Decentralized and Revised Content-Centric Networking-Based Service Deployment and Discovery Platform in Mobile Edge Computing for IoT Devices

Tien-Dung Nguyen, *Member, IEEE*, Eui-Nam Huh, *Member, IEEE*, and Minho Jo, *Senior Member, IEEE*

***Abstract*—Mobile edge computing (MEC) is used to offload services (tasks) from cloud computing in order to deliver those services to mobile Internet of Things (IoT) devices near mobile edge nodes. However, even though there are advantages to MEC, we face many significant problems, such as how a service provider (SP) deploys requested services efficiently on a destination MEC node, and how to discover existing services in neighboring MEC nodes to save edge resources. In this paper, we present a decentralized and revised content-centric networking (CCN)-based MEC service deployment/discovery protocol and platform. We organized a gateway in every area according to a three-tiered hierarchical MEC network topology to reduce computing overhead at the centralized controller. We revised CCN to introduce a protocol to help SP deploy their service on MEC node and assist MEC node discover services in neighboring nodes. By using our proposed protocol, MEC nodes can deploy or discover the requested service instances in the proximity of IoT devices to reduce transmission delay. We also present a mathematical model to calculate the round trip time to guarantee quality of service. Numerical experiments measure the performance of our proposed method with various mobile IoT device services. The results show that the proposed service deployment protocol and platform reduce the average service delay by up to 50% compared to legacy cloud. In addition, the proposed method outperforms the legacy protocol of the MEC environment in service discovery time.**

***Index Terms*—Cloud, fog, Internet of Things (IoT), mobile edge computing (MEC), offloading, service deployment, service discovery.**

I. Introduction

THE CONCEPT of mobile cloud computing (MCC) [1] was introduced to assist mobile Internet of Things (IoT) devices for high-intensive computing applications. Offloading the heavy computing tasks of mobile IoT devices to a centralized core cloud (CC) reduces the workload in mobile IoT devices (such as mobile phones, smart watches, smart glasses, sensors, etc.) and accelerates computation time. For example, if a user wears smart glasses to find a good restaurant with special discounts, the mobile IoT device first needs to send location information and the requested voice command (e.g., restaurant with special discount) to the service deployed in the cloud. The service then analyzes the command, checks the database, and finally sends matching results to the mobile IoT device. Although the processing time is extremely fast, transmission delay due to the distance from the mobile IoT device to the CC will degrade the performance of the service. Besides, if a large number of mobile IoT devices simultaneously request cloud services from the same node, the demands will impose a heavy burden on the backhaul links, which affects the latency of other cloud services.

Mobile edge computing (MEC) [2] was recently proposed to solve the problems in MCC. MEC is known as a decentralized cloud, or distributed cloud, from its offloading of tasks or services to mobile computing from cloud computing. MEC nodes (MECs) are located near mobile IoT devices so they can solve the latency problem of transmission too. The fifth generation (5G) mobile network brings many advantages to the MEC environment, such as high bandwidth, high reliability, and low-latency. And 5G technology is the main condition for quick development of cloud services, especially for location-based games, such as Pokemon [3], and virtual reality (VR) applications like Facebook spaces [4]. For example, the communications link in IEEE 802.11p is limited to 10 Mb/s per channel, which cannot satisfy the quality of service (QoS) for augmented reality applications on video transmission (required data rate: 40 Mb/s) [5]. Meanwhile, the 5G cellular network with wide-spectrum (28 GHz), multiple input/output, and ultradense network technologies can support a 7.5 Gb/s data rate, and achieve stable connections at 1.2 Gb/s in a mobile environment at speeds of 100 km/h [6].

In order to enable services running on MECs, they need to be deployed in the corresponding MEC itself. In this paper, we mainly focus on IoT applications, such as VR application or Facebook Spaces, in which IoT devices only transmit data to MEC when requested by users. For sake of simplicity, we suppose that an IoT application is deployed as a service instance

Manuscript received May 30, 2018; revised August 18, 2018; accepted October 5, 2018. Date of publication October 11, 2018; date of current version June 19, 2019. This work was supported in part by the Institute for Information and Communications Technology Promotion (IITP) grant funded by the Korea Government [Ministry of Science and ICT (MSIT)] under Grant 2017-0-00294 and in part by the MSIT, South Korea, through the Information Technology Research Center support program supervised by the IITP under Grant IITP-2018-2013-1-00717. *(Corresponding authors: Eui-Nam Huh and Minho Jo.)*

T.-D. Nguyen and E.-N. Huh are with the Department of Computer Science and Engineering, Kyung Hee University, Yongin 17104, South Korea (e-mail: ntiendung@khu.ac.kr; johnhuh@khu.ac.kr).

M. Jo is with the Department of Computer Convergence Software, Korea University, Sejong 30019, South Korea (e-mail: minhojo@korea.ac.kr).

Digital Object Identifier 10.1109/JIOT.2018.2875489

and stored at MEC. Service instances can be started/shutdown depending on user requests. Traditionally, service providers (SPs) have to predeploy services at MECs, like infrastructure as a service [7]. It means that SPs manually set up and run their services in target MECs before they start providing services for mobile IoT devices. Unfortunately, SPs do not know exactly where services should be deployed to guarantee the requested service to all mobile IoT devices. Thus, SPs may have to deploy the services either on all MECs or on predicted MECs. In the case of deploying services on all MECs, some of them that receive no requests from mobile IoT devices are wasting computing resources. In the latter case, there are two scenarios: when the predicted MEC for the service request is correct, it can perfectly provide the service requested from the IoT mobile device, but when SPs fail to predict the right MECs (i.e., a service is not running on the target MECs), the target MECs forward requests to the CC. Considering these cases, predeployment of services on all MECs is obviously inefficient, resulting in wasted computing resources and low availability. Besides, since edge computing resources are limited, an MEC can only run a limited number of services. Running unnecessary services will degrade the availability of other services. Therefore, it is necessary to achieve a mechanism for SPs to deploy the requested services on target MECs at run time, rather than predeploy them. In this paper, we present a decentralized and revised content-centric networking (CCN)-based MEC platform with a service deployment protocol to assist SPs in self-deploying their services on MECs whenever a service is requested. The term *self-deploy* means that the requested service will be deployed by SPs in MECs automatically without interaction with the infrastructure or by SP administrators. Entire systems, including MECs and SPs, will automatically deploy the requested services by themselves whenever service requests arrive. To enable the self-deployment feature, we implement a list of representational state transfer (RESTful) application programming interfaces (APIs) so SPs can remotely deploy their services in MEC. When a new service request from an mobile IoT device is sent to MEC, the information, including IP and port, of the MECs (or destinations) will be transferred to SPs by way of our proposed service deployment protocol. Then, SPs deploy the requested services easily using the information about the MECs (destinations).

In our proposed method, not all requested services need to be deployed on MECs. If any MECs receive a service request, but the requested service is running on neighboring MECs and the transmission delay from the requesting mobile IoT device to the neighboring MECs is acceptable (within the QoS level), the receiving MECs will forward the request messages to the neighboring MECs, instead of deploying the requested service themselves. This will save edge computing resources. Discovering requested services in neighboring MECs is also called *service discovery* in some of the literature. Traditional service discovery in MEC is based on the concept of software defined network (SDN) [8]. An SDN currently uses a centralized controller to communicate with all MECs in the network. As a mobile IoT device requests a service, MECs just ask the SDN controller to find the service's location. However, this centralized controller may create overhead and increase transmission delay due to distances between the SDN controller and the MECs. Therefore, we propose designing the control plane based on a three-tiered hierarchical MEC network topology that allows for decentralizing the controller. By using a three-tiered hierarchical network topology, all MECs are divided into smaller groups/clusters. Each group/cluster consists of MECs located in the same region, and they are managed by a group/cluster-head node [an regional cloud (RC)]. In this way, each RC manages fewer number of MECs than the traditional centralized controller. Therefore, the overhead from the centralized SDN controller is fortunately diverted to the RCs.

In this paper, we present an efficient service discovery protocol based on the CCN [9]. The CCN is known as a receiver-driven data-centric communication protocol. The CCN deploys tables on every network node to store content information so that incoming packets will be forwarded to the right destination nodes. However, since the existing CCN is designed for cache-based applications, such as Web pages, videos, documents, etc., and is designed for discovering content based on hierarchical naming schemes, it cannot be applied for edge computing services that are distinguished by IP addresses. To deploy such a distributed control-plane for edge computing services, we need to redesign the CCN tables to store information about edge computing services in each node (including the IP address and port of the MECs running the service). Using our redesigned CCN table, we introduce a service discovery protocol between MECs, a so-called CCN-based service discovery (C-SD) protocol, in order to share service information. Based on the revised CCN tables of each MEC, C-SD protocol finds the requested service running in neighboring MECs, not only in same region but in different regions as well. In other words, with C-SD, MECs can self-discover the requested service in neighboring MECs based on the revised CCN tables. We use the term *self-discovery* to emphasize the ability to share information among the MECs, rather than contacting a traditional centralized controller. An MEC that includes the features of both service self-discovery and self-deployment will be called automatic MEC.

