The Journal of Korean Institute of Communications and Information Sciences '24-08 Vol.49 No.08
https://doi.org/10.7840/kics.2024.49.8.1183

A Design and Implementation of a Self-Managed Kubernetes Mobile Edge Cluster

JangWon Lee[◆], YoungHan Kim[°], Dooho Keum[*], Gyu-min Lee[*], Suil Kim[**], Myoung-hun Han[**]

ABSTRACT

In recent years, edge computing has acquired a significant interest in developing and deploying applications by bringing computational resources close to the data source, improving the quality of application experience, and optimizing system resource usage. Edge node deployment strategies in existing studies suppose that once edge nodes are registered in an edge computing system, they will permanently operate with high availability. However, deploying the orchestration solutions in edge computing environments is challenging because of the mobility of edge nodes (mobile nodes) and low availability due to energy limitations and network unstable. To solve this problem, we propose a standalone management structure for deploying mobile nodes in an edge-cloud computing environment that enables mobility support in a Kubernetes-based Edge-Cloud infrastructure. We develop self-managed mobile edge nodes constructed as standalone nodes, ensuring workloads continue operating even when the connection between the mobile node and the cloud is interrupted. Additionally, we consider an energy-aware deployment strategy through energy monitoring and workload scaling schemes at the self-managed mobile node. In this paper, we also implement the proposed architecture on the infrastructure built and established by OpenStack and KubeEdge open-source projects. Measurements show the feasibility of the proposed architecture in deploying mobile nodes that operate independently.

Key Words : Mobile Edge Computing, Monitoring, Cloud Computing, KubeEdge, Energy-aware

Ⅰ. Introduction

Edge computing has emerged as a novel paradigm for data processing, addressing the challenges of the traditional computing model where data from end devices are transmitted to centralized computing servers for storage and analysis[1,2]. However, applications in edge environments, such as the Internet of Things (IoT), have introduced fresh complexities to this conventional approach. IoT ecosystems frequently comprise numerous end devices generating substantial volumes of raw data. The conventional method of transmitting all data to remote computing servers strains network infrastructures and computing resources significantly.

In addition, addressing network latency becomes increasingly arduous, as data traffic often undergoes multiple routing stages and traverses extensive distances before reaching cloud servers. Edge computing offers a solution to these hurdles by facilitating the deployment of edge nodes near data sources, thereby enabling localized data processing. However, implementing computational processing at the edge nodes also has difficulties because edge nodes often have limitations, such as limited battery life, bandwidth costs, computing capacity, and unstable edge network

※ This work was supported by the Agency for Defense Development by the Korean Government(UC210025VD).
◆ First Author : Soongsil University Department of Electronic Engineering, jangwon.lee@dcn.ssu.ac.kr, 학생회원
° Corresponding Author : Soongsil University Department of Electronic Engineering, younghak@ssu.ac.kr, 종신회원
* LIG Nex1, dooho.keum@lignex1.com, 정회원; gyumin.lee@lignex1.com
** Agency for Defense Development(ADD), sikim777@add.re.kr; mengddor@add.re.kr, 정회원
논문번호 : KICS2024-04-069-C-RN, Received April 17, 2024; Revised April 20, 2024; Accepted April 20, 2024

connection[2,3]. Therefore, these difficulties must be considered when designing the infrastructure for edge nodes to optimize performance and increase the longevity of edge networks.

Containerization[4], extensively researched for edge computing solutions, offers a virtualization technique that swiftly deploys and executes edge computing applications on edge devices[5-7]. This method isolates applications within self-sufficient environments and furnishes requisite dependencies (libraries, binaries, and other configuration files) for seamless operation. With the escalating scale of edge computing, container proliferation becomes inevitable. Consequently, managing this burgeoning container count necessitates efficient container orchestration to monitor and manage resource states via multiple edge nodes in an edge computing environment.

Kubernetes[8], a prominent container orchestration platform, has garnered considerable attention due to its capability to manage large container deployments. Recent investigations have explored Kubernetes' utility in orchestrating containers on edge nodes to furnish edge computing solutions[9-12]. However, existing studies[12,13] presuppose continuous availability of registered edge nodes for task processing until they are removed from the edge system. Consequently, edge nodes are typically situated in fixed locations to manage specific tasks within designated areas. While this assumption aligns with specific deployments, such as network routers and smart home devices designed for perpetual online presence, it may not be universally applicable.

Integrating mobile nodes introduces a dynamic dimension to data processing and communication in the edge computing landscape. However, the convergence of mobility and edge computing also brings a significant challenge concerning power and energy consumption[14,15]. For instance, in dynamic environments like military operations[16,17], mobile nodes, such as soldier-worn devices, unmanned aerial vehicles (UAVs), or reconnaissance vehicles, operate under stringent energy constraints due to prolonged deployments and missions. Communication between these mobile nodes and edge computing servers is crucial for real-time data collecting, situational awareness, and mission-critical decision-making. However, the inherent mobility of these nodes, coupled with the difficulties of military operations, exacerbates the challenge of managing power resources effectively. Communication activities, essential for transmitting data to and from edge computing nodes, incur significant energy overheads, rapidly depleting battery reserves. Moreover, the unpredictable and hostile environments where military operations occur necessitate robust communication monitoring and energy-efficient strategies to ensure continuous and reliable connectivity while conserving power.

Several open-source tools developed for the rapidly growing edge infrastructure deployment are Minikube[18], Microk8s[19], k3s[20], and KubeEdge[21,22]. These tools are all lightweight deployments of Kubernetes to enable data processing and analysis on edge devices with limited computational resources, power, and communication capabilities. However, one of the crucial factors that need to be considered in these open sources is mobility, as it directly impacts the implementation strategy, scalability, and performance of edge infrastructure[23].

Unlike the existing deployment strategy in the cloud environment, the connection from mobile edge nodes to the cloud anchor nodes is unstable in a mobile edge environment. Therefore, operation at the mobile node must be possible without connection to the anchor node. Mobile edge nodes must have the ability for standalone operation and an energy-aware strategy to perform their processes and tasks, which necessitate constant mobility in unstable network environments. Therefore, a new architecture is required to deploy, manage, and operate mobile nodes in an edge-cloud computing environment. Ensure connection and independent operation between mobile nodes and cloud servers.

