Journal of KIIT. Vol. 23, No. 5, pp. 117-128, May 31, 2025. pISSN 1598-8619, eISSN 2093-7571
http://dx.doi.org/10.14801/jkiit.2025.23.5.117

쿠버네티스 네트워킹을 위한 컨테이너 네트워크 인터페이스 성능 분석

백승훈*, 구세완**, 윤동원***

Performance Analysis of Container Network Interface for Kubernetes Networking

Seunghoon Paek*, Sewan Gu**, and Dongweon Yoon***

요 약

본 논문에서는 다양한 쿠버네티스 네트워킹을 위한 컨테이너 네트워크 인터페이스(CNI, Container Network Interface)들에 대하여 프로토콜별 성능을 정량적으로 비교 분석한다. 이를 위하여, Calico, Flannel, Cilium, Antrea 4종의 CNI에 대하여 터널링 방식, 암호화 통신 및 네트워크 트래픽 관리 방식을 적용하여 14가지 조합으로 테스트 베드를 설계하고 구현한 후 각 인터페이스 별 성능을 분석한다. 성능 분석을 위하여 TCP 및 UDP 처리량, TCP 및 UDP 지연시간, HTTP 지연시간 및 데이터 전송량, 타임아웃 발생 횟수 지표를 측정하여 CNI에 따른 쿠버네티스 네트워킹 성능 분석을 수행한다. 본 논문에서 제안하는 CNI 별 성능 분석 결과는 다양한 네트워크 제약 사항과 서비스 요구 사항에 적합하도록 선별하여 활용할 수 있을 것이다.

Abstract

This paper quantitatively compares and analyzes the performance of various Container Network Interfaces (CNIs) for Kubernetes networking based on different protocols. To achieve this, we design and implement a testbed to evaluate four CNIs—Calico, Flannel, Cilium, and Antrea—by applying different tunneling methods, encrypted communication, and network traffic management techniques, resulting in 14 unique combinations. Performance analysis is conducted by measuring key metrics, including TCP and UDP throughput, TCP and UDP latency, HTTP latency, data transfer volume, and network timeout occurrences. Based on these experimental results, We establish criteria for selecting the most suitable CNI, considering workload characteristics and performance requirements.

Keywords
container network interface, kubernetes, tunnel, network performance

* NHN Cloud 컨설턴트
- ORCID:https://orcid.org/0009-0006-4372-1168
** LG전자 책임연구원(공동1저자)
- ORCID: http://orcid.org/0000-0001-9547-7626
*** 한양대학교 융합전자공학과 석학교수(교신저자)
- ORCID :http://orcid.org/0000-0001-9631-3500

· Received: Mar. 06, 2025, Revised: Apr. 20, 2025, Accepted: Apr. 23, 2025
· Corresponding Author: Dongweon Yoon
Dept. of Electronic Engineering, Hanyang University,
222 Wangsimni-ro, Seongdong-gu, Seoul 04763, Korea
Tel.: +82-2-2220-0362, Email: dwyoon@hanyang.ac.kr

Ⅰ. 서 론

쿠버네티스는 컨테이너화된 애플리케이션의 확장, 운영, 배포를 자동화하여 효율적으로 관리할 수 있는 오픈소스 플랫폼으로써 마이크로서비스 아키텍처의 핵심 요소 중 하나로 주목받고 있다. 특히, 기존 가상 머신보다 경량화된 컨테이너를 기반으로 신속하고 유연하게 확장 가능하고 가용성과 내결함성을 갖춘 아키텍처 구성을 가능하게 한다[1][2].

쿠버네티스에서는 클러스터를 구성하는 내부 요소 및 외부 간 통신을 구현하기 위해 컨테이너 네트워크 인터페이스(CNI, Container Network Interace)를 필요로 한다[3]. 현재, 다양한 종류의 컨테이너 네트워크 인터페이스가 존재하는데, 이를 통해 쿠버네티스 네트워킹을 구현하는 방법에는 다음과 같은 것들이 있다. 첫째, 쿠버네티스 노드 간 통신은 IP-in-IP, VXLAN(Virtual Extensible LAN), GENEVE (Generic Network Virtualization Encapsulation) 등 다양한 종류의 터널링 방식을 통해 구현할 수 있고 암호화 환경에서는 Wireguard를 활용하여 VPN (Virtual Private Network) 터널을 통해 구현할 수 있다. 둘째, 쿠버네티스 네트워크 트래픽을 관리하는 방법으로 iptables 방식과 eBPF(Extended Berkeley Packet Filter) 방식이 존재한다[4].

본 논문에서는 쿠버네티스 네트워킹 구현을 위하여 다양한 컨테이너 네트워크 인터페이스들에 대한 프로토콜별 네트워크 처리량, 트래픽 지연시간 등의 성능을 정량적으로 비교 분석하여 특정 워크로드 요구사항에 적합한 CNI를 선택하는 기준을 제안한다. 먼저, CNI에 따른 쿠버네티스 네트워킹 성능을 살펴보기 위하여 쿠버네티스 테스트 베드를 설계하고 구현한 후, Calico, Flannel, Cilium, Antrea 4종의 CNI에 대하여 터널링 방식, 암호화 통신 및 네트워크 트래픽 관리 방식을 적용한다. 각 방식별 네트워킹 성능은 TCP(Transmission Control Protocol) 및 UDP(User Datagram Protocol) 처리량, TCP 및 UDP 지연시간, HTTP(HyperText Transfer Protocol) 지연시간 및 데이터 전송량, 타임아웃 발생 횟수 지표를 측정하여 판단한다. 이러한 지표들을 활용하여 각 컨테이너 네트워크 인터페이스의 성능 특성을 정량적으로 비교 및 분석하여 특정 워크로드와 네트워크 요구사항에 적합한 CNI 선택을 위한 기준 제안을 통하여 효율적인 서비스 구축과 운영에 대한 기본 자료를 제공한다.

논문의 구성은 다음과 같다. 2장에서는 쿠버네티스의 네트워킹 유형과 각 컨테이너 네트워크 인터페이스의 특징을 소개한다. 3장에서는 CNI별 성능 분석을 위한 테스트 베드 설계 및 구현 방법을 제시하고 4장에서는 CNI에 따른 쿠버네티스 네트워킹 성능 테스트 결과를 제시하고 분석한다. 이후 마지막 5장에서는 결론을 맺는다.

Ⅱ. 쿠버네티스 네트워킹

본 장에서는 CNI가 제공하는 쿠버네티스 네트워킹 성능 분석을 위한 이론적 토대를 마련하기 위하여 쿠버네티스 네트워킹 모델과 본 논문에서 다루는 4종의 CNI의 특징을 살펴본다.