Moreover, if there are many neighboring MEC nodes running one requested service, the one that should be chosen must guarantee the QoS of the service requested by the mobile IoT device. In this paper, we model the round trip time (RTT) from mobile IoT devices to neighboring MECs so we can choose the best destination node with respect to QoS. On the other hand, there will be a strain on processing time if multiple mobile IoT devices access the same MECs for services. Therefore, the service discovery protocol has to consider the processing burden on MECs to avoid QoS violations. In this paper, we also model the processing burden when making discovery decisions in order to guarantee QoS.

The main contributions of this paper are summarized as follows.

1) We solve overhead and transmission delay problems with service deployment and discovery, which are caused by a traditional SDN controller.

2) To solve the above problems, we introduce a revised CCN and a three-tiered hierarchical MEC network topology. We put MECs into multiple clusters/groups based on geographic location in order to decentralize control.
3) By introducing a revised CCN table to overcome problems, the traditional CCN cannot be used for an edge computing environment. Based on the revised CCN, we propose a combination of revised CCN-based service deployment and C-SD protocols, the so-called revised CCN-based service deployment and discovery protocol, to assist MECs self-deploy requested services and self-discover requested services in neighboring MECs.
4) We avoid duplicate deployments by discovering the same services running on neighboring MECs.
5) We verify the superiority of the proposed method with experimental simulation results. We measure the performance of our proposed decentralized and revised CCN-based MEC service deployment/discovery protocol and platform with three different applications [artificial intelligence (AI) Darknet-based object detection, data transfer, and Web applications]. We also carry out extensive experiments with real-world data to validate the performance of C-SDD, employing a Twitter dataset and a base station (BS) dataset in three major cities in the United States: Los Angeles, Phoenix, and New York City. Our proposed work shows decreases in the average service delay of up to 50%, compared to the legacy cloud. In addition, the proposed method outperforms the legacy protocol of the MEC environment in service discovery time.

The rest of this paper is organized as follows. Related works are reviewed in Section II. Section III describes the proposed decentralized and revised CCN-based MEC platform. We present our revised CCN-based service deployment discovery protocol in Section IV. We then present the service discovery decision model in Section V and the performance analysis in Section VI. We provide our implementation and performance evaluation in Section VII, and finally, Section VIII concludes this paper.

II. Related Works

The decentralized cloud has been a concern more and more in recent years and several new concepts have been proposed. The cloud radio access network introduced by China Mobile Research Institute in 2010, is a combination of a legacy radio access network and cloud computing at a BS to improve performance. Cisco introduced fog/edge computing for the IoT. They moved computing applications, data, and services away from centralized nodes to the network edge, and enabled analytics and knowledge generation to be processed in close proximity to data sources. Cloudlet [10], [11] is another concept known as a limited-resource cloud node located close to mobile IoT devices. Cloudlet, however, is concentrated mainly in cloud computing resources rather than network communications. In 2014, the European Telecommunications Standards Institute introduced MEC [2], which pushes the cloud computing capabilities close to the BS, and applies to radio access networks in 4G, and 5G. However, no concept mentions how to deploy edge computing services to MECs, as well as how to discover an existed edge computing service over the network. There was an effort to discover a shortest and feasible neighboring fog as an alternative which can provide the same service when a fog is overloaded [12].

Cloud4IoT [13] is a framework to provide cloud resources for IoT devices with features of automatic deployment and service discovery. However, the authors focused on automatic deployment for IoT devices, rather than edge computing services. Cloud4IoT does not describe service discovery in detail, and does not consider QoS parameters. Brogi and Forti [14] mentioned problems with service deployment in the IoT with respect to QoS constraints. However, they concentrated on finding a location to deploy a service to guarantee the QoS constraints, rather than describing how the service can be deployed and how to discover an existing service already running in the system.

Recently, the SDN has been considered in order to facilitate MEC [15]. By using the OpenFlow protocol [16], an SDN controller can deploy the paths on each switch or router to route requests from end users to the edge computing services. The northbound applications (e.g., a load balancer, optimal resource allocation, mobility management, and other operational tasks) are used to optimize cloud and network resources while guaranteeing QoS. Oueis *et al.* [17] took into account joint computational and link load balancing that minimizes power consumption per user with respect to QoS for each user. Sathyanarayana and Moh [18] proposed a load balancing mechanism between throughput of the network and the workloads on servers. Lin *et al.* [19] utilized an SDN-based cloud network architecture to optimize throughput and energy based on flow and virtual machine (VM) migration delay. However, these papers only focus on the centralized controller which can create overhead and takes more time to obtain the optimal discovery decision.

Information-centric networking (ICN) [20] architectures leverage in-network storage for caching, multiparty communications through replication, and interaction models that decouple senders and receivers. ICN has been explored in several projects, such as the data-oriented network architecture [21], CCN [9], the publish-subscribe Internet routing paradigm [22], and network of information from the design for the future Internet project [23]. Among these projects, only CCN with its service information tables could be applied to discovering edge computing services in a distributed cloud.

The combination of CCN and SDN has been studied and explored in [24]–[26]. However, the overhead created by the core controller has not been addressed, and the proposals in these papers only analyzed and simulated simple content/data messages, rather than edge computing services. Pan *et al.* [27] presented an edge cloud framework (HomeCloud) by using ICN and SDN to connect IoT applications running at HomeCloud datacenter. Besides, the edge service for IoT by enabling the combination of ICN and SDN has been studied in [28] and [29]. However, service instance

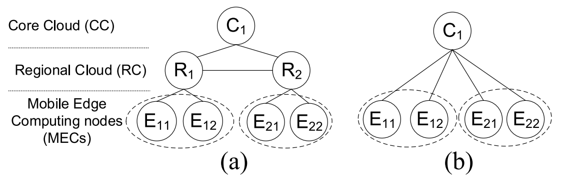

Fig. 1. Three-tiered hierarchical MEC network topology. (a) 3-tier. (b) 2-tier.

deployment as well as QoS management have not been fully described, and they only consider network topology as flat or two-layer network, rather than three-tiered hierarchical network topology.

III. Proposed Service Deployment/Discovery MEC Platform

A. Network Topology

In this paper, we consider a three-tiered hierarchical network topology as shown in Fig. 1. Layer 1 consists of access points or BSs, the so-called MECs, which directly communicate with mobile IoT devices through WiFi or mobile networks. One node (a so-called RC) will manage MECs located in same area. RCs (in layer 2) are able to communicate and share information with each other. RCs are managed by the CC, which is the central management node of the network operator (layer 3). To simplify from now on, we use the term *node* to mean MECs, RCs, and the CC.

B. MEC Architecture

Fig. 2 illustrates the detailed architecture of an MEC node, including four main components: 1) the aggregator; 2) the edge computing service pool; 3) the edge computing service controller; and 4) the service information database.

1) Aggregator: The aggregator component represents either a WiFi access point or a mobile BS. The aggregator transmits incoming requests from mobile IoT devices to the edge computing service management module.

2) Edge Computing Service Pool: A node provides computing resources for an edge computing service by using virtualization technology, such as Docker or a VM to create service instances (e.g., edge computing service CS_1, CS_2,..., CS_n). Basically, an edge computing service can be deployed as one or many instances. But in this paper, we implement one instance to represent one edge computing service. The limited number of service instances depends on the resource capacity of each node.

3) Edge Computing Service Controller: This is a software component to control edge computing services for each node, including the following four modules.

1) *Service Instance Manager:* It is designed to control the service instances. Since Docker performs better than a VM in terms of quick-deploying services [30], we implement a Docker manager to assist SPs in deploying their edge computing services in Docker containers. It includes a list of RESTful APIs for the front end to communicate with SPs, and list of Python APIs for the back end to communicate with the Docker manager. The Docker manager has administrator privileges to initialize/start/stop the service instances.
2) *Service Aware:* It is used to distinguish registered edge computing services from normal services. If an SP wants to deploy services on nodes, it has to register the services in the CC. The CC then broadcasts this information to all nodes in the system. For the unregistered services, the nodes will forward requests to the network as usual without any processing. The Service Aware module detects edge computing services based on IP addresses and ports. In practical terms, each edge computing service has a unique IP address and port (address of the end-point cloud service). If a mobile IoT device wants to request service A, it has to send a request to the end-point IP address and port of the service. Service Aware acts as a packet sniffer to determine whether an incoming request is for an edge computing service or not.
3) *Service Discovery:* It is based on the service information in the database, and discovers services running in neighboring MECs.
4) *Local Routing Table:* It stores a list of routing rules to route incoming packets to the target destination. It is implemented by destination network address translation with a netfilter to redirect a service packet from an original destination to an edge service's location.

