를 기반으로 하는 Kata container와 UK-vhost를 살펴보면, Kata container가 UK-vhost보다 GET, SET 요청을 처리하는 데 각각 4.25%, 10.03% 높은 RPS를 보인다. virtio를 사용하는 세 플랫폼을 비교해보면, UK-virtio, Firecracker, LinuxVM 순으로 높은 RPS를 나타내는데, UK-virtio가 나머지 둘보다 GET 요청 시 평균 20.88%, SET 요청 시 평균 15.7% 높은 RPS를 보인다. 이러한 결과를 요약하면, 이전 단락에서 살펴본 것처럼 RPS는 패킷 처리 드라이버의 차이에 따라 큰 차이를 보이며, 동일한 패킷 처리 드라이버를 사용하는 경우라면 각 플랫폼의 구조 및 가상화 방식에 따라 성능 차이가 있음을 알 수 있다.

이번에는 요청 패킷의 크기가 MTU를 넘는 구간인 4KB와 16KB의 결과를 분석한다. 해당 구간을 살펴보면, MTU 이하의 구간과는 다소 상이한 양상을 보이는데, 이를 세부적으로 이해하기 위해 요청의 메시지 크기가 MTU 이하인 64B와 MTU 이상인 16KB에서 보안



(a) GET (b) SET

그림 2. 네트워크 처리량 (GET/SET)
Fig. 2. Network throughput (GET/SET)

가상화 플랫폼내에서 패킷의 크기가 어떻게 변화하는지 확인한다.

표 2는 Redis 요청의 메시지 크기가 64B 와 16KB 일 때 각 플랫폼의 특정 지점에서 측정한 패킷 크기를 나타낸다. GET 명령을 수행할 때, MicroVM으로부터 전달받은 패킷은 1) 호스트 내부의 네트워크 인터페이스 (bridge 또는 tap)를 거쳐, 2) 외부 네트워크로 향하는 NIC 인터페이스 (eth0)를 통해 전송된다. 표 2a와 표 2b는 각각 64B와 16KB 메시지에 대한 GET 명령을 처리할 때 각 지점에서 측정한 패킷 크기를 보여준다. 측정은 vnstat[31]을 활용하여 수행되었다.

표 2a의 64B GET 메시지를 살펴보면, 가상화 플랫폼들 모두 세부 지점에서 평균 127B의 유사한 패킷 크기를 보인다. 반면, 표 2b의 16KB GET 메시지를 살펴보면, Firecracker와 Kata container는 각각 MicroVM으로부터 8263B와 8250B 크기의 패킷을 전달받는다. 이후, 최종 NIC 인터페이스에서는 약 8000B 이상의 큰 패킷이 NIC의 TSO (TCP segmentation offload)와 GRO (Generic receive offload) 등의 기능을 통해 분절 (fragmentation) 되어 1327B의 MTU 이하의 크기로 쪼개져 전송된다. 다시 말해, Firecracker와 Kata container는 메시지 크기가 커지더라도 소프트웨어적으로 처리하는 패킷 수에는 변동이 없으며, 처리에 필요한 지연도 유사하게 발생한다. 따라서, MTU보다 작은 메시지 크기에서 나타나는 상호간의 대소 관계가 MTU가 커지더라도 동일하게 유지된다.

표 2. 가상화 플랫폼의 패킷 크기-GET 요청
Table 2. Packet size of virtualization platforms-GET request

(a) 64B message

	Kata container	UK-vhost	UK-virtio	Firecracker
1) in-host instance (tap / bridge)	123B	124B	124B	137B
2) NIC interface (eth0)	137B	124B	124B	137B

(b) 16KB message

	Kata container	UK-vhost	UK-virtio	Firecracker
1) in-host instance (tap / bridge)	8250B	1224B	1224B	8263B
2) NIC interface (eth0)	1327B	1225B	1225B	1327B