2.1 쿠버네티스 네트워킹 모델

쿠버네티스의 네트워킹 모델은 크게 파드와 파드 간 네트워킹 모델, 파드와 서비스 간 네트워킹 모델로 구분할 수 있다[5][6]. 다음 절에서는 쿠버네티스의 네트워킹을 위한 컨테이너 네트워크 인터페이스 실험 환경 구축 및 성능 분석을 위하여 먼저 각 네트워킹 모델에 대하여 살펴본다.

2.1.1 파드와 파드간 네트워킹 모델

컨테이너는 쿠버네티스에서 가장 작은 단위로써 여러 개의 컨테이너가 단일 파드내에서 실행될 수 있다. 이때 컨테이너 간 네트워킹은 로컬 호스트를 통해 단일 파드 내에서 이루어진다[6][7].

파드의 경우 낮은 통신 지연시간을 위해 동일한 워커 노드에 배치할 수 있는 반면 가용성과 내결함성을 위해 서로 다른 워커 노드에도 배치할 수 있다. 동일한 워커 노드에 파드를 배치할 경우 파드와 파드 간 네트워킹은 외부 네트워크를 거치지 않고 노드 내에서 처리된다. 반면 서로 다른 노드에 위치한 파드와 파드 간 통신은 그림 1과 같이 나타낼 수 있다. 이때, 노드 간 통신을 위해 여러 터널링

방식이 존재 하는데, 대표적으로 비암호화 통신에서는 IP-in-IP, VXLAN, GENEVE, 암호화 통신에서는 Wireguard와 같은 VPN을 활용한 터널링 방식을 사용한다.

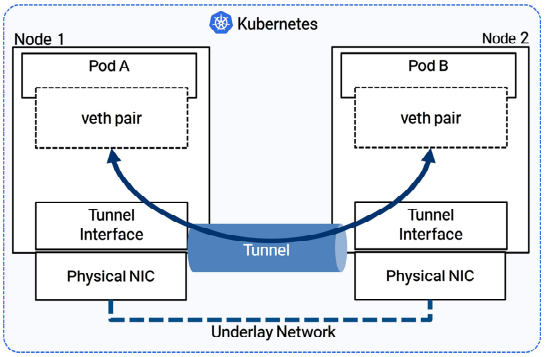


그림 1. 다른 노드에서 실행되는 파드와 파드 간 통신 블록도
Fig. 1. Block diagram of communication between pods running on different nodes

각 터널링 방식은 터널을 구성하기 위해 네트워크 패킷에 헤더가 추가되는데, 각 방식 마다 추가되는 헤더의 크기가 달라 패킷 처리에 소요되는 오버헤드가 상이하다. 또한 노드 간 통신을 위한 터널 인터페이스의 MTU(Maximum Transmission Unit) 역시 달라지기 때문에 패킷당 전송할 수 있는 데이터의 양에도 차이가 발생한다[8][9]. 표 1에는 본 논문에서 다룬 CNI를 대상으로 각 터널링 방식의 헤더 크기, 적용 가능한 CNI와 물리 인터페이스의 MTU가 9000인 경우의 터널 MTU를 나타내었다.

표 1. 터널링 방식 비교
Table 1. Comparison of tunneling methods

No.	Tunneling method	Header size	Supported CNIs	Tunnel MTU
1	IP-in-IP	20 bytes	Calico, flannel	8980
2	VXLAN	50 bytes	Calico, flannel, cilium, antrea	8950
3	GENEVE	least 40 bytes	Antrea, cilium	8950
4	Wireguard	60~80 bytes	Calico, flannel, cilium, anrea	8940~8920 (Varies by CNI)

2.1.2 파드와 서비스간 네트워킹

쿠버네티스의 서비스는 논리적인 집합으로 정의된 파드에 대한 외부 트래픽 접근, 로드 밸런싱 등이 가능하도록 하는 쿠버네티스 네트워킹의 핵심 구성 요소이다. 그림 2에는 파드와 서비스 간 통신 블록도를 나타내었다. 서비스로 트래픽이 라우팅 되면, ClusterIP를 통해 정의된 파드의 집합으로 트래픽이 전달된다.

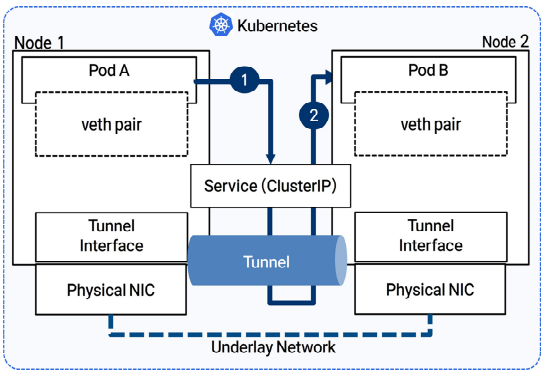


그림 2. 파드와 서비스 간 통신 블록도
Fig. 2. Block diagram of communication between pods and services

외부와 서비스 간 네트워킹은 쿠버네티스에서 호스팅 되는 애플리케이션이나 데이터베이스 등으로 외부로 노출하고 엑세스 할 수 있도록 하는 것을 의미하는데, NodePort와 LoadBalancer 유형의 서비스를 통해 외부에서 쿠버네티스 서비스로 접근 가능하다. 이러한 서비스를 구현하기 위한 쿠버네티스의 트래픽 제어 방식은 대표적으로 iptables와 eBPF 방식이 존재한다. iptables 방식의 경우 모든 쿠버네티스 노드에서 kube-prxoy가 실행되어 서비스 및 파드를 모니터링하고 변경사항을 반영하여 트래픽이 라우팅 되도록 하며, eBPF 방식의 경우 Kernel Space에서 프로그램이 실행되어 직접 트래픽을 필터링하고 라우팅 되도록 한다[10]-[12].

2.2 컨테이너 네트워크 인터페이스

쿠버네티스의 네트워킹을 구현하기 위해서는 다양한 CNI를 사용하는데, 대표적으로 Calico, Flannel,

Cilium, Antrea 등이 있다[13]. 각 CNI는 상이한 네트워크 기능 및 구성 방식을 제공하므로 애플리케이션에서 요구하는 사항에 맞게 선택할 수 있다. 표 2에는 CNI별 특징을 요약하여 설명하였으며, 4가지 CNI별 주요한 특징과 노드 간 네트워킹 방식, 트래픽 관리방식, 노드 간 터널링 방식에 대해 정리하여 나타내었다.