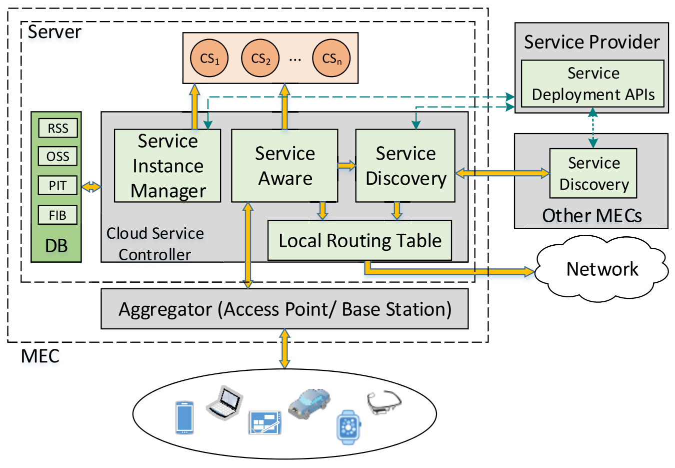

Fig. 2. MEC architecture.

4) Service Information Database: This stores edge computing services information, and consists of the following four tables.

1) *Registered Service Store (RSS):* It has the format *<Service ID - Service IP - Port>* to store information on all edge computing services that have been registered with the infrastructure provider.
2) *Forwarding Information Base (FIB):* It has the format *<Service ID - Next - Port>* to store neighboring MECs (in *Next*) in which an edge computing service is running.
3) *Pending Interest Table (PIT)* It has the format *<Service ID - Requester>* to store requester and service pairs that are waiting for the results of service discovery. In there

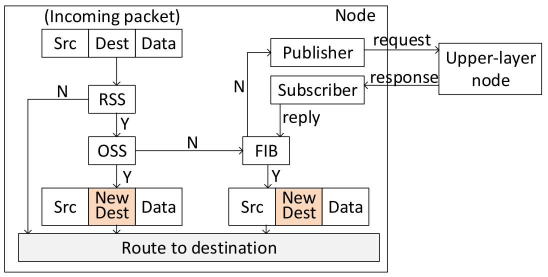

Fig. 3. Logical process at node.

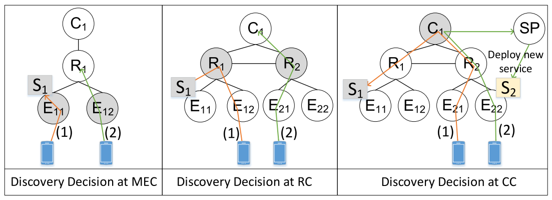

Fig. 4. Discovery decision at MEC, RC, and CC.

are many incoming requests from the same source to the same service, PIT will store only the first request and ignore the others.

4) *Online Service Store (OSS):* It has the format <*Service ID - Service IP - Port - Container ID*> to store service information, such as container ID, local IP address, and the local port of the container running on that node.

IV. Revised CCN-Based Service Deployment Discovery Protocol

In this section, we describe the procedure of the revised CCN-based service deployment discovery protocol for MEC.

A. Logical Process in General Node

Fig. 3 shows the logical process of a node based on RSS, OSS, and FIB tables. An incoming request consists of three main fields: 1) source; 2) destination; and 3) data. As a packet arrives, the Service Aware module checks whether the destination exists in the RSS table or not. If the destination does not exist in the RSS table (i.e., the service was not registered to use the infrastructure), the Service Aware module routes the packet to the destination as a normal packet. Otherwise, the Service Aware checks for its existence in the OSS table to find whether a container of the service is already running or not. If the service is running in a container, the node immediately forwards the packet to the container to process the service. Otherwise, it looks in the FIB table to find which neighboring MECs are hosting the service. If the service exists in the FIB table, it forwards the packet to that neighboring MEC. Otherwise, it requests the service discovery module to discover where the service is stored. In order to deploy a communications method between nodes, we can apply various messaging protocols, such as message queuing telemetry transport (MQTT) protocol [31], advanced message queuing protocol [32], or simple text orientated messaging protocol [33].

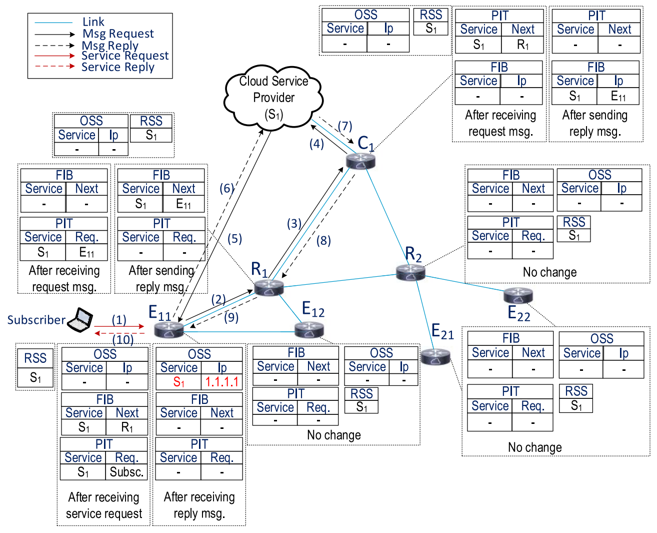
Link
Msg Request
Msg Reply
Service Request
Service Reply
Cloud Service Provider (S1)
Subscriber
After receiving request msg.
After sending reply msg.
After receiving service request
After receiving reply msg.
No change
1.1.1.1

Fig. 5. Service initialization stage.

B. Service Discovery Description

The aforementioned service selection/discovery problem can be solved by the service discovery protocol. The discovery decisions are considered at three layers: 1) MEC; 2) RC; and 3) CC, as shown in Fig. 4. For a discovery decision in MEC, when a mobile IoT device requests a service, if the requested service is running in MEC, or a discovery rule for the service exists in the FIB table, MEC will automatically route the request packet to the destination. Otherwise, MEC will ask its connected RC for the service. For a discovery decision at the RC, the RC will look up the requested service from all MECs located in its region, and replies with the destination of the requested service. If no requested service is running in its region, the RC will ask its connected CC for the service. At the last layer, if the CC finds the service running in any region from which QoS is guaranteed, it replies to the RC. If there is no service already running in the system, or if QoS is not guaranteed, the CC will send a request to the SP to deploy the service at the requested node.

In our scenario, we use two main stages (service initialization and service discovery) to explain our service discovery protocol in detail.

1) Service Initialization: In this stage, the SP first deploys services in MEC. Fig. 5 describes the step-by-step service deployment, as well as updates in the tables. The message structure is designed in the following format: $<MsgType - Source - ServiceID - SerivceIP - port - EdgeNodeIP>$. Suppose Service S_1 has been registered and stored in all nodes' RSS tables. As a mobile IoT device requests service S_1, MEC E_{11} adds the pair $<S_1; Subscriber>$ into the PIT table and finds S_1 in the OSS table and the FIB table. Because it is the initialization stage, there is no S_1 information in MEC, RC, or CC. E_{11} sends a message to ask the RC for S_1 (step 2), and the RC then sends a message to the CC (step 3). The CC sends a message including the service information and IP address of E_{11} to the SP. The SP uses RESTful APIs to deploy service S_1 on E_{11} in step 5, and E_{11} replies with the information about the container in step 6. As the service is

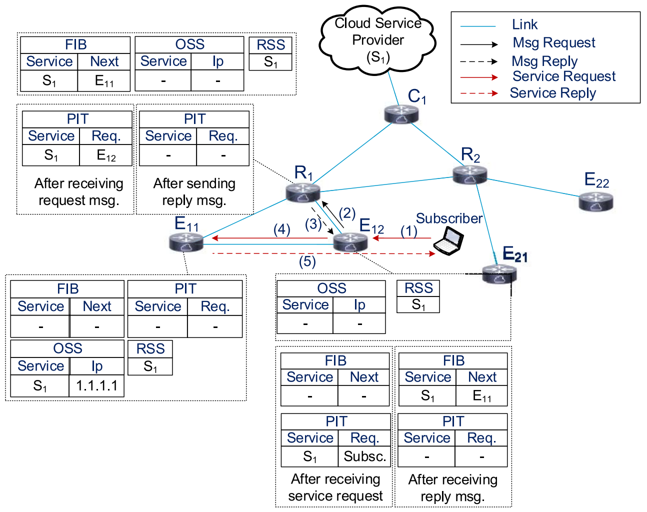
Fig. 6. Service in the same region.