Motivated by this limitation, in this paper, we propose an architecture for implementing mobile nodes in a Kubernetes-based edge-cloud environment with the critical consideration of empowering self-managed mobile nodes with independent operation from the cloud servers while ensuring seamless communication between them. The self-managed mobile nodes have the autonomy to operate independently yet remain interconnected with cloud servers for coordination and data exchange when connected and reactive. We leveraged and improved KubeEdge's architecture for deploying self-managed mobile nodes and added features to support the development of nodes' monitoring and data collection mechanisms. KubeEdge is an edge computing framework constructed atop Kubernetes. It offers functionalities encompassing computer resource management, deployment, runtime, and operational capabilities across geographically distributed edge computing resources.

Additionally, the proposed architecture is enhanced with energy data monitoring and energy-aware decision-making processes among mobile nodes in the network instead of sending all data to centralized servers. By this integration, our design enables flexibility, resiliency, and scalability of cloud-edge computing solutions in self-managed mobile node deployments, especially in some applications where mobility and independence are critical, such as the military.

Our contribution is as follows:

- Propose an architecture for deploying self-managed mobile nodes in Kubernetes-based edge-cloud management. The self-managed mobile node is developed with an independent metadata management system. This allows a workload management system to be configured without a control plan that avoids the unstable network environment between mobile and anchor nodes.
- To consider mobile nodes' energy constraints, we propose energy monitoring and energy-aware schemes in self-managed mobile nodes to design a management operation execution structure that improves node survival time in an unstable network environment.
- The proposal verifies that working through our real implementation experiments is feasible.

The remainder of this paper is structured as follows: Section II introduces the mobile edge-cloud environment by analyzing challenges in the case of edge node implementation. Section III describes our proposal architecture with the enhancement of KubeEdge for mobile node implementation. The system setup for implementation is reported in Section IV, along with our measurements for feasibility verification of the proposed architecture. Finally, the discussion points and our future work are concluded in Section V.

Ⅱ. Mobile Edge-Cloud Environment

As illustrated in Fig. 1, we consider an edge-cloud environment to be a wide range; it may correspond to geographical regions or organizational divisions, reflecting the distributed nature of edge-cloud computing infrastructures.

In this environment, mobile nodes with mobility are initially provisioned in a particular area by the

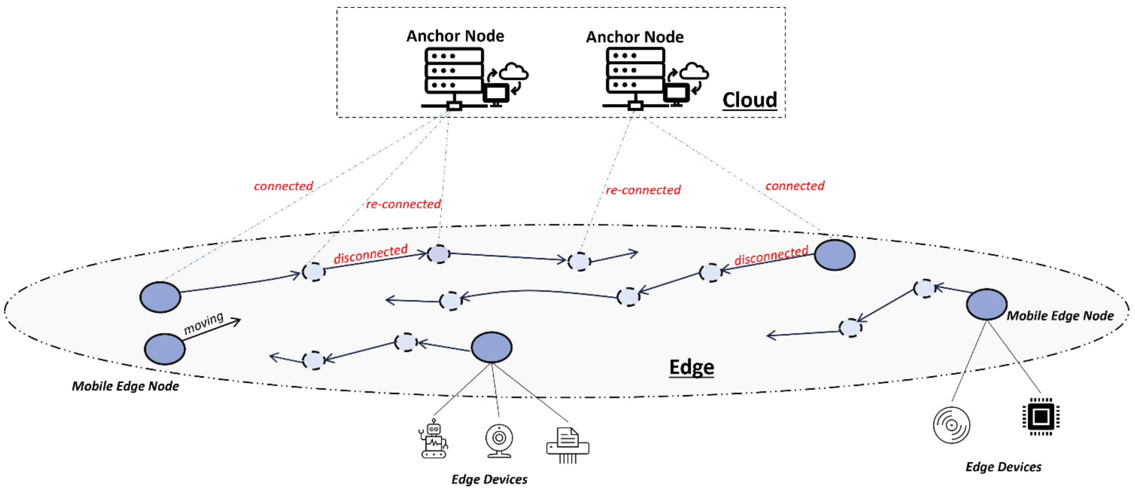


Fig. 1. Mobile Edge Node and Anchor Node with wireless links

anchor node in the cloud. The mobility pattern of moving nodes follows: after being provisioned in a local area and successfully establishing a connection with the initial anchor node, the mobile node collects and processes data from edge devices within its coverage area. However, the edge environment is often unstable and subject to many impacts, so the mobile node may be temporarily disconnected from the anchor node, raising several problems that need to be considered.

2.1 Standalone mobile node for workload operations

The first problem is that the connection to anchor nodes is interrupted, affecting operations between mobile nodes and the cloud. Therefore, it is necessary to have a mechanism to ensure and restore the connection between mobile nodes and anchor nodes. Furthermore, keeping workloads running on mobile nodes unaffected by loss of connection to the anchor node is essential to maintaining network operations and local processes at the edge layer. Therefore, the requirement is to build a mobile node that can operate independently even when it loses connection to the cloud. This means that workloads running on mobile nodes will continue to operate and collect data as if they were still connected to the cloud.

Kubernetes is known as a useful and potential open source for deploying edge infrastructure. Kubernetes is a widely adopted container orchestration platform that automates containerized applications' deployment, scaling, and management. It has become the de facto standard for container orchestration in cloud-native environments. In the design of Kubernetes, workloads are scheduled by the master node and executed at worker nodes. The master node manages and coordinates the operation of application containers; it connects with worker nodes through the kubelet module, as shown in Figure 2.

The kubelet is a critical component of a Kubernetes cluster. An agent runs on each node in the cluster and manages the containers on that node. The kubelet ensures containers are running in a pod, the basic unit of deployment in Kubernetes. It interacts with the Kubernetes API server to receive instructions about

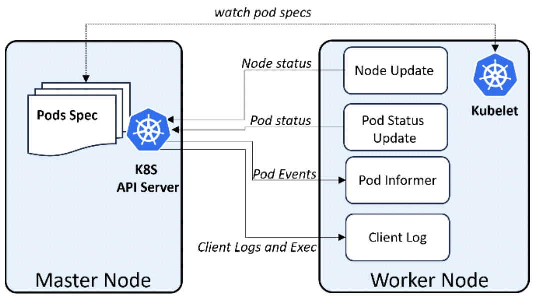


Fig. 2. Kubelet Functionalities in Kubernetes

which pods should be running on the node and takes action to ensure that the desired state matches the actual state of containers on the node. This includes starting, stopping, monitoring containers, and reporting their status back to the control plane. The kubelet also performs various other tasks, such as handling container image management, managing local storage volumes, and executing container health checks. Overall, the kubelet plays a central role in maintaining the health and functionality of Kubernetes nodes within a cluster.