반면 UK-virtio와 UK-vhost는 그림 2a와 그림 2b 모두 급격한 저하를 보인다. 가령, MTU 이하의 메시지에서는 LinuxVM이 가장 낮은 RPS를 보였는데, MTU 이상의 메시지들에서는 오히려 상기 두 플랫폼이 각각 69.2% 그리고 31.68% 낮은 성능을 보인다. 해당 양상을 이해하기 위해, 표 2b의 16KB GET 요청 처리 시 메시지의 크기를 살펴보면, UK-virtio와 UK-vhost 모두 unikernel로부터의 패킷 크기가 1224B로 나타난다. Unikraft를 세부 분석한 결과, 이는 앞서 III장에서 설명한 Unikraft의 lwIP 라이브러리 때문이다. lwIP는 NIC이 패킷을 분절하는 TSO와 GRO 등을 지원하지 않는다. 이로 인해 네트워크 스택이 소프트웨어적으로 패킷 분절을 수행해야 하며, 따라서 Firecracker와 Kata container보다 패킷 당 처리 지연이 증가한다.

상기 분석을 SET 요청에 대해서도 수행했으며, 동일하게 lwIP가 TSO, GRO를 지원하지 않음을 확인할 수 있었다 (세부 결과는 위와 동일하여 생략한다). 요약하자면, 패킷의 크기가 MTU 이상으로 커지면 Unikernel은 NIC의 도움 없이 패킷 분절을 수행해야 하므로 처리 지연이 급격히 증가하며, 이로 인해 RPS 또한 급격히 감소한다.

4.3 CPU 사용량

본 장에서는 IoT 게이트웨이에 실행된 Redis가 소모하는 CPU 사용량을 측정한다. 구체적으로, CPU 사용량을 user, system, softirq, guest로 나누어 측정한다. user는 호스트의 사용자 영역에서 실행되는 Firecracker VMM 또는 QEMU의 CPU 사용량을 나타낸다. system은 호스트의 커널 영역에서 VMM이 호출하는 시스템 콜을 처리하는 데 필요한 CPU 사용량을 나타낸다. softirq는 네트워크 패킷 인터럽트 처리에 필요한 CPU 사용량을 나타낸다. guest는 MicroVM 내 운영체제가 사용하는 CPU 사용량을 나타낸다. 각 플랫폼 간 동일한 양의 CPU를 활용하도록, 모든 실험은 CPU를 1 core씩 활용하도록 설정한다.

그림 3은 메시지 크기 별 GET과 SET 요청을 처리하기 위한 CPU 사용량을 나타낸다. 먼저 user에서의 사용량을 분석한다. user의 CPU 사용량은 GET과 SET 요청들 모두 virtio를 사용하는 UK-virtio, Firecracker, LinuxVM이 vhost를 사용하는 Kata container와 UK-vhost보다 많은 자원을 사용한다. 예를 들어, 64B 메시지를 기준으로 살펴보면, UK-virtio, Firecracker, LinuxVM은 나머지 두 플랫폼보다 GET과 SET 요청에서 각각 평균 7.3%와 7.8% CPU를 더 많이 소모한다.