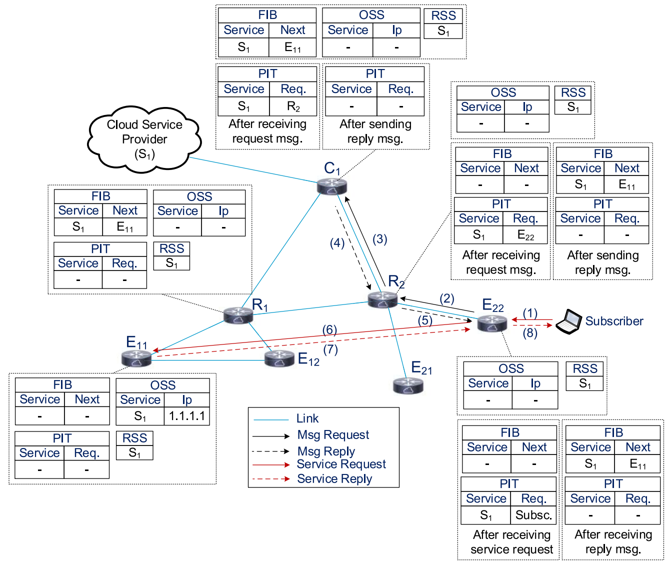
Fig. 7. Service in neighboring region.

deployed, the OSS table will be updated with the container IP address of S_1. After receiving reply messages in steps 7–9, the CC and RC consequently update the FIB table and clear the PIT table. MEC does not need to update the FIB table because the service was already updated in the OSS table when S_1 was completely deployed in the container. MEC then clears the PIT tables and forwards the requested packets to the service destination in the OSS table. Therefore, thanks to the control messages in C-SDD and the edge computing service API, the requested service can be self-deployed in the specific MEC.

2) Service Discovery: In this stage, we consider three cases: 1) service discovery in the same region; 2) discovery in a different region; and 3) if no existing service satisfies the QoS requirements.

Case 1: Fig. 6 presents how E_{12} discovers a service running in E_{11}. When a request for service S_1 arrives, MEC E_{12} uses the control plane to find the destination of S_1. E_{12} sends a message to RC R_1 to ask for service S_1. R_1 finds S_1 in the FIB table and immediately replies to E_{12}. E_{12} updates its FIB table and forwards the requested package to E_{11}. If one service is running in many MECs, the RC randomly chooses MEC E_i in which the network condition between the source (E_{12}) and the destination (E_i) satisfies the QoS requirements of S_1.

Case 2: This case is similar to case 1 but extends to other regions, as shown in Fig. 7. It takes four steps (2–5) to discover service S_1 from the CC, because the S_1 information is only stored in E_{11}, R_1, and C_1. If the service is running in many MECs, C_1 will randomly choose the MEC in which the network condition between source and destination satisfies QoS requirements, and replies to R_2, E_{22} in steps 4 and 5. After step 5, E_{22} updates the FIB table and forwards the requested packet to E_{11} where service S_1 is running.

Case 3: This happens when MEC, the RC, or the CC cannot find any existing service that satisfies QoS. In Fig. 8, when R_2 asks C_1 for service S_1, C_1 collects network bandwidth from the source (E_{12}) to the destination (E_{11}), and realizes that QoS for service S_1 is not guaranteed. C_1 sends a request to the SP

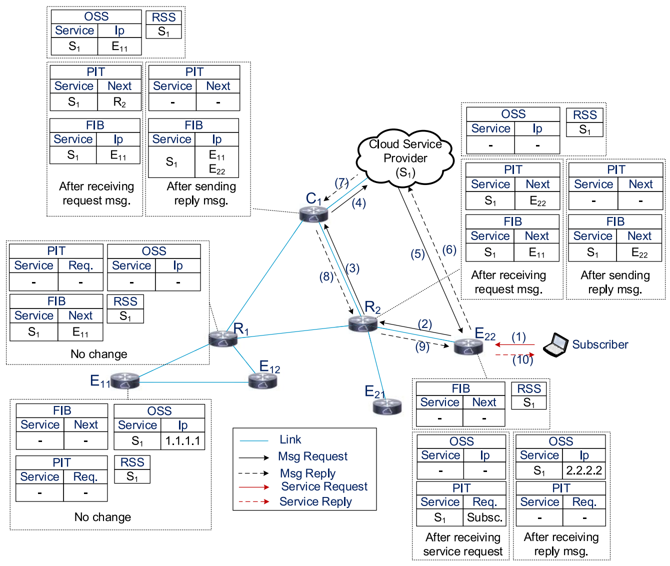
Fig. 8. Service in neighboring region not satisfying QoS.

to deploy service S_1 on (E_{12}). After step 7, C_1 updates the FIB table to show two MECs running S_1.

Through the procedure of the service discovery protocol, each node can find information about the requested service by looking for it in its revised CCN tables or by contacting upper layer nodes. By using C-SDD, since all deployed service information will be updated in both the CC and the RC, MEC can therefore easily discover a requested service in neighboring MECs. In other words, MECs can self-discover the requested service by using C-SDD.

C. Synchronization

The synchronization of service information between MEC, the RC, and the CC is also an important feature in C-SDD. Suppose a service in MEC is shutdown, and MEC needs to broadcast this information. The basic protocol is to announce

TABLE I
EXAMPLE OF RC FIB TABLE

Service	Next	Interest Node
S_1	E_{11}	E_{21}, E_{22}
S_1	E_{31}	E_{23}

the shutdown to the RC and the CC. The CC broadcasts the information to all RCs, and each RC then broadcasts to MECs in the region. This protocol may cause overhead in the network. In our implementation, we propose an additional field in the FIB table, a so-called *interest_node* field, to store all nodes that requested the service. Therefore, the RC will notify MECs that refer to the MECs in its region for the services. Like the RC, the CC stores a list of RCs in the FIB table for synchronization. Table I shows an example of the FIB table for RC R_2. When the service shuts down, MEC erases the service from the OSS table and announces the shutdown to the RC. The RC sends updates to the CC and to all MECs listed in the *interest_node* field of the FIB table. The CC similarly sends an update to all RCs listed in the *interest_node* field of the FIB table. The RCs finally send updates to the MECs in their region.

D. QoS-Based Service Discovery Decision

According to service discovery protocol, a node (MEC, RC, or CC) will return the address of target node (in FIB table if exist) for a service discovery request. However, to guarantee QoS, we need to measure the RTT from requested user to the target node and compare it with the threshold service delay. Only the address of target node, which guarantees RTT less than the threshold, will be returned to requester, otherwise, the node will continue to check RTT of other target nodes or send service discovery request to upper layer nodes. In this paper, we consider threshold time delay as a QoS parameter. If RTT of a service is greater than the threshold, it will violate the QoS requirements, and vice versa. Therefore, at every layer (MEC, RC, or CC), C-SDD needs to discover the requested service to ensure RTT satisfies QoS requirements. RTT can be affected by network delay between two nodes and computation time inside a service instance. It means that if there are too many mobile IoT devices requesting only one service instance, the computation at the service instance could suffer overhead, and QoS could be degraded. We will formulate the RTT and propose a service discovery decision algorithm to guarantee QoS in Section V.

V. SERVICE DISCOVERY DECISION MODEL

As mentioned in Section IV, we must find a service such that RTT is less than the threshold delay to guarantee QoS. The typical RTT is defined as the total time from when the mobile IoT device sends the service request until receiving the result, and can described with the following three stages [34].

1) Uploading the input.
2) Remote execution at the service instance.
3) Receiving the computation result.

TABLE II
SUMMARY OF NOTATIONS

Notation	Definition
$\mathcal{N}_e$	Set of MECs
$\mathcal{N}$	Set of nodes
$\mathcal{U}$	Set of mobile IoT devices
$\mathcal{S}$	Set of services
$r_{u,e}(t)$	Uplink data rate of mobile IoT device u at time t
$\omega_{u,e}$	Bandwidth allocated for mobile IoT device u via MEC e
p_u	Transmission power of mobile IoT device u
$G_{u,e}$	Channel gain between mobile IoT device u and MEC e
σ_u^2	Power of Gaussian noise at mobile IoT device u
$\tau_{i,j}^s(t)$	Transmission delay for uploading input data of service s from i to j at time t
$x_{m,n}^s(t)$	Service discovery decision variable at time t
I_s	Size of input data for service s
$D_{m,n}$	Network bandwidth between node m and n at time t
$T_u^s(t)$	Total transmission delay of service s at time t for mobile IoT device u
μ_s	Service rate of service s
$C_n^s(t)$	Average processing time of one request for service s at time t

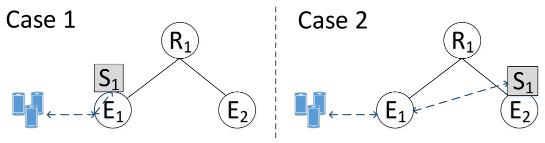

Fig. 9. Two cases of service discovery.

Because the size of the computation result is much smaller than input, stage 3 can be ignored [35]. We now construct the mathematical formula for RTT from a communications model and a computation model. The notations used in this paper are listed in Table II.

A. Communication Model

As Fig. 9 shows, there are two cases in which to deploy a service instance.