However, when the connection between the master node and a kubelet on a worker node is severed, it disrupts the vital communication link essential for the functioning of the Kubernetes cluster. The master node, which hosts the control plane components, relies on this connection to manage and monitor containers running on the worker node. Consequently, the loss of connectivity results in the control plane's inability to schedule new pods, manage existing ones, or detect changes in their state. Without visibility into the node's containerized workloads, the cluster may experience decreased responsiveness to failures and updates, leading to operational challenges and potential service interruptions. In addition, the original design of Kubernetes was built to orchestrate and manage containers across a large scale and execution size. This results in resource-intensive and complex deployment in resource-constrained edge devices and mobility management capabilities may be limited in highly dynamic environments. Therefore, the current kubelet design is unsuitable for building a standalone mobile node that can operate independently and does not depend on connection to the control plan. This also requires some control plan functions to be moved and deployed to the edge layer.

2.2 Data consistency and synchronization of Mobile node

In unstable network conditions, a critical challenge arises concerning data consistency and synchronization. Intermittent connectivity issues may lead to data discrepancies and potential loss, particularly in distributed systems where data integrity is paramount. To address this challenge, the requirement of a standalone node with robust capabilities for auto-re-establishing connections and ensuring data synchronization becomes imperative. This standalone node serves as a resilient intermediary, capable of autonomously detecting connection disruptions and initiating reconnection attempts to restore communication with both the master node and the affected worker node. Moreover, it must possess sophisticated mechanisms for data synchronization, ensuring that any discrepancies resulting from the connection instability are promptly resolved to maintain data consistency across the cluster.

Integrating a standalone node with auto-reconnection and data synchronization capabilities represents a proactive approach to mitigating the risks associated with unstable connections in Kubernetes clusters, enhancing resilience, and ensuring continuous operation in dynamic deployment scenarios.

2.3 Energy-aware operation

A standalone mobile node with energy-aware aspects is crucial in edge environments, where energy efficiency is paramount. Such a node must feature power-efficient hardware components to minimize power consumption and extend battery life. Dynamic power management techniques allow the node to adapt its energy usage based on workload demands and environmental conditions, ensuring optimal performance while conserving power. Energy-aware networking enables intelligent network connectivity management, reducing energy consumption during data transmission. Real-time energy monitoring and reporting capabilities are required to provide insights into power consumption, facilitating proactive management decisions to optimize efficiency. Additionally, effective battery management and health monitoring mechanisms ensure the longevity and reliability of the node's power source. By integrating these energy-aware aspects, standalone mobile nodes in edge environments can achieve sustainable and efficient operation, meeting the demands of resource-constrained deployment scenarios while contributing to environmental sustainability.

In summary, deploying mobile nodes poses several challenges that must be addressed for effective operation in dynamic and unstable environments: (i) Ensuring the functionality of standalone mobile nodes in the absence of connectivity to the master node is imperative. This necessitates the development of autonomous capabilities within the mobile node to sustain operations independently, mitigating dependencies on central control and enabling seamless functionality even in disconnected scenarios; (ii) Critical concerns include maintaining data resilience, re-establishing connections, and facilitating data synchronization. Mobile edge nodes must have robust mechanisms to handle intermittent connectivity issues, ensuring data integrity and consistency across distributed systems; (iii) Energy awareness is a significant requirement for mobile edge nodes, emphasizing optimizing power consumption and extending battery life. Energy-efficient hardware, dynamic power management techniques, and proactive battery management mechanisms are essential to enhancing the sustainability and efficiency of mobile node deployments in energy-constrained environments. Addressing these three key challenges is essential for enabling mobile edge nodes' successful deployment and operation and supporting resilient and sustainable edge computing ecosystems.

Ⅲ. Proposed Architecture of Self-Managed Mobile Edge Node

In this proposal, the self-managed mobile node concept is built on leveraging an open source that is useful in deploying and managing edge nodes in an edge environment. It's an extension of Kubernetes, tailored specifically for edge computing environments.

However, to satisfy the requirements stated in section III, the basic design architecture of KubeEdge architecture is not enough. Based on that, in this study, we propose and further improve other features based on KubeEdge's architecture to build a strategy to deploy a standalone edge node with energy-aware cognitive operation capabilities.

3.1 Preliminary about KubeEdge.

KubeEdge is an extension to Kubernetes, tailored specifically for edge computing environments. This open-source project, incubated by the CNCF, addresses the unique challenges posed by edge computing, where resources are distributed across a vast network of devices and systems. KubeEdge bridges the gap between cloud and edge by extending Kubernetes' capabilities to the edge nodes, enabling the deployment and management of containerized workloads closer to the data sources. Its architecture decentralizes the control plane, allowing for local execution of workloads and facilitating low-latency data processing while maintaining centralized management and coordination through Kubernetes' familiar APIs. With features such as device management, data synchronization, and event-driven programming models, KubeEdge empowers organizations to harness the potential of edge computing for a wide range of use cases, from industrial IoT to smart cities.

In KubeEdge architecture, a cluster is considered with master nodes on the cloud side and worker nodes on the edge side. The main components of KubeEdge are cloudcore and edgecore, which are responsible for communication between cloud and edge. In cloudcore, CloudHub is constructed as a socket server to update all changes on the cloud side and send them to EdgeHub.

In edgecore, edged is an agent that manages the life cycle of containerized applications inside the pod. The working status of these applications is updated to the cloud by EdgeHub, a web socket client responsible for communicating edge-side to the cloud and cloud-side resources to the edge. Edgecore is responsible for deploying containerized workloads at edge nodes and provides management functionality for the pod lifecycle.

Because of the flexibility in establishing and maintaining connections between edge nodes and servers through cloudcore and edgecore, KubeEdge is a potential platform for standalone mobile node deployment in edge environments. However, the increased energy consumption associated with KubeEdge's resource-intensive nature and network dependency could impact battery life and operational efficiency, which requires the additional monitoring mechanism and energy-aware operation of mobile nodes.