표 2. CNI 별 특징
Table 2. Features by CNI

No.	CNI	Key features	Traffic management method	Tunneling method
1	Calico	-Provides network policies by IP address, port, and namespace	iptables	IP-in-IP
				VXLAN
				Wireguard
			eBPF	VXLAN
				Wireguard
2	Flannel	-Network policy not provided	iptables	IP-in-IP
				VXLAN
				Wireguard
3	Cilium	-Traffic control at layers 3, 4, and 7	eBPF	VXLAN
				GENEVE
				Wireguard
4	Antrea	-OVS based -Provides network policy	iptables	VXLAN
				GENEVE
				Wireguard

III. 컨테이너 네트워크 인터페이스 테스트 베드 설계 및 구현

본 장에서는 컨테이너 네트워크 인터페이스 성능 측정을 위한 주요 지표 및 분석 실험에 사용할 테스트 베드를 설계 및 구현하는 구체적인 방법에 대하여 설명한다.

3.1 컨테이너 네트워크 인터페이스 성능 지표

성능 관점에서 CNI를 선택하는 기준으로 노드 간 터널링 방식과 네트워크 트래픽 관리 방식을 고려한다[14]. 또한 암호화 통신을 위해 각 CNI에 Wireguard를 적용하여 암호화 통신 시 네트워킹 성능도 함께 비교 분석한다.

성능 측정은 파드와 파드 간 네트워킹 모델, 파드와 서비스 간 네트워킹 모델을 적용하여 실험을 수행 하는데, 표 3에는 성능 측정 지표와 측정 도구, 측정 회수를 나타내었다. 성능 측정의 신뢰성을 확보하기 위해 각 성능 분석 지표에 사용해야 하는 측정 도구, 파드 배치 방식, 커스텀 파라미터 설정을 적용한다.

표 3. 컨테이너 네트워크 인터페이스별 성능 분석 지표 및 측정 도구
Table 3. Container network interface-specific performance analysis metrics and measurement tools

No.	Performance metrics	Measurement tool (Version)	Duration per measurement / Number of measurements
1	TCP throughput (Gbits/sec)	iperf3 (v3.7) [15]	Performed for 10 minutes per measurement, repeated 3 times
2	UDP throughput (Gbits/sec)		
3	UDP packet loss (%)		
4	Average TCP latency (usec)	sockperf (v3.10) [16]	
5	Average UDP latency (usec)		
6	Server node CPU usage (%)	top	
7	Server node memory usage (%)	free	
8	Average HTTP latency(ms)	wrk (v4.2.0) [17]	
9	HTTP data transfer (GB)		
10	Number of HTTP timeouts		

다음은 성능을 측정하는 방법에 대해 논하도록 한다. 성능 측정은 클라이언트가 서버로 요청을 보내고 서버에서 응답하는 형태로 진행하는데, 서버 역할을 담당하는 파드의 경우 워커 노드 1에 배치하고 클라이언트 역할을 담당하는 파드는 워커 노드 2에 배치하여 실험을 한다. 실험과 무관한 쿠버네티스 운영하는 파드에 대한 설계와 구현은 별도로 3.2절에서 논하기로 한다. 또한, 표 3에 언급한 각 측정 도구 별 실험에서 사용한 커스텀 파라미터 설정값을 표 4에 나타내었다.

표 4. 성능 측정 도구별 커스텀 파라미터
Table 4. Custom parameters per performance measurement tool

No.	Measurement tool	Parameter
1	iperf3	Threads: 4 threads Message size: 8800 bytes Transmission rate (UDP): 10 Gbps Duration per measurement: 10min Number of measurements: 3 times
2	sockperf	Message size: 8800 bytes Duration per measurement: 10 min Number of measurements: 3 times
3	wrk	Threads: 4 threads Number of concurrent connections: 3000 Duration per measurement: 10 min Number of measurements: 3 times

3.2 성능 분석을 위한 테스트 베드 설계 및 구현

그림 3에는 쿠버네티스 테스트 베드의 물리적 링크 구성도를 나타내었는데, 노드 간 물리적 링크의 경우 마스터 노드의 네트워크 카드(enp1s0f0, enp1s0f1)에 각 워커 노드의 네트워크 카드를 DAC (Direct Attach Copper) 케이블로 직결한 후 워커 노드와 직결된 마스터 노드의 인터페이스를 브릿지 인터페이스로 구성한다. 물리적 링크를 구성한 후 통신이 가능하도록 모든 노드에 IP 패킷 포워딩을 활성화한다.

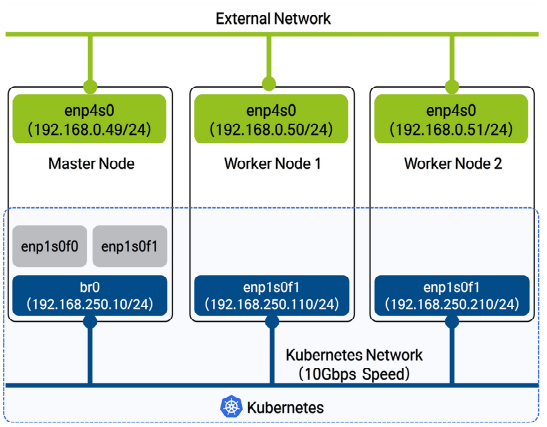


그림 3. 쿠버네티스 테스트 베드 물리적 링크 구성도
Fig. 3. Block diagram of Kubernetes test bed physical link configuration

이렇게 구성하여 노드 간 통신 구성에 사용되는 L3 장비를 배제하여 특정 L3 하드웨어의 성능에 종속되지 않도록 한다. 네트워크 카드의 오프로딩 기능과 커널 옵션은 기본 설정을 유지하여 특정 CNI에 유리하게 작용할 수 있는 상황을 배제한다[18].

쿠버네티스 클러스터를 구성하는 하드웨어는 정량적인 네트워크 성능 측정을 위해 표 5와 같이 동일한 제원의 물리 노드 3대를 활용하여 1대의 마스터 노드와 2대의 워커 노드로 구성된 쿠버네티스 클러스터를 구축한다. 컨테이너 런타임의 경우 쿠버네티스에서 권장하는 Containerd를 사용한다[19].