Case 1: Service instance is deployed on the connected MEC.

Case 2: The service instance is deployed in a remote node.

Therefore, we can generalize the upload time as including two parts: 1) transmission time from mobile IoT device to connected MEC and 2) transmission time from the connected MEC to the other node (in case 2).

Part 1: Transmission time from mobile IoT device to the connected MEC.

Let $\mathcal{N}$ denote the total number of nodes in the systems, including MECs, RCs, and CC. $\mathcal{N}_e$ is the set of MECs. We consider mobile IoT device $u \in \mathcal{U}$ as requesting service $s \in \mathcal{S}$ at MEC $e \in \mathcal{N}_e$. Given the number of mobile IoT devices requesting service within range of MEC e at time t, the uplink data rate of mobile IoT device u at time t is expressed as

$$r_{u,e}(t) = \omega_{u,e}\log_2\left(1 + \frac{p_u G_{u,e}}{\sigma_u^2}\right), \quad \forall u \in \mathcal{U}, e \in \mathcal{N}_e \tag{1}$$

where ω_u is the bandwidth allocated to mobile IoT device u via MEC e, p_u is the transmission power of mobile IoT device u, G_u^e is the channel gain between mobile IoT device u and connected MEC e, and σ_u^2 is the power of Gaussian noise at mobile IoT device u.

The transmission delay for uploading the input data to service s from mobile IoT device u to connected MEC e is expressed as

$$\tau_{u,e}^s(t) = \frac{I_s}{r_{u,e}(t)}, \quad \forall u \in \mathcal{U}, e \in \mathcal{N}_e, s \in \mathcal{S} \tag{2}$$

where I_s is the size of input data for service s.

Part 2: Transmission time from the connected MEC to the other node hosting the requested service instance. The transmission time between two certain nodes is given as follows:

$$\tau_{m,n}^s(t) = x_{m,n}^s(t) I_s D_{m,n}(t), \quad \forall m, n \in \mathcal{N}, s \in \mathcal{S} \tag{3}$$

where $D_{m,n}$ is the network delay between nodes m and n at time t. We assume that end-to-end latency between two certain MECs will be measured by the network administrator periodically, and $D_{n,n} = 0$.

Therefore, the total transmission delay is expressed as

$$\begin{aligned} T_{u,n}^s(t) &= \tau_{u,e}^s(t) + \tau_{e,n}^s(t) \\ &= I_s\left(\frac{1}{r_{u,e}(t)} + D_{e,n}(t)\right), \ \forall u \in \mathcal{U}, e \in \mathcal{N}_e, n \in \mathcal{N}, s \in \mathcal{S}. \end{aligned} \tag{4}$$

B. Computation Model

Since a service can serve multiple mobile IoT devices, there will be a strain on processing time if the number of service requests increases. Different services have their own delays from stretching out multiple service requests. Let μ_s denote processing time for only one request for service s. We assume that the services running in the containers are isolated from the performance of other services. Therefore, μ_s is not changed if another service is deployed in the same node. Let ε_s^k denote the performance degradation ratio due to k simultaneous requests for service s. The processing time for service s at node n will be expressed as follows:

$$C_n^s(t) = \mu_s + \varepsilon_s^k, \quad \forall n \in \mathcal{N}, s \in \mathcal{S}, k = [0, |\mathcal{U}|]. \tag{5}$$

Let $x_{m,n}^s(t)$ denote the discovery decision variable, which indicates whether or not node m routes all packets requesting service s to destination node n at time t

$$x_{m,n}^s(t) = \begin{cases} 1, & \text{Node } m \text{ routes all packages requested} \\ & \text{for service } s \text{ to node } n \text{ at time } t \\ 0, & \text{otherwise.} \end{cases} \tag{6}$$

The performance degradation ratio ε_s^k can be modeled by a general function, depending on the service type. Suppose the service follows a simple round-Robin scheduling policy; we can use a general function $\alpha_s k + \beta_s$ for the performance degradation ratio (ε_s^k), where α_s and β_s are parameters that can be tuned to suit any practical needs depending on service type [4]. Thus, (5) can be written as

$$C_n^s(t) = \mu_s + \alpha_s \sum_{u \in \mathcal{U}} x_{e_u,n}^{s,u}(t) + \beta_s, \quad \forall n \in \mathcal{N}, s \in \mathcal{S}. \tag{7}$$

C. QoS Model

Combining transmission and computation time, the RTT of mobile IoT device u requesting service s running in node n at time t can be expressed as follows:

$$RTT_{u,n}^s(t) = T_{u,n}^s(t) + C_n^s(t). \tag{8}$$

Let $q_{u,e}^s(t) \in \{0, 1\}$ denote a service request binary, which indicates whether or not mobile IoT device u requests service s at time t

$$q_u^s(t) = \begin{cases} 1, & \text{if service } s \text{ is requested from} \\ & \text{mobile IoT device } u \text{ at time } t, \\ 0, & \text{otherwise.} \end{cases} \tag{9}$$

Therefore, in order to satisfy QoS requirements, the RTT of each service must not exceed the given QoS threshold, and can be expressed as follows:

$$q_u^s(t) RTT_{u,n}^s(t) \le \zeta_s, \quad \forall n \in \mathcal{N}, s \in \mathcal{S}, u \in \mathcal{U} \tag{10}$$

where ζ_s is the given QoS threshold of service s.

D. Service Discovery Decision Model

As described in the logical process of a node, all service requests come and go through the Service Aware module to check whether the service is registered or not. If the service is not registered, all incoming packets of the service are automatically forwarded to the destination as normal packets. Otherwise, the incoming service request will go through the service discovery decision algorithm, C-SDD (as shown in Algorithm 1), to discover the location of the service. Algorithm 1 can be applied to any node (i.e., MECs, RCs, or CC) to make decisions for service discovery while guaranteeing QoS.

C-SDD first looks up the service information in the OSS and FIB tables to forward the incoming packets to the node hosting the service instance. When C-SDD is running on MECs, the FIB table only contains a unique service s, i.e., if service s within destination node f, it exists in the FIB table (f = s∩FIB), all the requests for service s will only be forwarded to destination node f. If C-SDD is running on RCs or CC, the FIB table probably contains multiple rows of service s (i.e., service s is hosted in multiple nodes), so we propose function *findBestNode* to find the best location of service s to satisfy the QoS requirements. If OSS and FIB tables have no information on the requested service, or all existing locations of service s do not satisfy QoS, the node will send a request to ask its upper layer node for the service (i.e., MEC will ask the RC; the RC will ask CC; the CC will ask the SP).

Algorithm 1 Service Discovery Decision (C-SDD) at Node

```
Input: Request for service $s$ at time $t$
Output: Service discovery rule at time $t$ ($x_e^s(t)$)
1:  Initialize $x_{e,n}^s(t) = 0, \forall n \in \mathcal{N}$
2:  if ($s \in$ OSS) then
3:      $r \leftarrow$ get node hosting service $s$ in OSS
4:      $x_{e,r}^s(t) = 1$
5:      return $x_e^s(t)$
6:  else if ($s \in$ FIB) then
7:      $f$ = findBestNode(FIB($s$))
8:      if $f > -1$ then
9:          $x_{e,f}^s(t) = 1$
10:         return $x_e^s(t)$
11:     end if
12: end if
13: $x_e^s(t) \leftarrow$discoverService()
14: return $x_e^s(t)$
Function: findBestNode
Input: FIB($s$) at current node, Network delay matrix ($D$)
  $F_s \leftarrow$ Get list of nodes providing service $s$
  min = $\zeta_s$
  $f = -1$
  for $i \in F_s$ do
    $RTT_{u,i}^s(t) \leftarrow T_{u,i}^s(t) + C_i^s(t)$
    if $RTT_{u,i}^s(t) \leq$ min then
      min = $RTT_{u,i}^s(t)$
      $f = i$
    end if
  end for
  return $f$
Function: discoverService
  if (node is CC) then
    Request Service Provider to deploy service $s$ at the
    connected MEC of mobile IoT device
  else
    Send service request to its upper-layer node
  end if
  Waiting for reply
  Update FIB and $x_e^s(t)$
  return $x_e^s(t)$
```

VI. Performance Analysis

In this section, we use probability theory to formulate the average service discovery time in our proposed model and the traditional SDN-CCN model.

A. General Node Model

As mentioned above, our proposed model is based on a three-tiered hierarchical network topology including MEC and regional and CCs corresponding with each tier. When a service request arrives, the service discovery protocol will sequentially check service information in MEC, the RC and the CC. Therefore, a service can be discovered in the following four cases.