3.2 Proposed Self-Managed Mobile Edge Node Architecture

Figure 3 illustrates our proposed architecture for deploying and managing mobile nodes in the edge-cloud environment. This architecture can deploy and self-operate computing tasks on mobile nodes.

The proposed system relies on the KubeEdge platform to handle communication between mobile nodes and anchor nodes in the cloud. *CloudCore* and *MobileCore* are two main modules that establish and re-establish connections between mobile nodes and anchor nodes. Even in an interrupted connection, the exchange mechanism between these two modules allows them to automatically reconnect when the network status of mobile nodes is available. *EdgeHub* receives all operational information at mobile nodes through *MetaDataManager* and *EdgeRuntime*, which updates the status of active container workloads on mobile nodes. These two modules enable the self-managed mobile node to operate independently from the anchor node. Metadata management allows the workload management process to be configured without a control plane and keeps all workloads running at the mobile node. All activities on the mobile node are maintained and stored at *NodeLocalStore* when the connection to the anchor nodes is interrupted. MobileCore will automatically recover and synchronize data with CloudCore when the connection is restored. The operating mechanism between *CloudCore* and *MobileCore* helps ensure data consistency and synchronization between mobile and anchor nodes in the cloud.

Monitoring Agents Manager, which manages and deploys agents to collect and process data on mobile

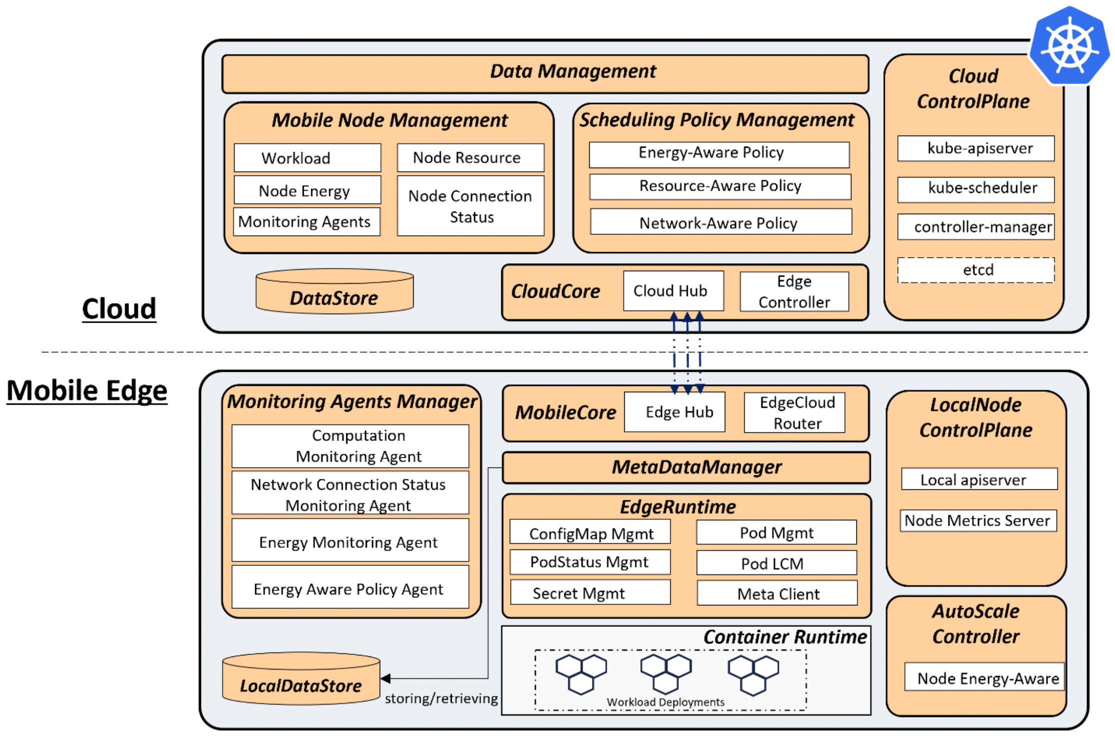


Fig. 3. Architecture for Self-Managed Mobile Node Implementation in Edge Computing Environment

nodes. They operate continuously and monitor the health of the mobile node for a certain period to evaluate the node's condition and ensure timely actions in handling issues related to the mobile node's health. Some of the agents proposed to be deployed in this study include (i) energy monitoring agent, which plays an important role in collecting data on energy consumption of mobile nodes; (ii) network connection status monitoring, collecting data as well as patterns about the network connection of new mobile nodes and anchor nodes in the cloud, (iii) computation monitoring agent, track data about changes in mobile node's computing resources such as CPU and memory, and (iv) energy-aware policy agent, plays the role of making decisions about node operations corresponding to the node's current energy situation. The data collected from these monitoring agents will be stored locally at the node and synchronized to the cloud for further processing.

To build a standalone self-managed mobile node that can operate autonomously based on energy, *AutoScale Controller* is proposed to coordinate node operations. With the node energy-aware mechanism, the *AutoScale Controller* receives data collected about the node's energy and data from the energy-aware policy agent to make decisions regarding the operation of workloads on mobile nodes.

On the cloud side, *Mobile Node Management* plays a role in managing the status of mobile nodes throughout the network. Data about mobile nodes is updated and saved here. They are also the basis for developing workload distribution mechanisms in the *Scheduling Policy Management* module, which coordinates workload deployment on mobile nodes depending on the node's state, such as node energy, node resource, and network.

With this proposed architecture, the operation and management of self-managed mobile nodes in the edge computing environment are deployed wholly and consistently. Self-managed mobile nodes can operate independently even if they lose connection to anchor nodes in the cloud. All workloads are maintained and automatically synchronized with the cloud when there is a connection. Energy is considered for self-operation with a mechanism to control workload and node operations based on energy.

Ⅳ. Implementation

In this section, we present the implementation process on OpenStack Infrastructure and other open-source platforms to verify the feasibility of implementing the proposed architecture for building self-managed mobile nodes.

4.1 Experimental Environment Setup

In this experiment, we construct an environment based on OpenStack and Kubernetes, in which each anchor node is located as a node in a general Kubernetes cluster, and the moving node is composed of edge nodes through KubeEdge as shown in Figure 4. Each anchor node independently configures a cloud core for communication with the edge. At the edge, essential elements for configuring the environment (such as Local DNS), agents for monitoring, scalers for adjusting the number of workloads themselves, and Util and nginx web servers, which may correspond to actual workloads, are operated with 3pods deployed for nginx server.