표 5. 쿠버네티스 각 노드 제원
Table 5. Kubernetes node specifications

No.	Item	Description
1	CPU / Memory / SMT	AMD ryzen 5500GT, 16GB, SMT disabled
2	Operating system	Ubuntu 20.04 LTS
3	Kubernetes version	1.29.9
4	Container runtime	Containerd
5	Network interface card 1	Realtek RTL8111/8168/8411, node access purpose, 1Gbps
6	Network interface card 2	Intel X710 DA4-FH, kubernetes configuration purpose, 10Gbps
7	Interface MTU	9000

테스트 베드를 설계하고 구현하기 위한 쿠버네티스 클러스터 내부 파드 구성은 그림 4와 함께 설명한다.

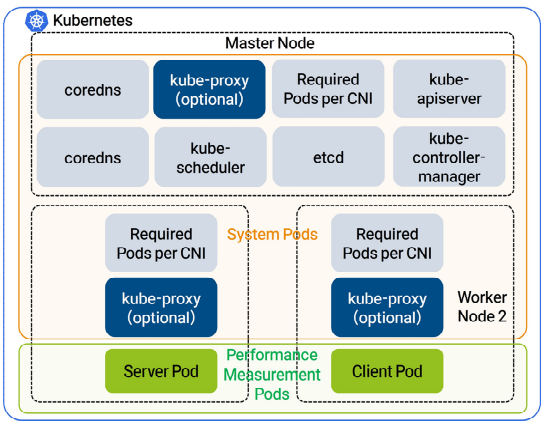


그림 4. 쿠버네티스 테스트 베드 파드 배치 블록도
Fig. 4. Block diagram of Kubernetes testbed pod

마스터 노드에는 기본적인 운영을 위한 coredns, etcd, kube-apiserver, kube-controller-manager, kube-scheduler, kube-proxy 파드를 배포한다.

각 컨테이너 네트워크 인터페이스에 따라 추가적으로 필요한 파드가 존재하는데, 표 6에 정리하여 나타내었다. 예외적으로 트래픽 관리를 위해 eBPF가 활성화된 CNI의 경우 kube-proxy 파드가 필요하지 않다.

표 6. 컨테이너 네트워크 인터페이스별 추가되는 운영용도 파드
Table 6. Additional operational pods per container network interface

No.	CNI	Required pods	Quantity	Placement nodes
1	Calico	calico-kube-controllers	1	Master node
		calico-node	3	1 Pod per node
		calico-typha	1	Master node
2	Flannel	kube-flannel-ds	3	1 Pod per node
3	Cilium	cilium	3	1 Pod per node
		cilium-envoy	3	1 Pod per node
		cilium-operator	1	Master node
4	Antrea	antrea-agent	3	1 Pod per node

한편, 워커 노드에는 쿠버네티스 클러스터 운영에 참여하는 최소한의 파드만을 실행하고, CNI 성능 측정 실험에 참여하는 파드를 배치한다. 이렇게 하여 쿠버네티스 운영에 필요한 파드들과 성능 측정실험에 참여하는 파드들을 서로 다른 노드에 배치하여 쿠버네티스 운영에 참여하는 파드들이 실험 결과에 미치는 영향이 최소가 되도록 구현 한다.

IV. 네트워킹 실험 및 성능 분석

본 장에서는 3장에서 구현한 테스트베드를 기반으로 CNI별 쿠버네티스 네트워킹 성능을 실험하고, 실험에서 측정된 데이터를 비교하고 분석한다. 각 CNI별 특징과 장단점을 살펴보고 이를 통해 특정 워크로드와 네트워크 요구사항에 적합한 CNI 선택을 위한 기준을 제공한다.

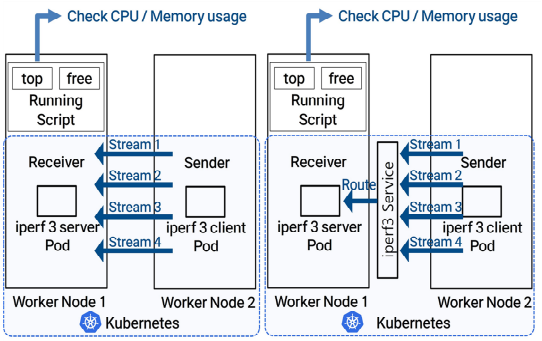
4.1 실험 방법

본 절에서는 CNI별 TCP 및 UDP 처리량, TCP 및 UDP 지연시간, HTTP 트래픽 처리 성능을 측정하기 위한 실험 방법을 논한다.

4.1.1 TCP 및 UDP 처리량

TCP 및 UDP 처리량을 측정하기 위하여 파드와 파드 간 네트워킹 모델과 파드와 서비스 간 네트워킹 모델을 적용하는데, 특정 워커 노드에는 서버 역할의 파드를 배치한다.

그림 5에는 TCP 및 UDP 처리량 측정 블록도를 나타내었다. 그림 5(a)는 파드와 파드 간 네트워킹 모델, 그림 5(b)는 파드와 서비스 간 네트워킹 모델을 나타낸다. 파드와 파드 간 네트워킹 모델에서는 iperf3 클라이언트가 iperf3 서버(Node1)로 4개의 병렬 스트림으로 직접 트래픽을 전송한다. 10분간 전송한 후에 최종적으로 iperf3 서버가 수신한 트래픽 양을 집계하여 처리량을 반환한다. 파드와 서비스 간 네트워킹 모델에서는 iperf3 클라이언트가 iperf3 서비스로 10분간 트래픽을 전송하면, 서비스는 iptables 혹은 eBPF를 통해 iperf3 서버로 트래픽을 라우팅하고, iperf3 서버가 트래픽을 수신하며 최종적으로 수신한 트래픽 양을 집계하여 처리량을 반환한다. 한편, Linux OS 내의 top과 free 명령을 script를 사용하여 활용하는데, 성능 측정 간 iperf3 서버가 위치한 노드(Node 1)의 CPU 및 메모리 사용량을 함께 집계한다.



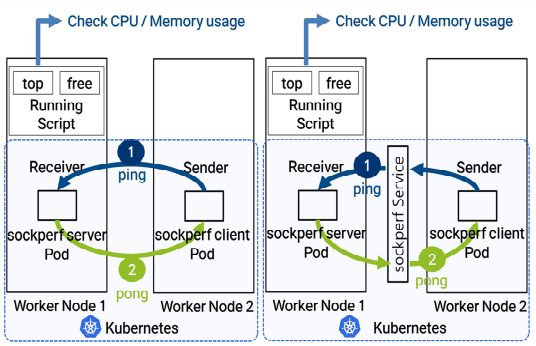
(a) (b)

그림 5. TCP 및 UDP 처리량 측정 블록도
(a) 파드와 파드 간 통신, (b) 파드와 서비스 간 통신
Fig. 5. Block diagram of TCP and UDP throughput measurement
(a) Pod to pod communication, (b) Pod to service communication

4.1.2 TCP 및 UDP 지연시간

TCP 및 UDP 지연시간을 측정하기 위하여 파드와 파드 간 네트워킹 모델과 파드와 서비스 간 네트워킹 모델을 적용하는데, 특정 워커 노드에는 서버 역할을 수행할 파드를 배치한다. 각 CNI가 제공하는 네트워크의 지연시간을 sockperf를 활용하여 측정하고, 이를 통해 각 네트워킹 모델에서의 TCP 및 UDP 지연시간에 대한 각 CNI의 성능 특성을 비교 분석한다.