1) *Case 0 (0, 0, 0):* No service information is stored in MECs, RCs, and the CC (i.e., no mobile IoT device has requested the service yet).
2) *Case 1 (0, 0, 1):* Service information is stored in the CC. This happens when a mobile IoT device from other RCs has already requested the service, but no mobile IoT device in the current RC has requested the service yet.
3) *Case 2 (1, 0, 1):* Service information is stored in MEC and an RC. This happens when a mobile IoT device from other MECs in the same RC has requested the service, but no mobile IoT device in the MEC has requested the service yet.
4) *Case 3 (1, 1, 1):* Service information is stored in MEC, the RC, and the CC (i.e., the service was already requested by a mobile IoT device in MEC).

According to C-SDD described above, if MEC stores the service information, the RC and the CC will store it as well. Therefore, the remaining cases have not happened. Let P_0, P_1, P_2, and P_3 denote the probability that the case 0, 1, 2, and 3, respectively, have happened. Suppose the network topology has one CC, and each RC has the same number of MECs. Let n_r be the number of RCs managed by the CC and n_e be the number of MECs managed by one RC. Let n_u denote the average number of mobile IoT devices per MEC. P_0 is the probability that no mobile IoT device in the system has requested the service yet, and it is formulated by

$$P_0 = \frac{1}{2^{n_u.n_e.n_r}}. \tag{11}$$

P_1 is the probability that no mobile IoT device in the same RC has requested the service yet, and at least one mobile IoT device in another RC has requested the service. It is formulated by

$$P_1 = \frac{2^{n_u.n_e.(n_r-1)} - 1}{2^{n_u.n_e.n_r}}. \tag{12}$$

P_2 is the probability that no mobile IoT device in MEC has requested the service yet, and at least one mobile IoT device in another MEC node from the same RC has requested the service. It is formulated by

$$P_2 = \frac{2^{n_u.(n_e-1)} - 1}{2^{n_u.n_e}}. \tag{13}$$

Then, we have

$$P_3 = 1 - P_0 - P_1 - P_2. \tag{14}$$

The average service discovery time is calculated as follows:

$$\overline{T_{sd}^{\text{3tier}}} = \sum_{i=0}^{3} T_i.P_i \tag{15}$$

where T_i is the service discovery time for case i.

B. Traditional SDN-CCN Model

The traditional SDN-CCN model consists of two tiers. One is the edge computing layer and the other is a centralized controller to provide service information (as shown in Fig. 1).

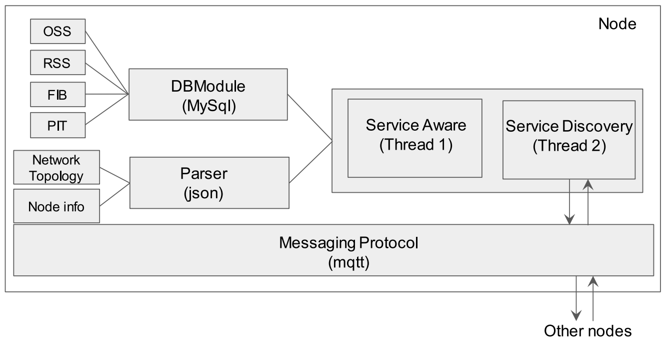

Fig. 10. Edge computing implementation.

TABLE III
EXPERIMENT PARAMETERS

Parameter	Value
Number of Base Stations	99
Number of Users	20467
Number of Services	10
Service threshold	[1,10] second
Input size	[1,10] Mb
μ	[1,10]
α	0.5
β	0
Network delay	ratio of distance
User request service	[1,10]
Start time request service	random from previous time
Service usage duration	[1,10] epoch

All service information will be checked only in the centralized controller, rather than in the RC of the above model. By comparison with this model, we can prove that our proposed model with RC always achieves better performance than the traditional SDN-CCN model. In this model, service will be discovered in the following three cases.

1) *Case 0 (0, 0):* No service information is stored in the controller and MEC. No mobile IoT device has requested the service yet.
2) *Case 1 (0, 1):* Service information is stored in the controller. This happens when a mobile IoT device from another MEC network requests the service.
3) *Case 2 (1, 1):* Service information is stored in the controller and MEC.

We can obtain the probability of each case as follows:

$$P_0 = \frac{1}{2^{n_u.n_e.n_r}} \tag{16}$$

$$P_1 = \frac{2^{n_u.(n_e.n_r-1)} - 1}{2^{n_u.n_e.n_r}} \tag{17}$$

$$P_2 = 1 - P_0 - P_1 \tag{18}$$

and the average service discovery time is calculated as follows:

$$\overline{T_{sd}^{2\text{tier}}} = \sum_{i=0}^{2} T_i.P_i. \tag{19}$$

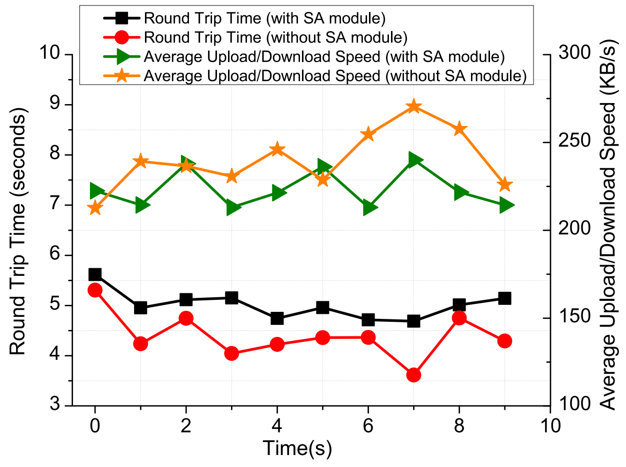

Fig. 11. Effects on service aware module.

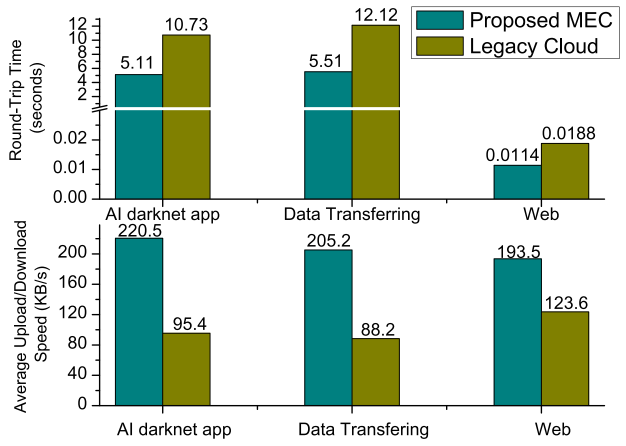

Fig. 12. Advantages of our proposed MEC compared with legacy cloud.

VII. IMPLEMENTATION AND EVALUATION

A. Implementation

To demonstrate C-SDD, we set up a scenario using two MECs and two RCs in our laboratory, one CC and one end-point cloud service on Amazon Web Services (AWS). MECs and RCs are deployed on PCs equipped with a wireless card and an Ethernet card. In particular, the MECs are configured as access points to provide a WiFi signal. The CC is deployed in a VM on AWS. Fig. 10 describes the detailed implemented modules of a node. On each node, we implement two threads for Service Aware and Service Discovery so MEC can receive requests and discover service in the neighboring MECs simultaneously. A database module, including revised CCN tables, is implemented with MySQL, and a parser module is used to get information of network topology and node information. A service discovery thread uses MQTT messaging protocol [31] to communicate with other nodes.

B. Evaluation

1) Service Aware Module: In our proposed MEC, the Service Aware module is implemented to detect edge computing service. However, this module may cause delays in MEC,