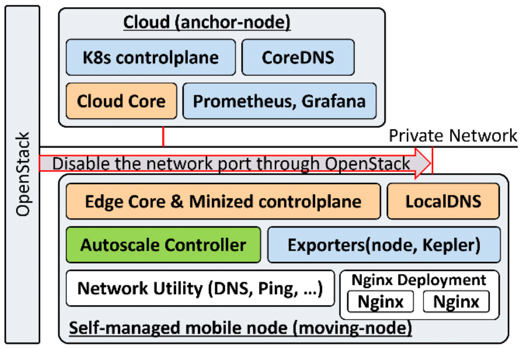


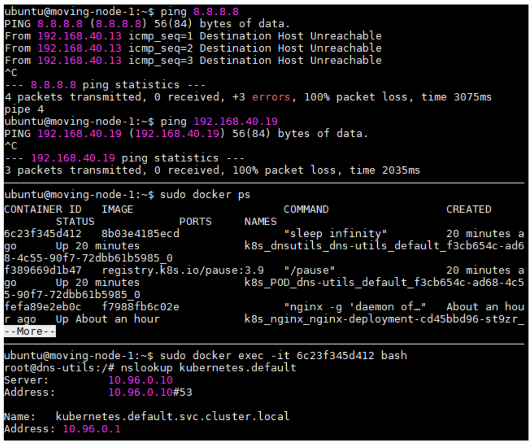
Fig. 4. Experimental Setup in OpenStack

4.2 Self-managed mobile nodes operate under disrupted networks from anchor nodes.

In this experiment, we verify the mobile node's ability to operate independently in an unstable network environment where the mobile node is disconnected from the anchor node. We inject network disruption in moving node-1 by disabling OpenStack's network port to prevent packets from moving node-1.

Network access is impossible after deactivating the network port, as shown by the failed execution of ping commands in Figure 5. However, access and execution of workloads at the moving node are still maintained. by DNS, which can be used through internal access (virsh console) to the mobile node and DNS utility workload. addition, it is confirmed that nginx web server workloads operating inside edge nodes are accessed and operated through IP and FQDN addresses in the same way as in Fig 6.

We concluded that our proposed self-managed mobile node can work independently without connection



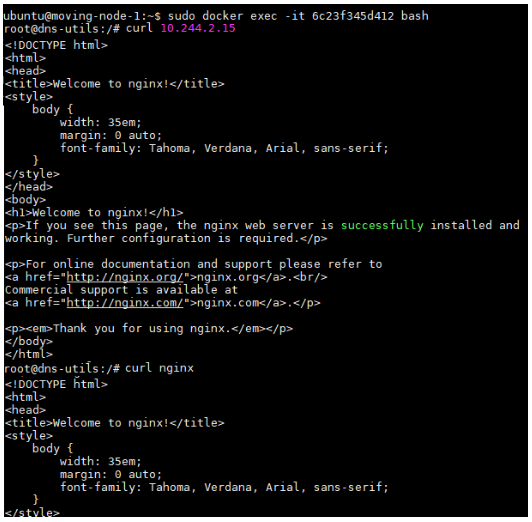
```
ubuntu@moving-node-1:~$ ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
From 192.168.40.13 icmp_seq=1 Destination Host Unreachable
From 192.168.40.13 icmp_seq=2 Destination Host Unreachable
From 192.168.40.13 icmp_seq=3 Destination Host Unreachable
^C
--- 8.8.8.8 ping statistics ---
4 packets transmitted, 0 received, +3 errors, 100% packet loss, time 3075ms
pipe 4
ubuntu@moving-node-1:~$ ping 192.168.40.19
PING 192.168.40.19 (192.168.40.19) 56(84) bytes of data.
^C
--- 192.168.40.19 ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 2035ms
```

```
ubuntu@moving-node-1:~$ sudo docker ps
CONTAINER ID   IMAGE                              COMMAND                   CREATED
        STATUS              PORTS     NAMES
6c23f345d412   8b03e4185ecd                       "sleep infinity"          20 minutes a
go      Up 20 minutes                 k8s_dnsutils_dns-utils_default_f3cb654c-ad6
8-4c55-90f7-72dbb61b5985_0
f389669d1b47   registry.k8s.io/pause:3.9          "/pause"                  20 minutes a
go      Up 20 minutes                 k8s_POD_dns-utils_default_f3cb654c-ad68-4c5
5-90f7-72dbb61b5985_0
fefa89e2eb0c   f7988fb6c02e                       "nginx -g 'daemon of…"   About an hou
r ago   Up About an hour              k8s_nginx_nginx-deployment-cd45bbd96-st9zr_
--More--
```

```
ubuntu@moving-node-1:~$ sudo docker exec -it 6c23f345d412 bash
root@dns-utils:/# nslookup kubernetes.default
Server:         10.96.0.10
Address:        10.96.0.10#53

Name:   kubernetes.default.svc.cluster.local
Address: 10.96.0.1
```

Fig. 5. Network check, DNS check (moving node)



```
ubuntu@moving-node-1:~$ sudo docker exec -it 6c23f345d412 bash
root@dns-utils:/# curl 10.244.2.15
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
root@dns-utils:/# curl nginx
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
```

Fig. 6. The workload is still working under disconnection conditions

to the anchor nodes. The workload deployment is still working usually, as shown in Fig. 6

4.3 Energy-aware workload scaling at mobile node

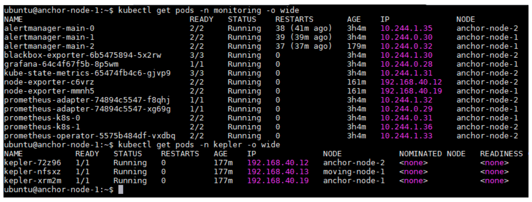
To implement the monitoring part, we set up a general monitoring environment configured through Prometheus and Grafana, and energy monitoring is possible by configuring a Kepler agent. Fig 7 shows the additionally configured monitoring environment.