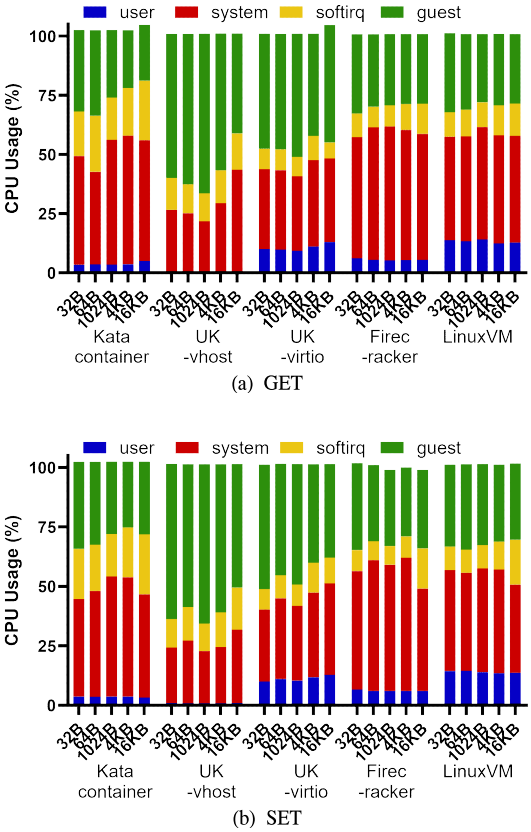


그림. 3. CPU 활용률
Fig. 3. CPU utilization

이는 vhost 기반의 플랫폼이 패킷 처리를 위해 QEMU를 우회하는 반면, virtio 기반의 플랫폼은 그렇지 않기 때문이다. 따라서, 호스트 소모량이 반영되는 user CPU 사용량이 virtio 기반 플랫폼에서 높게 나타난다. 또한, 같은 virtio 기반 플랫폼 중에서 Firecracker는 UK-virtio와 LinuxVM에 비해 GET과 SET 요청 처리 시 user에서 평균 6.5% 낮은 CPU 자원을 소모한다. 이는 경량화된 Firecracker VMM이 QEMU에 비해 패킷 처리에서 더 낮은 오버헤드를 발생시키는 것을 보여준다.

두 번째로 system의 CPU 사용량을 살펴보면 Unikraft 플랫폼인 UK-virtio와 UK-vhost가 나머지 세 플랫폼에 비해 상대적으로 낮은 사용량을 보여준다. 가령, 64B 데이터를 기준으로 볼 때, GET에서 평균 28.8%, SET에서 평균 16.7% 낮은 소모량을 나타낸다. 이는 Unikraft가 unikernel을 기반으로 응용 프로그램을 실행하기 때문이다. unikernel은 다른 보안 가상화 플랫폼과 달리 컴파일 단계에서 가상 메모리 주소와 물리 메모리 주소를 매핑하는 정적 페이지 테이블 할당을 사용한다. 정적 페이지 테이블 할당은 페이지 테이블 교체 작업이 줄어듦으로써 system 영역에서 요구되는 추가적인 메모리 접근이 감소한다. 이러한 특성 덕분에 Unikraft는 system 영역의 CPU 사용량이 상대적으로 낮아진다.

셋째, softirq 사용량을 살펴본다. 플랫폼들을 비교해 보면, virtio 기반의 UK-virtio, Firecracker, LinuxVM이 vhost 기반의 Kata container와 UK-vhost이 비해 적은 CPU를 softirq에 소모한다. 64B 메시지를 기준으로 살펴보면 Kata container와 UK-vhost는 나머지 세 플랫폼보다 GET 과 SET 요청에 각각 평균적으로 4.8%와 7.7% 더 많은 CPU를 소모한다. 이는 vhost 기반 플랫폼이 virtio 기반 플랫폼보다 더 높은 RPS를 처리하기 때문이다. RPS가 높을수록 네트워크 패킷의 송수신이 증가하여, 그에 따라 Kata와 UK-vhost의 softirq 사용량이 높아진다.

마지막으로 guest CPU 사용량을 살펴보면, Unikraft 플랫폼인 UK-virtio와 UK-vhost가 나머지 세 플랫폼에 비해 상대적으로 많은 CPU 사용량을 보여준다. 예를 들어, 64B 메시지를 기준으로 보면 GET과 SET 요청에서 각각 평균 23.5%와 19.2% 더 많은 CPU를 소모한다. 이는 모든 플랫폼이 하나의 CPU 코어를 사용하는 상황에서, UK-virtio와 UK-vhost가 system CPU 사용량을 타 플랫폼들에 비해 적게 사용하고, 이로 인해 확보된 CPU 자원이 MicroVM의 guest 응용 프로그램에 의해 활용되기 때문이다.

4.4 요약

지금까지의 분석을 요약하면 다음과 같다.