그림 6에는 TCP 및 UDP 지연시간 측정 블록도를 표현하는데, 그림 6(a)는 파드와 파드 간 네트워킹 모델, 그림 6(b)는 파드와 서비스 간 네트워킹 모델을 나타내었다. 파드와 파드 간 네트워킹 모델에서는 sockperf 클라이언트가 sockperf 서버로 메시지를 보내고 sockperf 서버가 메시지를 수신한 후 sockperf 클라이언트로 응답한다. 이러한 일련의 과정을 10분간 거친 후 지연시간을 반환한다. 파드와 서비스 간 네트워킹 모델에서는 sockperf 클라이언트가 sockperf 서비스로 메시지를 전송하면, sockperf 서비스가 iptables 혹은 eBPF를 통해 sockperf 서버로 트래픽을 라우팅하고, sockperf 서버가 메시지를 수신한 후 동일한 경로를 사용하여 sockperf 클라이언트로 응답한다.



(a) (b)

그림 6. TCP 및 UDP 지연시간 측정 블록도
(a) 파드와 파드 간 통신, (b) 파드와 서비스 간 통신
Fig. 6. Block diagram of TCP and UDP latency measurement
(a) Pod to Pod communication, (b) Pod to Service communication

이러한 실험 과정을 10분간 실행 한 후 지연시간을 반환한다. 한편, Linux OS 내의 top과 free 명령을 script를 사용하여 활용하는데, 성능 측정 간 sockperf 서버가 위치한 노드(Node 1)의 CPU 및 메모리 사용량을 함께 집계한다.

4.1.3 HTTP 트래픽 처리 성능

HTTP 트래픽 처리 성능을 측정하기 위하여 파드와 서비스 간 네트워킹 모델만을 적용하며, 파드와 파드간 네트워킹 모델을 적용하지 않는다. 이유는 파드와 파드 간 네트워킹 모델을 적용하는 경우 HTTP 프로토콜의 특성을 충분히 반영할 수 없기 때문인데, 원인은 다음과 같다. HTTP 워크로드는 Layer 3, Layer 4 계층에서 동작하는 TCP 및 UDP와 다르게 Layer 7 계층에서 동작하므로 경로 기반 라우팅, 세션 처리 등 다양한 추가 작업을 거치게 된다. 쿠버네티스에서는 Ingress를 통해 경로 기반 라우팅 등 Layer 7의 기능을 구현할 수 있으며, 내부적으로 쿠버네티스의 서비스와 연결되기 때문이다[20].

실험은 다음과 같이 수행한다. 특정 워커 노드에 클라이언트 역할을 수행할 wrk 파드를 배치하고, 서버 역할을 수행할 웹서비스인 Nginx 파드를 다른 노드에 배치한다. 고부하 환경에서의 HTTP 지연시간, 데이터 전송량, 타임아웃 발생 횟수를 측정하고, 이를 통해 HTTP 트래픽 처리 성능에 대한 각 CNI의 성능 특성을 비교 분석한다.

그림 7에는 HTTP 트래픽 처리 성능 측정 블록도를 나타내었다. wrk 클라이언트가 4개의 쓰레드를 활용하여 3000개의 연결을 유지하여 Nginx 서비스로 웹 페이지 요청을 보내면, Nginx 서비스가 iptables 혹은 eBPF를 통해 Nginx 서버로 트래픽을 라우팅하고, Nginx 서버는 웹페이지를 동일한 경로로 반환한다. 이러한 프로세스를 10분간 거친 후 지연시간과 웹페이지 데이터 전송량, 타임아웃 발생 횟수를 반환한다. 한편, Linux OS 내의 top과 free 명령을 script를 사용하여 활용하는데, 성능 측정 간 Nginx 서버가 위치한 노드(Node 1)의 CPU 및 메모리 사용량을 함께 집계한다.

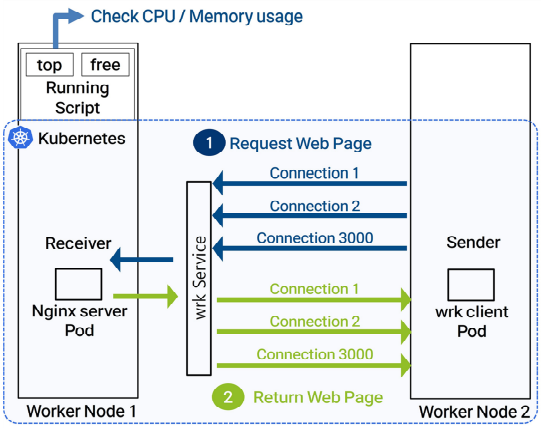


그림 7. HTTP 트래픽 처리 성능 측정 블록도
Fig. 7. Block diagram of HTTP traffic processing performance measurement

4.2 네트워킹 성능 분석

본 연구에서 주요 지표로 설정한 TCP, UDP, HTTP 프로토콜은 실제 네트워크 환경에서 다음의 의미를 갖는다. 먼저 TCP의 경우 데이터베이스 접근, 파일 전송 등과 같은 신뢰성 있는 데이터 통신이 필요한 환경에서 손실 없이 안정적인 전송을 목적으로 하는 프로토콜이다[21]. 높은 처리량과 낮은 지연시간은 얼마나 많은 데이터를 빠르게 전송할 수 있는지의 평가 근거가 되며 최종 사용자의 서비스 경험 품질을 향상 시키는데 기여한다. 다음으로 UDP의 경우 실시간 데이터 전송이 필요한 서비스에서 주로 사용하는 프로토콜이다. 데이터 전송 간 신뢰성이 보장되지 않으므로 높은 처리량은 효율적인 UDP 데이터 전송량을 의미하며 낮은 지연시간은 서비스의 실시간 반영을 의미한다. HTTP의 경우 웹 서비스에서 사용되는 프로토콜로 높은 처리량과 낮은 지연시간은 웹 서비스 환경에서 얼마나 많은 데이터를 제공할 수 있는지에 대한 판단 근거이자 최종 사용자의 체감 성능에 반영되는 지표이다.