Monitoring data of the configured environment may be visually checked through the Grafana dashboard, as shown in Fig 8. The first indicates the node's current power situation, and the second indicates the amount of power consumed in the node. PKG refers to CPU-related consumption, and OTHER refers to additional power consumption, such as GPUs, excluding CPUs and memories. Currently, information on elements consumed due to the specificity of the KubeEdge is not displayed in units of workloads, but labeling will be updated later.

Finally, Fig 9 shows the log of scalers operating on edge nodes. It periodically checks the node's energy situation, and when a value below a preset criterion is observed, the scaler scales in workloads and operates internally to reduce energy consumption.

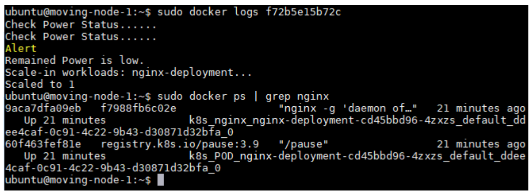
In addition, we evaluate the efficiency of the proposed energy-aware workload scaling scheme at the mobile node by comparing scaling-in and without applying the scaling-in mechanism in the workload deployment of the nginx-server. In particular, in the scaling-in scenario, we adjust the node's remaining energy to a level (<10%), then immediately, the number of nginx pods is scaled to 1pods and in the typical case, the number of pods is kept unchanged (3pods). The result is shown in Fig 10.

As we can see in Fig 10, by applying an energy-aware scaling scheme, the number of workload pods is reduced to the minimum with one pod, and the remaining energy of nodes in low energy conditions is adjusted from 10 percent to 5.25% over 4-time units. We can see that this makes the node



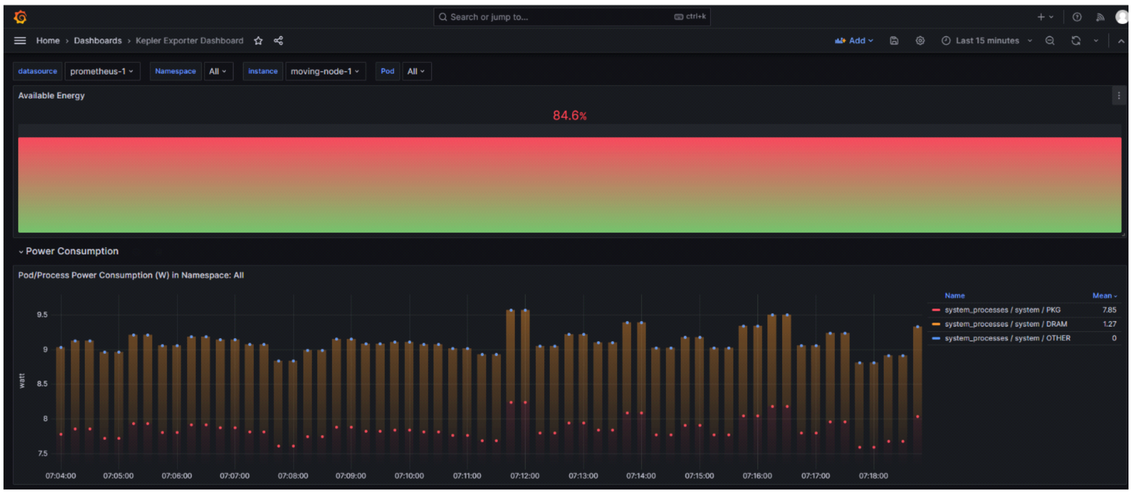
```
ubuntu@anchor-node-1:~$ kubectl get pods -n monitoring -o wide
NAME                                   READY   STATUS    RESTARTS       AGE    IP              NODE
alertmanager-main-0                    2/2     Running   38 (41m ago)   3h4m   10.244.1.35     anchor-node-2
alertmanager-main-1                    2/2     Running   39 (39m ago)   3h4m   10.244.0.30     anchor-node-1
alertmanager-main-2                    2/2     Running   37 (37m ago)   179m   10.244.0.32     anchor-node-1
blackbox-exporter-6b5475894-5x2rw      3/3     Running   0              3h4m   10.244.1.30     anchor-node-2
grafana-64c4f67f5b-8p5wm               1/1     Running   0              3h4m   10.244.0.28     anchor-node-1
kube-state-metrics-65474fb4c6-gjvp9    3/3     Running   0              3h4m   10.244.1.31     anchor-node-2
node-exporter-c6vrz                    2/2     Running   0              161m   192.168.40.12   anchor-node-2
node-exporter-mmnh5                    2/2     Running   0              161m   192.168.40.19   anchor-node-1
prometheus-adapter-74894c5547-f8qhj    1/1     Running   0              3h4m   10.244.1.32     anchor-node-2
prometheus-adapter-74894c5547-xg69g    1/1     Running   0              3h4m   10.244.0.29     anchor-node-1
prometheus-k8s-0                       2/2     Running   0              3h4m   10.244.0.31     anchor-node-1
prometheus-k8s-1                       2/2     Running   0              3h4m   10.244.1.36     anchor-node-2
prometheus-operator-5575b484df-vxdbq   2/2     Running   0              3h4m   10.244.1.33     anchor-node-2
ubuntu@anchor-node-1:~$ kubectl get pods -n kepler -o wide
NAME           READY   STATUS    RESTARTS   AGE    IP              NODE            NOMINATED NODE   READINESS
kepler-72z96   1/1     Running   0          177m   192.168.40.12   anchor-node-2   <none>           <none>
kepler-nfsxz   1/1     Running   0          177m   192.168.40.13   moving-node-1   <none>           <none>
kepler-xrm2m   1/1     Running   0          177m   192.168.40.19   anchor-node-1   <none>           <none>
ubuntu@anchor-node-1:~$
```