먼저, RPS 측면에서 패킷을 구성하는 최대 크기인 MTU 기준으로 성능이 구분된다. MTU (1500 byte)보다 작은 메시지를 빈번히 송수신하는 경우, Kata container와 UK-vhost가 나머지 세 가지 플랫폼에 비해 높은 성능을 달성한다.

반면, MTU보다 큰 메시지가 빈번한 상황에서는 Unikraft가 다른 플랫폼에 비해 패킷 분절(fragmentation)에 대한 오프로드 기능을 활용할 수 없어, 상대적으로 급격한 RPS 저하를 보인다. 따라서, IoT 게이트웨이가 빈번히 처리하는 패킷 크기를 고려하여 높은 성능을 달성할 수 있는 경량 가상화 플랫폼인 Kata container를 선택할 수 있다.

다음으로, CPU 소모량을 살펴보면, 패킷 처리 과정이 상대적으로 짧은 vhost 기반의 플랫폼 (Kata container, UK-vhost)은 virtio 기반의 플랫폼 (UK-virtio, Firecracker, LinuxVM)보다 플랫폼 내 연산에 포함된 user의 소모량은 적은 반면, 호스트 커널 내 패킷 송수

신 처리에 포함된 softirq는 많다. 따라서, 송수신할 패킷의 집중적으로 (burst) 생성되는 환경에서는 vhost 기반의 플랫폼을 선택하는 것이 효율적이다.

반면, MicroVM 내 사용자 응용 프로그램의 경우, Unikraft가 다른 플랫폼에 비해 더 많은 CPU 자원을 확보할 수 있다. 즉, MicroVM 내 응용 프로그램의 연산이 많이 필요하는 경우라면, Unikraft를 선택하는 것이 적절하다.

V. 결 론

본 연구는 IoT 엣지 컴퓨팅 환경에서 경량 보안 가상화 플랫폼의 네트워크 성능과 CPU 자원 효율성을 평가하였다. Firecracker, Kata container, Unikraft 플랫폼과 Redis 워크로드를 대상으로 한 분석 결과, MTU 크기를 기반으로 메시지 크기의 빈도에 따라 적절한 플랫폼을 선택할 수 있음을 확인하였다. 또한, 패킷 처리 연산이 중요한 상황에서 적절한 플랫폼과 MicroVM 내 응용 프로그램에 충분한 CPU 자원을 제공해야 할 때 적합한 플랫폼도 구체적으로 발견하였다.

본 연구의 실험과 분석을 기반으로, 향후에는 영상 등의 큰 IoT 데이터를 게이트웨이가 처리할 때, 파일 시스템의 입출력 병목이 큰 워크로드들의 네트워크 스택과 파일 시스템 스택에 미치는 영향을 다각도로 분석할 예정이다. 또한, 특정 상황이 아닌 대부분의 상황에서 최적의 성능과 자원 효율성을 보이는 새로운 경량 보안 가상화 플랫폼을 연구할 계획이다.

References

[1] *Unleash IoT with Intelligent Edge Devices*, Retrieved Apr. 26, 2024, from https://www.intel.com/content/www/us/en/edge-computing/edge-devices.html.

[2] Y. Yoo, Z. Niu, C. Yoo, P. Cheng, and Y. Xiong, "SegaNet: An advanced IoT cloud gateway for performant and priority-oriented message delivery," in *Proc. 7th Asia-Pacific Wkshp. Netw.*, pp. 54-60, 2023. (https://doi.org/10.1145/3600061.3600072)

[3] Y. Yoo, G. Yang, C. Shin, J. Lee, and C. Yoo, "Machine learning-based prediction models for control traffic in SDN systems," *IEEE Trans. Serv. Comput.*, vol. 16, no. 6, pp. 4389-4403, Nov.-Dec. 2023. (https://doi.org/10.1109/TSC.2023.3324007)

[4] J. H. Cha, J. M. Lee, and D. S. Kim, "Data distribution service based on edge computing platform of industrial IoT," in *Proc. Symp. KICS*, pp. 1164-1165, 2020.