표 7에는 실험 결과를 나타낸 것으로 No 1~4는 파드와 파드 간 통신, No 5~9는 파드와 서비스 간 통신 결과를 상세히 나타내었다. 다음에 파드와 파드 간 통신과 파드와 서비스 간 통신에 대한 실험 결과에 대하여 상세히 논한다.

4.2.1 파드와 파드 간 통신

파드와 파드간 통신에서 나온 측정 결과를 크게 4가지 측정 지표인 TCP 처리량, UDP처리량, TCP 평균 지연시간, UDP 평균 지연시간으로 나누어 설명하는데, 비암호화 통신과 암호화 통신 별로 나누어서 논한다.

우선 비암호화 통신의 경우 CNI별 측정 결과는 다음과 같다. 첫째, TCP 처리량과 TCP 처리시에 사용되는 CPU 사용률, 메모리 사용률은 CNI별 유의미한 차이가 없었다. 둘째, UDP 처리시에 패킷 손실률은 Flannel이 다른 CNI에 비해 약 25배에서 70배가량 높게 측정되었고, CPU 사용률은 Cilium과 Antrea에서 다른 CNI 대비 약 1.6배 정도 높게 나타났다. 셋째, TCP 평균 지연시간은 Calico와 Flannel(IPinIP)가 약 10% 적은 지연시간을 보였으나, CPU, 메모리 사용률은 CNI별 큰 차이가 없었다. 넷째, UDP 평균 지연시간은 CNI별로 큰 의미있는 차이가 없었다.

암호화 통신의 경우는 4가지 측정 지표에서 UDP 패킷 손실률을 제외하고, 각 CNI별로 큰 차이가 없었으며, UDP 패킷 손실률에서 Calico-eBPF와 Cilium이 다른 CNI와 비교해서 약 5배 이상 낮은 UDP 패킷 손실률을 보인다.

4.2.2 파드와 서비스 간 통신

파드와 서비스 간 통신의 경우도 마찬가지로 위에서 논의된 4가지 측정 지표와 함께 추가로 HTTP 프로토콜 관련된 측정 지표를 추가하여 설명한다.

우선 비암호화 통신의 경우 CNI별 측정 결과부터 설명하도록 한다. 첫째, TCP 처리량과 TCP 처리시에 사용되는 CPU 사용률, 메모리 사용률은 파드와 파드 간 통신의 결과와 유사하게 CNI별 유의미한 차이가 없었다. 둘째, UDP 처리량도 유의미한 차이가 없었고, UDP 패킷 손실률이 파드와 파드간 통신의 경우와 유사하게 Flannel 이 약 25배에서 70배 높게 나타났다. CPU 사용률도 Cilium과 Antreal에서 다른 CNI에 비해 약 1.6배 높게 측정이 되었다. 셋째, TCP 평균 지연시간은 Calico (Vxlan), Calico-eBPF(Vxlan), Flannel(IPinIP)에서 약 10% 적은 지연시간을 보였다.

표 7. 성능 측정 결과
Table 7. Performance measurement result

No.	Measurement metrics	Non-encrypted communication									Encrypted communication				
		Calico (IPinIP)	Calico (Vxlan)	Calico-eBPF (Vxlan)	Flannel (IPinIP)	Flannel (Vxlan)	Cilium (Vxlan)	Cilium (Geneve)	Antrea (Geneve)	Antrea (Vxlan)	Calico (Wire guard)	Calico-eBPF (Wire guard)	Flannel (Wire guard)	Cilium (Wire guard)	Antrea (Wire guard)
	Pod to Pod (Measured for 10 Minutes per trial, average of 3 trials)														
1	TCP throughput	9.80	9.78	9.84	9.80	9.66	9.84	9.84	9.84	9.84	9.81	9.80	9.81	9.76	9.80
	CPU usage	13.3	14.2	11.9	13.0	11.7	14.0	14.7	13.5	13.4	43.0	42.6	41.5	40.4	43.8
	Memory usage	4.0	4.0	5.0	3.0	3.0	4.0	4.3	4.0	4.0	4.0	5.0	3.0	4.0	
2	UDP throughput	9.88	9.80	9.78	9.90	9.86	9.81	9.82	9.84	9.84	9.28	9.76	9.30	9.00	8.95
	UDP packet loss	0.06	0.05	0.04	2.83	1.27	0.04	0.03	0.04	0.04	1.03	0.28	1.37	0.16	1.4
	CPU usage	13.7	14.4	12.7	12.3	12.4	20.0	20.0	19.0	19.1	39.1	40.6	39.6	40.3	39.1
	Memory usage	4.0	4.0	5.0	3.0	3.0	4.0	4.3	4.0	4.0	4.0	5.0	3.0	4.0	4.0
3	Avg. TCP latency	90.6	90.6	91.0	86.7	100.4	103.1	102.9	103.1	100.5	138.2	136.0	131.4	137.1	136.6
	CPU usage	3.1	3.1	2.8	2.7	2.6	3.1	3.0	3.2	3.3	5.0	5.2	4.9	5.2	5.2
	Memory usage	4.0	4.0	5.0	3.0	3.0	4.0	4.3	4.0	4.0	4.0	5.0	3.7	4.0	4.0
4	Avg. UDP atency	94.1	96.6	89.3	95.2	87.2	89.2	97.3	91.5	95.8	136.4	135.9	129.4	141.8	134.8
	CPU usage	3.2	3.1	3.1	2.7	2.8	3.4	3.3	3.4	3.3	5.2	5.2	4.8	5.6	5.3
	Memory usage	4.0	4.0	5.0	3.0	3.0	4.0	4.3	4.0	4.0	4.0	5.0	3.3	4.0	4.0
	Pod to service (Measured for 10 minutes per trial, average of 3 trials)														
5	TCP throughput	9.81	9.77	9.84	9.80	9.63	9.84	9.84	9.84	9.84	9.81	9.80	9.81	9.76	9.80
	CPU usage	13.3	14.3	11.8	13.1	12.2	14.0	14.1	12.9	13.2	43.0	42.5	41.5	40.1	43.9
	Memory usage	4.0	4.0	5.0	3.0	3.0	4.3	4.0	4.0	4.0	4.0	5.0	3.3	4.0	4.0
6	UDP throughput	9.51	9.79	9.69	9.90	9.86	9.82	9.82	9.84	9.84	9.26	9.76	9.22	9.23	8.77
	UDP packet loss	0.05	0.04	0.04	2.13	1.60	0.03	0.04	0.03	0.04	1.00	0.26	1.27	0.15	1.33
	CPU usage	13.4	14.2	12.5	11.9	12.4	19.9	19.7	19.1	18.9	38.9	40.6	39.5	40.7	38.2
	Memory usage	4.0	4.0	5.0	3.0	3.0	4.3	4.0	4.0	4.0	4.0	5.0	3.3	4.0	4.0
7	Avg. TCP latency	101.1	89.8	86.3	86.2	101.8	101.5	101.5	100.5	101.9	138.4	136.4	132.0	137.1	137.0
	CPU usage	2.9	3.1	2.9	2.7	2.6	3.0	3.1	3.2	3.3	5.0	5.2	4.9	5.2	5.0
	Memory usage	4.0	4.0	5.0	3.0	3.0	4.3	4.0	4.0	4.0	4.0	5.0	3.3	4.0	4.0
8	Avg. UDP latency	98.7	99.5	92.9	93.7	92.8	89.5	92.5	87.4	87.8	136.5	137.5	130.1	142.2	135.5
	CPU usage	3.1	3.1	3.1	2.8	2.7	3.5	3.4	3.4	3.5	5.1	5.0	4.7	5.5	5.3
	Memory usage	4.0	4.0	5.0	3.0	3.0	4.3	4.0	4.0	4.0	4.0	5.0	3.3	4.0	4.0
9	Avg.HTTP latency	26.8	27.8	24.6	25.4	35.0	36.1	26.7	31.8	27.7	76.5	87.0	78.4	93.6	92.5
	HTTP data transfer	60.15	59.06	64.88	59.89	61.04	64.66	62.36	61.72	59.24	45.08	46.65	49.02	38.34	39.76
	Number of HTTP timeouts	0.00	0.00	0.00	0.00	72.33	91.00	0.33	15.00	0.00	798.00	1475.67	1169.00	2176.33	2656.67
	CPU usage	96.9	97.0	96.5	97.3	97.0	96.9	97.3	97.0	97.4	91.4	88.3	90.8	81.5	86.3
	Memory usage	4.0	4.0	5.0	3.0	3.0	4.3	4.0	4.0	4.0	4.0	5.0	3.3	4.0	4.0