Fig. 7. Monitoring Environment



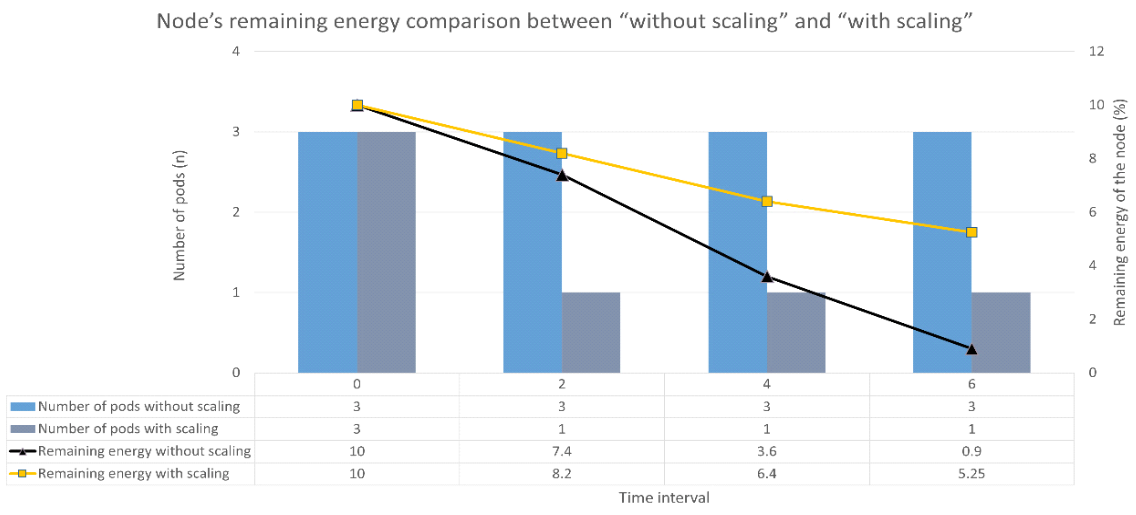
```
ubuntu@moving-node-1:~$ sudo docker logs f72b5e15b72c
Check Power Status......
Check Power Status......
Alert
Remained Power is low.
Scale-in workloads: nginx-deployment...
Scaled to 1
ubuntu@moving-node-1:~$ sudo docker ps | grep nginx
9aca7dfa09eb   f7988fb6c02e                "nginx -g 'daemon of…"   21 minutes ago
   Up 21 minutes             k8s_nginx_nginx-deployment-cd45bbd96-4zxzs_default_dd
ee4caf-0c91-4c22-9b43-d30871d32bfa_0
60f463fef81e   registry.k8s.io/pause:3.9   "/pause"                 21 minutes ago
   Up 21 minutes             k8s_POD_nginx-deployment-cd45bbd96-4zxzs_default_ddee
4caf-0c91-4c22-9b43-d30871d32bfa_0
ubuntu@moving-node-1:~$
```

Fig. 9. Workload Scale-in



	Name	Mean
🟥	system_processes / system / PKG	7.85
🟧	system_processes / system / DRAM	1.27
🟦	system_processes / system / OTHER	0

Fig. 8. Energy Monitoring Dashboard (Up-remained Power, Down-Current usage)



Time interval	0	2	4	6
🟦 Number of pods without scaling	3	3	3	3
▨ Number of pods with scaling	3	1	1	1
▲ Remaining energy without scaling	10	7.4	3.6	0.9
🟨 Remaining energy with scaling	10	8.2	6.4	5.25

Fig. 10. Energy Comparison between Node's Remaining Energy under Energy-aware schemes

survival time longer compared to the normal case. After 4-time units, the node's energy is exhausted nearly to zero; in that condition, the mobile node may not work for any operation.

In summary, the mobile node can work with the proposed features. It can maintain workloads in unstable connection conditions. At the same time, through the energy-scaling deployment strategy, the mobile node self-adjusts its workload to match the available energy level.

V. Conclusion and Future Works

In this study, we consider the problem of building and deploying proposed self-managed mobile nodes in a mobile edge-cloud environment that can operate independently when the connection between the mobile node and the cloud server is interrupted. Original KubeEdge architecture is utilized and improved to build connection modules between the cloud core and mobile code to help establish and re-establish connections between self-managed mobile nodes and anchor nodes in the cloud effectively. At the same time, monitoring agents are also considered for integration and architecture of mobile nodes for the energy-aware development strategy at mobile nodes. With this implementation, it is possible to effectively manage and monitor the lifecycle of self-managed nodes and ensure data integrity in unstable network environments. Through implementation with OpenStack and Kubedge infrastructure, we confirm the feasibility of the proposed architecture. The proposed energy-aware scheme at the self-managed mobile node can ensure a longer node survival time.

In the future, this integration and architecture will consider the development of screening features based on intelligent technologies such as machine learning. It helps monitor the health of self-managed mobile nodes proactively and promptly prov ides timely recovery policies whenever problems occur.

References

[1] G. Carvalho, B. Cabral, V. Pereira, and J. Bernardino, "Edge computing: Current trends, research challenges, and future directions," *Comput.*, vol. 103, pp. 993-1023, 2021. (https://doi.org/10.1007/s00607-020-00896-5)

[2] Q. Luo, S. Hu, C. Li, G. Li, and W. Shi, "Resource scheduling in edge computing: A survey," in *IEEE Commun. Surv. & Tuts.*, vol. 23, no. 4, pp. 2131-2165, Fourth quarter 2021. (https://doi.org/10.1109/COMST.2021.3106401)

[3] Y. Mansouri and M. A. Babar, "A review of edge computing: Features and resource virtualization," *J. Parallel and Distrib. Comput.*, vol. 150, pp. 155-183, 2021.

(https://doi.org/10.1016/j.jpdc.2020.12.015)

[4] O. Bentaleb, A. S. Belloum, A. Sebaa, and A. El-Maouhab, "Containerization technologies: Taxonomies, applications and challenges," *J. Supercomputing*, vol. 78, no. 1, pp. 1144-1181, 2022. (https://doi.org/10.1007/s11227-021-03914-1)

[5] S. Hu, W. Shi, and G. Li, "CEC: A containerized edge computing framework for dynamic resource provisioning," in *IEEE Trans. Mobile Comput.*, vol. 22, no. 7, pp. 3840-3854, Jul. 2023. (https://doi.org/10.1109/TMC.2022.3147800)

[6] O. Oleghe, "Container placement and migration in edge computing: Concept and scheduling models," in *IEEE Access*, vol. 9, pp. 68028-68043, 2021. (https://doi.org/10.1109/ACCESS.2021.3077550)

[7] T. Kim, M. Al-Tarazi, J. W. Lin, and W. Choi, "Optimal container migration for mobile edge computing: Algorithm, system design and implementation," *IEEE Access*, vol. 9, pp. 158074-158090, 2021. (https://doi.org/10.1109/ACCESS.2021.3131643)

[8] N. Poulton, *The Kubernetes book*, NIGEL POULTON LTD, 2023.