[5] S. O. Choi, Y. J. Lee, J. H. Choi, and J. B. Kim, "Design and implementation of IoT training system based on open source hardware," *J. KICS*, vol. 45, no. 9, pp. 1651-1658, 2020. (https://doi.org/10.7840/kics.2020.45.9.1651)

[6] C. H. Hong and B. Varghese, "Resource management in fog/edge computing: A survey on architectures, infrastructure, and algorithms," *ACM Comput. Surv. (CSUR)*, vol. 52, no. 5, pp. 1-37, 2019. (https://doi.org/10.1145/3326066)

[7] C. Pahl, S. Helmer, L. Miori, J. Sanin, and B. Lee, "A container-based edge cloud PaaS architecture based on raspberry pi clusters," *2016 IEEE 4th Int. Conf. Future Internet of Things and Cloud Wkshps. (FiCloudW)*, pp. 117-124, 2016. (https://doi.org/10.1109/W-FiCloud.2016.36)

[8] F. Sierra-Arriaga, R. Branco, and B. Lee, "Security issues and challenges for virtualization technologies," *ACM Comput. Surv. (CSUR)*, vol. 53, no. 2, pp. 1-37, 2020. (https://doi.org/10.1145/3382190)

[9] J. Sakhdari, B. Zolfaghari, S. Izadpanah, et al., "Edge computing: A systematic mapping study," *Concurrency and Comput.: Practice and Experience*, vol 35, no. 22, pp. 1-32, 2023. (https://doi.org/10.1002/cpe.7741)

[10] Z. Tao, Q. Xia, Z. Hao, et al., "A survey of virtual machine management in edge computing," in *Proc. IEEE*, vol. 107, no. 8, pp. 1482-1499, 2019. (https://doi.org/10.1109/JPROC.2019.2927919)

[11] S. Arnautov, B. Trach, F. Gregor, et al., "SCONE: Secure linux containers with intel SGX," in *12th USENIX Symp. Operating Syst. Design and Implementation (OSDI 16)*, pp. 689-703, 2016.

[12] A. Randazzo and I. Tinnirello, "Kata

containers: An emerging architecture for enabling MEC services in fast and secure way," in *2019 Sixth Int. Conf. IOTSMS*, pp. 209-214, 2019. (https://doi.org/10.1109/IOTSMS48152.2019.8939164)

[13] A. Agache, M. Brooker, A. Florescu, et al., "Firecracker: Lightweight virtualization for serverless applications," in *17th USENIX Symp. NSDI 20*, pp. 419-434, 2020.

[14] S. Kuenzer, V. Bădoiu, H. Lefeuvre, et al., "Unikraft: Fast, specialized unikernels the easy way," in *Proc. Sixteenth Eur. Conf. Comput. Syst.*, pp. 376-394, 2021. (https://doi.org/10.1145/3447786.3456248)

[15] Anjali, T. Caraza-Harter, and M. M. Swift, "Blending containers and virtual machines: A study of firecracker and gVisor," in *Proc. 16th ACM SIGPLAN/SIGOPS Int. Conf. Virtual Execution Environ.*, pp. 101-113, 2020. (https://doi.org/10.1145/3381052.3381315)

[16] G. Li, K. Takahashi, K. Ichikawa, H. Iida, P. Thiengburanathum, and P. Phannachitta, "Comparative performance study of light-weight hypervisors used in container environment," *CLOSER*, pp. 215-223, 2021. (https://doi.org/10.5220/0010440502150223)

[17] X. Wang, J. Du, and H. Liu, "Performance and isolation analysis of RunC, gVisor and Kata Containers runtimes," *Cluster Comput.*, vol. 25, no. 2, pp. 1497-1513, 2022. (https://doi.org/10.1007/s10586-021-03517-8)

[18] G. Li, K. Takahashi, K. Ichikawa, et al., "The convergence of container and traditional virtualization: Strengths and limitations," *SN Comput. Sci.*, vol. 4, no. 387, 2023. (https://doi.org/10.1007/s42979-023-01827-9)

[19] E. Ko, W. Choi, Y. Yoo, and C. Yoo, "A study on the light-weighted virtual machine," in *Proc. Symp. KICS*, pp. 494-495, 2022.