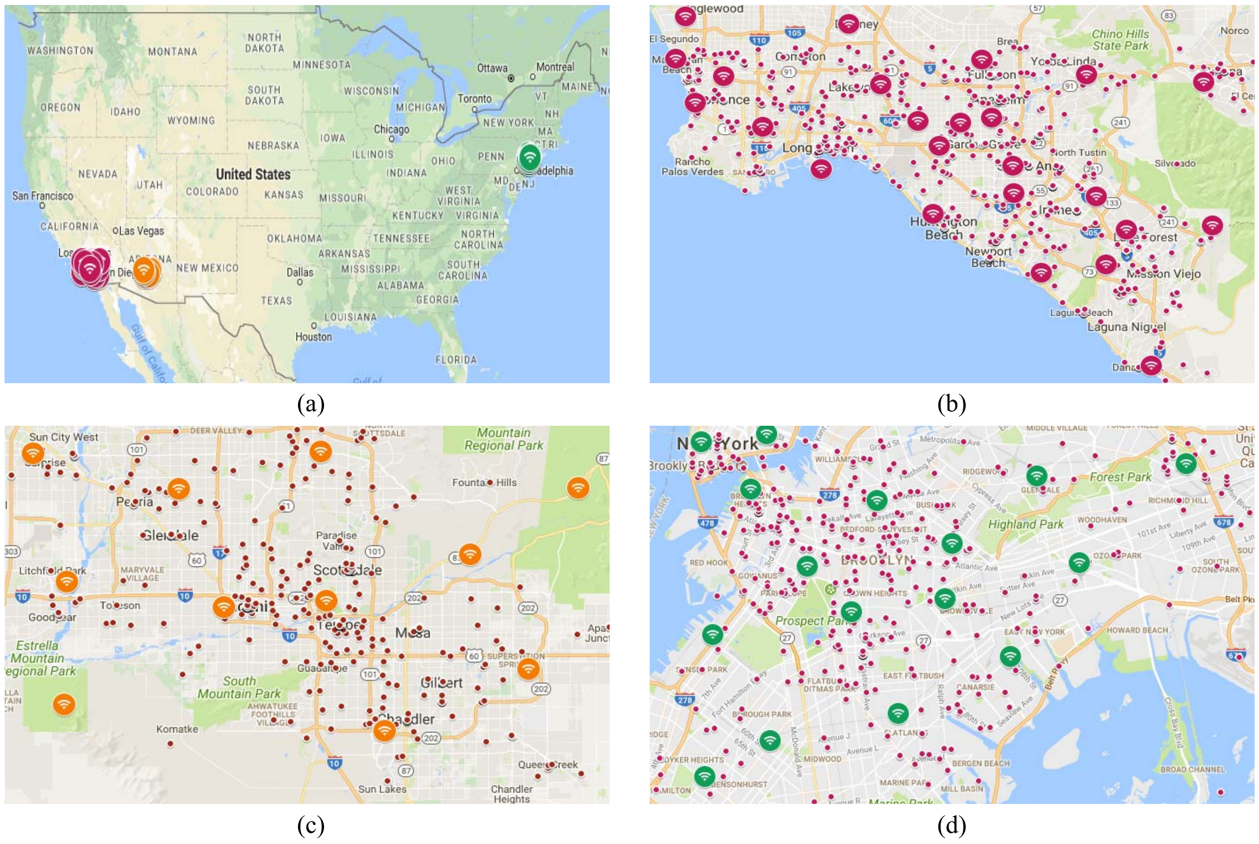

Fig. 13. Map of BSs and twitter users at three big cities: Los Angeles, Phoenix, and New York. (a) U.S. map. (b) Los Angeles. (c) Phoenix. (d) New York.

compared with the original router. In this experiment, we measure the delay time of the Service Aware module as well as the effects on average speed (Fig. 11). We use an FTP service to transfer a 1MB file from the server to the client, and vice versa. The result shows that the Service Aware module causes about a 600 ms delay per 1MB transferred (16.8 kb/s slower than average network speed). However, this delay is a trivial disadvantage, compared with the benefits of the proposed MEC compared to a legacy cloud, which will be presented in the next section.

2) Benefits of Our Proposed MEC Compared With Legacy Cloud: In this experiment, we point out that, despite including the Service Aware module, our proposed MEC still outperforms the legacy cloud. The legacy cloud is the centralized cloud in which edge computing services are located in the CC rather than edge computing. We measured performance with three applications: 1) AI Darknet; 2) a data transfer service (1 MB file); and 3) a simple Web server. AI Darknet is a client/server application that helps a mobile IoT device to detect objects, such as people, animal, or vehicle, in an image. The mobile IoT device runs the client application to send an image (around 500KB) to a server application running an edge computing service. The server application processes AI functions based on the Darknet library [36], and returns the results to the mobile IoT device. Fig. 12 shows the performance of our proposed MEC compared with the legacy cloud for response time and average network speed. Since a Web server requires less transmission time over the network, there is no big gap between our MEC and the legacy cloud. But for AI Darknet service and the data transfer service, the delay time in the legacy cloud is nearly twice as long as our proposed MEC.

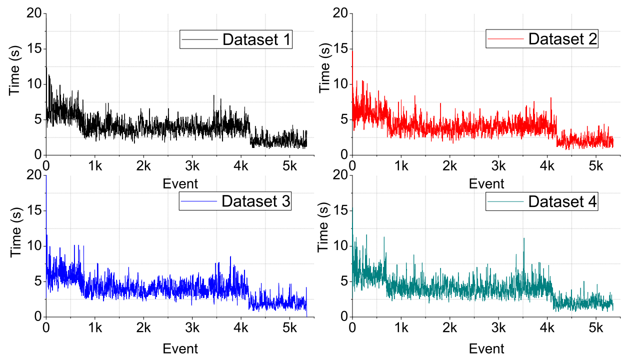

Fig. 14. Total service discovery time for each event.

3) Service Discovery Time: In this experiment, we measured service discovery time with a dataset of BS locations in three large cities in the United States: Los Angeles, Phoenix, and New York [37], as shown in Fig. 13(a). User locations are extracted from a Twitter user dataset [4]. We then filtered all Twitter users located in the above three cities. Fig. 13 presents location maps of BSs and Twitter users in those three cities. The parameters of the experiment are presented in Table III, where $[x, y]$ is randomly selected from the range x to y. For the sake of simplicity, the network bandwidth from the user

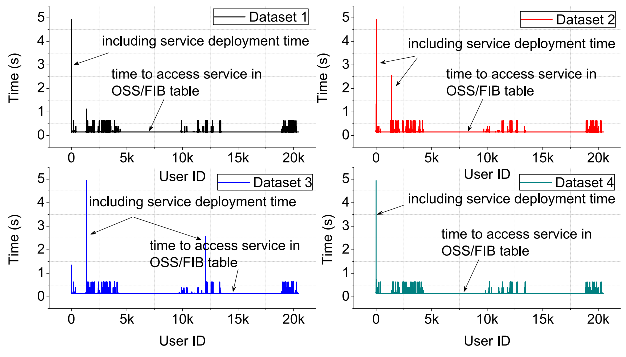

Fig. 15. Average service discovery time for each user.

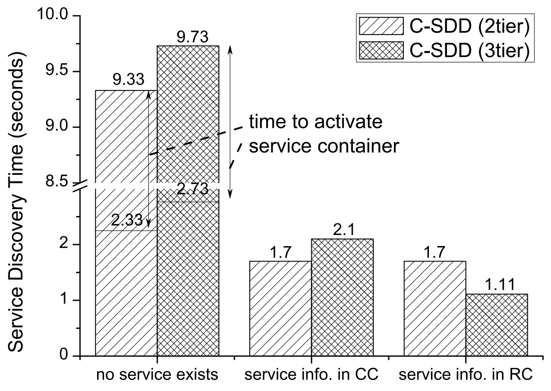

Fig. 16. Case 1: One MEC requests one service.

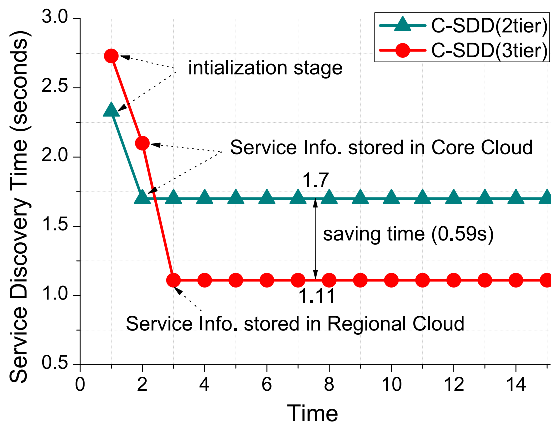

Fig. 17. Case 2: Many MECs requests one service.

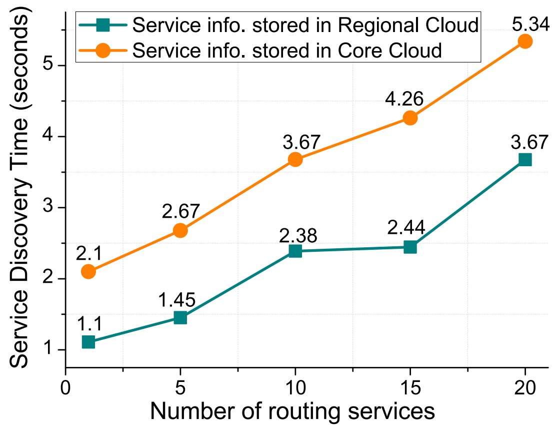

Fig. 18. Effects of the number of services on discovery time.

to the BS is allocated equally for all users connecting to the same BS.

From the above datasets, we created an event epoch, t, consisting of users requesting service in epoch t. Therefore, the service duration for a user can last one or more events. From a random process of parameters, we created four datasets of service and user requests. Fig. 14 shows the total service discovery time in the second of the four datasets. Generally, the service discovery time of our system will decrease over time, and converge to a value less than 1s. We continued to measure the average service discovery time for each user with the four datasets. Fig. 15 shows that almost all users got service information located in the OSS or the FIB tables in real time, except for some cases in which the service deployment time of the requests is included for the first request of a services.