* Throughput (Gbits/sec), Packet loss (%). Usage (%), TCP / UDP latency (usec), Data transfer (GB), Timeouts (count), HTTP latency (ms)

넷째, UDP 평균 지연시간 관련한 측정 결과에서는 특별히 유의미한 차이점을 발견할 수 없었다. 다섯째, HTTP 관련 측정 결과에서는 Flannel(Vxlan), Cililum(Vxlan), Antrea(Geneve)에서 HTTP 타임아웃이 다른 CNI에 비해 많이 발생하였다. 기타 측정 지표는 각 CNI별 차이가 없는 것으로 나타났다.

암호화 통신의 경우, 한 가지를 제외하고 전체 측정 지표에 큰 차이를 보이지 않았으며, HTTP 타임아웃 발생 측정에서 Cilium과 Antrea가 다른 CNI비해 2배에서 3배 많이 발생한 것을 확인할 수 있다.

V. 결 론

쿠버네티스는 다수의 컴퓨터 노드 상에 다양한 응용 프로그램을 내외부를 대상으로 가용성과 내결함성 특징으로 안정적인 서비스를 할 수 있는 기술이나, 쿠버네티스 네트워킹을 구현하는 최적의 컨테이너 네트워크 인터페이스들을 선정하기 위한 성능

분석에 어려움이 있었다.

본 눈문에서는 쿠버네티스 네트워킹 구현을 위하여 쿠버네티스 CNI 성능 측정을 위한 테스트 베드를 설계하고 구현 한 후, 다양한 컨테이너 네트워크 인터페이스들에 대한 프로토콜별 네트워크 처리량, 트래픽 지연시간 등의 성능을 정량적으로 비교 분석하여 특정 워크로드 요구사항에 적합한 CNI를 선택하는 기준을 제안하였다. 컨테이너 네트워크 인터페이스들에 대한 주요 성능 결과를 요약하면 다음과 같다. 첫째, 비암호화 UDP 통신에서 Flannel이 타 CNI 대비 패킷 손실률이 25~70배 높았고, Cilium과 Antrea에서 타 CNI대비 1.6배 높은 CPU 사용률을 보였다. 둘째, TCP 평균 지연시간의 경우 비암호화 통신에서 파드와 파드 간 통신에서는 Calico와 Flannel(IPinIP)가 약 10% 낮은 지연시간을 보였고 파드와 서비스 간 통신에서는 Calico(IPinIP)를 제외하고 비슷한 양상을 보였다. 셋째, 암호화 통신의 경우 Calico-eBPF와 Cilium이 UDP 통신에서 타 CNI 대비 약 5배이상 낮은 패킷 손실률을 보였고, HTTP 통신에서 Cilium과 Antrea가 타 CNI 대비 2배에서 3배이상 많은 타임아웃이 발생하여 다소 열등한 결과를 나타내었다. 이러한 CNI별 차이가 나타나는 이유는 각 CNI가 사용하는 네트워크 터널링 방식, 패킷 필터링 및 처리를 구현하는 방식과 함께 각 CNI를 구성하는 컴포넌트가 상기 기능을 수행하는 리소스 사용량 및 처리 성능이 다르기 때문인 것으로 추정된다.

본 논문에서 제안한 이러한 CNI별 성능 분석 결과는 향후 쿠버네티스 기반 서비스를 설계하고 구축하는데 활용하여 다양한 네트워크 제약사항과 요구사항 효율적인 서비스 구축 및 운영을 가능하게 할 것이다. 이를 통해 서비스의 성능 요구 사항에 맞는 적합한 네트워크 환경을 제공하고 TCP, UDP, HTTP 등의 서비스의 주요 프로토콜에 최적화된 CNI를 제공하여 리소스 효율화 및 서비스 품질 향상에도 기여할 것으로 기대된다.

References

[1] J. E. Shin, J. H. Kwon, and M. H. Kim, "Web-based microservice deployment system in kubernetes environment", Korea Society of Computer and Information, Summer Conference, Jeju, Korea, Vol. 28, No. 2, pp. 45-48, Jul. 2020.