[20] T. Goethals, M. Sebrechts, M. Al-Naday, B. Volckaert, and F. De Turck, "A functional and performance benchmark of lightweight virtualization platforms for edge computing," in *2022 IEEE Int. Conf. Edge Comput. and Commun. (EDGE)*, pp. 60-68, 2022. (https://doi.org/10.1109/EDGE55608.2022.00020)

[21] C. Fanchi, *Redis: What it is, what it does, and why you should care* (2022), Retrieved May 05, 2024, from https://backendless.com/redis-what-it-is-what-it-does-and-why-you-should-care/.

[22] F. Parola, R. Procopio, R. Querio, and F. Risso, "Comparing user space and In-Kernel packet processing for edge data centers," *SIGCOMM Comput. Commun. Rev.*, vol. 53, no. 1, pp. 14-29, 2023. (https://doi.org/10.1145/3594255.3594257)

[23] F. Bellard, "Qemu a fast and portable dynamic translator," *USENIX Annual Technical Conf. FREENIX Track*, pp. 41-46, 2005.

[24] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori, "kvm: The linux virtual machine monitor," in *Proc. Linux Symp.*, pp. 225-230, 2007.

[25] E. P. Martín, *Deep dive into Virtio-networking and vhost-net* (2019), Retrieved Apr. 20, 2024, from https://www.redhat.com/en/blog/deep-dive-virtio-networking-and-vhost-net.

[26] *Kata Containers Architecture* (2024), Retrieved May 01, 2024, from https://github.com/kata-containers/kata-containers/blob/main/docs/design/architecture/README.md.

[27] A. Dunkels, "Design and implementation of the lwIP TCP/IP stack," *Swedish Inst. of Comput. Sci.*, vol. 2, no. 77, 2001.

[28] J. Carlson, *Redis in action*, Simon and Schuster, 2013.

[29] *Redis - the real-time data platform*, Retrieved May 05 2024, from https://redis.io.

[30] *Mpstat*, Retrieved May 05 2024, from https://linux.die.net/man/1/mpstat.

[31] *Vnstat*, Retrieved May 05 2024, from https://humdi.net/vnstat.

고 의 진 (Eu Jin Ko)

2021년 5월 : The University of Arizona, Tucson Computer Science 졸업
2022년 2월~현재 : 고려대학교 컴퓨터공학과 석사 과정
<관심분야> 컨테이너 가상화, 클라우드 오케스트레이션, 컨테이너 보안
[ORCID:0009-0001-9911-1194]

최 원 미 (Wonmi Choi)

2021년 2월 : 고려대학교 컴퓨터학과 졸업
2021년 3월~현재 : 고려대학교 컴퓨터학과 석박통합 과정
<관심분야> 컨테이너 가상화, 컨테이너 오케스트레이션, 커널 네트워킹 스택, 연합 학습
[ORCID:0009-0001-3290-8262]

양 경 식 (Gyeongsik Yang)

2015년 2월 : 고려대학교 컴퓨터학과 졸업
2017년 2월 : 고려대학교 컴퓨터학과 석사
2019년 8월 : 고려대학교 컴퓨터학과 박사
2023년 9월~현재 : 고려대학교 컴퓨터학과 조교수
<관심분야> 시스템 소프트웨어, 네트워크 시스템, AI 시스템, 데이터센터 인프라
[ORCID:0000-0003-4560-2972]

유 혁 (Chuck Yoo)

1982년 2월 : 서울대학교 전자공학과 졸업
1986년 8월 : University of Michigan At Ann arbor 석사
1990년 8월 : University of Michigan At Ann arbor 박사
1990년 8월~1995년 2월 : Sun Microsystems Lab 연구원
1995년 3월~현재 : 고려대학교 정보대학 교수
<관심분야> 운영체제, 소프트웨어정의네트워크, 디지털 헬스
[ORCID:0000-0002-1115-1862]