[9] S. Böhm and G. Wirtz, "Towards orchestration of cloud-edge architectures with Kubernetes," *Int. Summit Smart City 360°*, pp. 207-230, Cham: Springer International Publishing, 2021. (https://doi.org/10.1007/978-3-031-06371-8_14)

[10] L. A. Phan and T. Kim, "Traffic-aware horizontal pod autoscaler in Kubernetes-based edge computing infrastructure," *IEEE Access*, vol. 10, pp. 18966-18977, 2022. (https://doi.org/10.1109/ACCESS.2022.3150867)

[11] S. Böhm and G. Wirtz, "Cloud-edge orchestration for smart cities: A review of kubernetes-based orchestration architectures," *EAI Endorsed Trans. Smart Cities*, vol. 6, no. 18, pp. e2-e2, 2022. (https://doi.org/10.4108/eetsc.v6i18.1197)

[12] E. Kristiani, C. T. Yang, C. Y. Huang, Y. T. Wang, and P. C. Ko, "The implementation of a cloud-edge computing architecture using OpenStack and kubernetes for air quality monitoring application," *Mobile Netw. and Appl.*, vol. 26, pp. 1070-1092, 2021. (https://doi.org/10.1007/s11036-020-01620-5)

[13] L. Ju, P. Singh, and S. Toor, "Proactive autoscaling for edge computing systems with kubernetes," in *Proc. 14th IEEE/ACM Int. Conf. Utility and Cloud Comput. Companion,* pp. 1-8, Dec. 2021. (https://doi.org/10.1145/3492323.3495588)

[14] P. McEnroe, S. Wang, and M. Liyanage, "A survey on the convergence of edge computing and AI for UAVs: Opportunities and challenges," *IEEE Internet of Things J.*, vol. 9, no. 17, pp. 15435-15459, 2022. (https://doi.org/10.1109/JIOT.2022.3176400)

[15] F. S. Abkenar, P. Ramezani, S. Iranmanesh, S. Murali, D. Chulerttiyawong, X. Wan, A. Jamalipour, and R. Raad, "A survey on mobility of edge computing networks in IoT: State-of-the-art, architectures, and challenges," *IEEE Commun. Surv. and Tuts.* vol. 24, no. 4, pp. 2329-2365, 2022. (https://doi.org/10.1109/COMST.2022.3211462)

[16] P. McEnroe, S. Wang, and M. Liyanage, "A survey on the convergence of edge computing and AI for UAVs: Opportunities and challenges," in *IEEE Internet of Things J.*, vol. 9, no. 17, pp. 15435-15459, Sep. 2022. (https://doi.org/10.1109/JIOT.2022.3176400)

[17] R. Bajracharya, R. Shrestha, S. A. Hassan, H. Jung, and H. Shin, "5G and beyond private military communication: Trend, requirements, challenges and enablers," in *IEEE Access*, vol. 11, pp. 83996-84012, 2023. (https://doi.org/10.1109/ACCESS.2023.3303211)

[18] The Kubernetes Authors, *Minikube start(2024)*, Retrieved Mar. 4, 2024, from https://minikube.sigs.k8s.io/docs/start

[19] Canonical Ltd., *MicroK8s - The effortless*

Kubernetes, Retrieved Mar. 4, 2024, from https://microk8s.io/

[20] K3s Project Authors, *K3s - Lightweight Kubernetes,* Retrieved Mar. 4, 2024, from https://docs.k3s.io/

[21] Y. Xiong, Y. Sun, L. Xing, and Y. Huang, "Extend cloud to edge with kubeedge," in *Proc. IEEE/ACM Symp. Edge Comput. (SEC)*, pp. 373-377, Oct. 2018. (https://doi.org/10.1109/SEC.2018.00048)

[22] KubeEdge Project Authors, *KubeEdge - Why KubeEdge*. Retrieved Mar. 4, 2024, https://kubeedge.io/docs/

[23] R. Singh, R. Sukapuram, and S. Chakraborty, "A survey of mobility-aware multi-access edge computing: Challenges, use cases and future directions," *Ad Hoc Netw.*, 140, p. 103044, 2023. (https://doi.org/10.1016/j.adhoc.2022.103044)

JangWon Lee

Feb. 2020 : B.S. degree, Soongsil University
Feb. 2022 : M.S. degree, Soongsil University
Mar. 2022~Current : Ph.D. student, Soongsil University

<Research Interests> Cloud computing, Autonomous, and 5G/6G network infrastructure
[ORCID:0009-0002-6194-2520]

YoungHan Kim

Feb. 1984 : B.S. degree, Seoul University
Feb. 1986 : M.S. degree, KAIST
Feb. 1990 : Ph.D. degree, KAIST
Mar. 1994~Current : Professor, Soongsil University

<Research Interests> ICT technology, wireless communications, data communications
[ORCID:0000-0002-1066-4818]

Dooho Keum

Feb. 2015 : M.S. degree, Ajou University
Feb. 2020 : Ph.D. degree, Ajou University
Jan. 2022~Current : Senior Research Engineer, LIG Nex1

<Research Interests> Military IoT, Trust based Routing, and Wireless Ad-hoc/Mesh/Sensor Networks
[ORCID:0000-0002-8267-2331]

Gyu-min Lee

Feb. 2014 : B.S. degree, Ajou University
Feb. 2016 : M.S. degree, Ajou University
Aug. 2022 : Ph.D. degree, Ajou University
Feb. 2023~Current : Senior Research Engineer, LIG Nex1

<Research Interests> C5ISR, Tactical network architecture, SDN/NFV
[ORCID:0000-0002-6384-795X]

Suil Kim

Feb. 1986 : B.S. degree, Soongsil University
Feb. 1988 : M.S. degree, Soongsil University
Aug. 2000 : Ph.D. degree, KAIST
Mar. 1988~Current : ADD

<Research Interests> Military Tactical Satellite, Wireless communications, Network Centric Operation Environment

Myoung-hun Han

Feb. 2007 : B.S. degree, Chung-Ang University

Aug. 2009 : M.S. degree, Chung-Ang University

Aug. 2021 : Ph.D. degree, Chung-Ang University

Oct. 2014~Current : ADD

<Research Interests> Tactical Network, Network M&S, Satellite Network