[2] S. W. Lee and J. W. Lee, "Kubernetes of cloud computing based on STRIDE threat modeling", Journal of the Korea Institute of information and Communication Engineering, Vol. 26, No. 7, pp. 1047-1059, Jul. 2022. https://doi.org/10.6109/jkiice.2022.26.7.1047.

[3] H. D. Jin and B. S. Kim, "Types and Comparative Analysis of Kubernetes CNI in Cloud Systems", Journal of the Institute of Electronics and Information Engineers, Vol. 56, No. 8, pp. 21-27, Aug. 2019. https://doi.org/10.5573/ieie.2019.56.8.21.

[4] S. H. Son, H. W. Joo, J. I. Choi, and Y. H. Kim, "Performance Enhancement and Resource Optimization of Cilium CNI through SmartNIC Offloading", Proc. of The Korean Institute of Communications and Information Sciences, Winter Conference, Yongpyong, Korea, pp. 1390-1391, Jan. 2024.

[5] Cluster Networking, https://kubernetes.io/docs/concepts/cluster-administration/networking/. [accessed: Feb. 15, 2025].

[6] S. Susnjara and I. Smalley, What Is Kubernetes Networking?, https://www.ibm.com/topics/kubernetes-networking. [accessed: Feb. 15, 2025]

[7] J. S. Byun, S. S. Kim, H. R. Lee, B. S. Baek, and K. W. Lee, "Exploring Kubernetes-based Network Performance Control in Edge Computing Environments", The Journal of Korean Institute of Next Generation Computing, Vol. 20, No. 1, pp. 114-124, Feb. 2024. https://doi.org/10.23019/kingpc.20.1.202402.009.

[8] W. T. Hong, "Performance monitoring of BBRv2 alpha for parallelism and MTU size on emulation-based environment", Proc. of The Korean Institute of Communications and

Information Sciences, Autumn Conference, Hankyong National University, Anseong, Korea Vol. 27, No. 2, pp. 388-390, Oct. 2023.

[9] S. W. Jun, S. W. Min, and Y. J. Kim, "The Method of Utility with Path MTU Discovery in Fast handoff for seamless data transmission", Proc. of The Korean Institute of Communications and Information Sciences, Summer Conference, Yongpyong, Korea, pp. 1016-1019, Jun. 2005.

[10] Q. M. Nguyen, L. A. Phan, and T. H. Kim, "Operation of kube-proxy in Kubernetes Cluster", Proc. of Korean Institute of Information Scientists and Engineers, Spring Conference, Jeju, Korea, pp. 484-485, Jun. 2021.

[11] S. Y. An and J. W. Park, "Analyzing the impact of eBPF on Linux I/O", Korean Institute of Information Scientists and Engineers, Proc. of the KIISE Korea Computer Congress 2023, Jeju, Korea, pp. 1598-1600, Jun. 2023.

[12] Y. E. Choe, J. S. Shin, and J. W. KIM, "Performance Comparison of eBPF/XDP Packet Filtering for Linux Firewall in Hypervisor-virtualized Machines", The Korean Institute of Communications and Information Sciences, KICS Fall Conference 2019 Program, Kookmin University, Seoul, Korea, pp. 99-101, Nov. 2019.

[13] V. Dakić, J. Redžepagić, M. Bašić, and L. Žgrablić, "Performance and Latency Efficiency Evaluation of Kubernetes Container Network Interfaces for Built-In and Custom Tuned Profiles", Electronics, Vol. 13, No. 19, Oct. 2024. https://doi.org/10.3390/electronics13193972.

[14] W. Jagath and A. Francois, "On the Cost of Tunnel Endpoint Processing in Overlay Virtual Networks", IEEE/ACM 7th International Conference on Utility and Cloud Computing (UCC), London, UK, pp. 756-761, Dec. 2014. https://doi.org/10.1109/UCC.2014.123.

[15] iperf3, https://iperf.fr/. [accessed: Feb. 15, 2025].

[16] sockperf, https://github.com/Mellanox/sockperf. [accessed: Feb. 15, 2025].

[17] wrk, https://github.com/wg/wrk [accessed: Feb. 15, 2025].

[18] C. H. Cho, "A Study on Virtual Interfaces and Hardware Offloading in Linux Kernel", Journal of Knowledge Information Technology and Systems, Vol. 17, No. 2, pp. 271-281, Apr. 2022. https://doi.org/10.34163/jkits.2022.17.2.010.

[19] Y. E. Yoon, M. J. Kim, J. H. Bae, and J. H. Lee, "Performance Comparison Study of Containerd and CRI-O in Kubernetes Environment", Journal of Korean Institute of Information Technology, Vol. 22, No. 6, pp. 95-101, Jun. 2024. https://doi.org/10.14801/jkiit.2024.22.6.95.

[20] Ingress, Kubernetes.Io. https://kubernetes.io/docs/concepts/services-networking/ingress/. [accessed: Feb. 15, 2025].

[21] A. Mateen and M. Zaman, "Congestion Control Algorithms Evaluation of TCP Linux Variants in Dumbbell", Journal of The Institute of Internet, Broadcasting and Communication, Vol. 16, No. 1, pp. 139-145, Jan. 2016. http://dx.doi.org/10.7236/JIIBC.2016.16.1.139.

저자소개

백 승 훈 (Seunghoon Paek)

2020년 8월 : 한국외국어대학교
헝가리어 / EU연계전공(학사)
2025년 2월 : 한양대학교
통신정보공학과(공학석사)
2021년 1월 ~ 2023년 6월 :
삼양데이타시스템 클라우드
엔지니어
2023년 6월 ~ 현재 : NHN Cloud 클라우드 컨설턴트
관심분야 : 클라우드 플랫폼, 마이크로 서비스 아키텍처

구 세 완 (Sewan Gu)

1994년 2월 : 한양대학교
전자통신공학과(공학사)
1996년 2월 : 한양대학교
전자통신공학과(공학석사)
2023년 8월 : 한양대학교
전자컴퓨터통신공학과(공학박사)
1997년 3월 ~ 2000년 3월 :
한국철도기술연구원
2000년 3월 ~ 현재 : LG전자 책임연구원
관심분야 : 클라우드 로보틱스, 네트워크 제어 시스템,
지능형 로봇시스템, 제어이론

윤 동 원 (Dongweon Yoon)

1989년 2월 : 한양대학교
전자통신공학과(공학사)
1992년 2월 : 한양대학교
전자통신공학과(공학석사)
1995년 8월 : 한양대학교
전자통신공학과(공학박사)
2004년 3월 ~ 현재 : 한양대학교
융합전자공학부 석학교수
관심분야 : 통신이론, 위성 및 우주통신, 추정 및 검출