We then measured the service discovery time by formulas in a performance analysis for two network topology model: 1) three-tiered and 2) two-tiered. Our proposed C-SDD is implemented with three-tiered model, which consists of the CC, RCs, and MEC, the so-called C-SDD (three-tiered) or simply, 3-tier. To demonstrate the centralized SDN-CCN protocol, we implemented our C-SDD on a two-tiered network topology in which the CC acts as a centralized controller to manage all service information, the so-called C-SDD (two-tiered) or simply, 2-tier. Fig. 16 presents the result of discovery time in three cases. In the first case, no service exists in MEC, an RC, or the CC, and both protocols take 7 s to activate the container in MEC. Although 3-tier takes 2.73s to communicate from MEC to SP, 2-tier takes around 2.33 s. If service information is stored in the FIB table of the CC, 3-tier takes around 2.1 s, which is worse than 2-tier (1.7 s). However, 3-tier outperforms 2-tier in the third case (service information is stored in an RC); 3-tier takes only 1.11 s, whereas 2-tier still takes 1.7 s, as shown in Fig. 17. With many MECs in the same region requesting the service, 3-tier could save lots of discovery time and overhead in the network to the CC because service information already exists in the RC. Fig. 18 shows the impact on service discovery time when increasing the number of services from 1 to 20.

With the service discovery time measured in our implementation (as shown in Fig. 16), we apply (15) and (19) to obtain the average service discovery time. Fig. 19 presents the average discovery time when increasing the number of users in each edge. We measured on four cases with different pairs for the number of RCs, n_r, and the number of MECs from each RC (n_e). The result shows that when either the number of users from MEC or the number of RCs and MECs increases, the probability of service information being stored in MEC will increase. It means that the average service discovery time will converge to the case when service information is stored in MEC. We further compared average service time of the three-tiered and two-tiered models. Fig. 20 shows that 3-tier always takes less time for service discovery than 2-tier. Therefore,

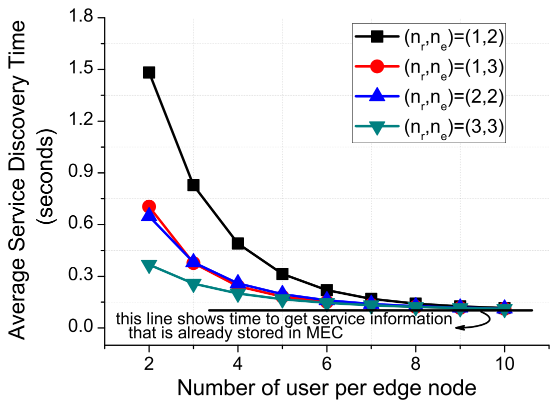

Fig. 19. Average service discovery time.

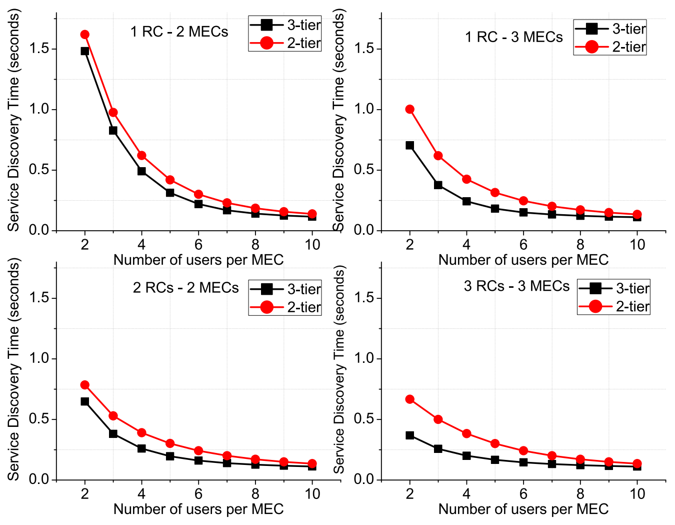

Fig. 20. Comparison of service discovery time between two-tiered and three-tiered models.

C-SDD deployed in the model with three tiers (CC, RC, and MEC) outperforms the centralized SDN-CCN protocol.

VIII. Conclusion

In this paper, we presented a decentralized and revised CCN-based MEC service deployment/discovery protocol and platform. We describe in detail the C-SDD with three stages: 1) the initial stage; 2) service discovery in the same region; and 3) service discovery in different regions. We then provide a formula for RTT to guarantee that C-SDD always satisfies the required QoS. In our experiment, we showed that our C-SDD outperforms legacy cloud with respect to network bandwidth and RTT. We also analyzed and measured the average service discovery time of C-SDD and showed the advantages of the three-tiered model, compared to a two-tiered model. For future research, our proposed platform can be implemented as a basis for building smart MECs that provide more AI features for more and more enhanced convenience in mobile IoT devices.

References

[1] H. T. Dinh, C. Lee, D. Niyato, and P. Wang, "A survey of mobile cloud computing: Architecture, applications, and approaches," *Wireless Commun. Mobile Comput.*, vol. 13, no. 18, pp. 1587–1611, Nov. 2011.

[2] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, "Mobile edge computing: A survey," *IEEE Internet Things J.*, vol. 5, no. 1, pp. 450–465, Feb. 2018.

[3] N. Wang, B. Varghese, M. Matthaiou, and D. S. Nikolopoulos, "ENORM: A framework for edge node resource management," *IEEE Trans. Services Comput.*, vol. PP, no. 99, 2018.

[4] L. Wang, L. Jiao, T. He, J. Li, and M. Muhlhauser, "Service entity placement for social virtual reality applications in edge computing," in *Proc. IEEE INFOCOM*, Honolulu, HI, USA, Apr. 2018, pp. 1–8.

[5] J. Wu, Q. Zhao, N. Yang, and J. Duan, "Augmented reality multi-view video scheduling under vehicle-pedestrian situations," in *Proc. Int. Conf. Connected Veh. Expo (ICCVE)*, Shenzhen, China, Oct. 2015, pp. 163–168.

[6] M. Nekovee, "Radio technologies for spectrum above 6 GHz—A key component of 5G," in *Proc. 5G Radio Technol. Seminar Explor. Tech. Challenges Emerg. 5G Ecosyst.*, London, U.K., Mar. 2015, pp. 1–46.

[7] M. Zhou, R. Zhang, D. Zeng, and W. Qian, "Services in the cloud computing era: A survey," in *Proc. 4th Int. Univ. Commun. Symp.*, Beijing, China, Oct. 2010, pp. 40–46.

[8] D. Kreutz *et al.*, "Software-defined networking: A comprehensive survey," *Proc. IEEE*, vol. 103, no. 1, pp. 14–76, Jan. 2015.

[9] V. Jacobson *et al.*, "Networking named content," in *Proc. 5th Int. Conf. Emerg. Netw. Exp. Technol.*, Rome, Italy, Dec. 2009, pp. 1–12.

[10] M. Satyanarayanan *et al.*, "The role of cloudlets in hostile environments," *IEEE Pervasive Comput.*, vol. 12, no. 4, pp. 40–49, Oct. 2013.

[11] Y. Li, N. T. Anh, A. S. Nooh, K. Ra, and M. Jo, "Dynamic mobile cloudlet clustering for fog computing," in *Proc. IEEE ICEIC 17th Int. Conf. Electron. Inf. Commun.*, Honolulu, HI, USA, Jan. 2018, pp. 1–4.

[12] D. Satria, D. Park, and M. Jo, "Recovery for overloaded mobile edge computing," *Future Gener. Comput. Syst.*, vol. 70, pp. 138–147, May 2017.

[13] D. Pizzolli *et al.*, "Cloud4IoT: A heterogeneous, distributed and autonomic cloud platform for the IoT," in *Proc. IEEE Int. Conf. Cloud Comput. Technol. Sci. (CloudCom)*, Luxembourg City, Luxembourg, Dec. 2016, pp. 476–479.

[14] A. Brogi and S. Forti, "QoS-aware deployment of IoT applications through the fog," *IEEE Internet Things J.*, vol. 4, no. 5, pp. 1185–1192, Oct. 2017.

[15] A. C. Baktir, A. Ozgovde, and C. Ersoy, "How can edge computing benefit from software-defined networking: A survey, use cases, and future directions," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 4, pp. 2359–2391, Jun. 2017.

[16] N. McKeown *et al.*, "OpenFlow: Enabling innovation in campus networks," *Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Apr. 2008.

[17] J. Oueis, E. C. Strinati, and S. Barbarossa, "The fog balancing: Load distribution for small cell cloud computing," in *Proc. IEEE 81st Veh. Technol. Conf. (VTC Spring)*, Glasgow, U.K., May 2015, pp. 1–6.

[18] S. Sathyanarayana and M. Moh, "Joint route-server load balancing in software defined networks using ant colony optimization," in *Proc. Int. Conf. High Perform. Comput. Simulat. (HPCS)*, Innsbruck, Austria, Jul. 2016, pp. 156–163.

[19] W.-C. Lin, C.-H. Liao, K.-T. Kuo, and C. H. P. Wen, "Flow-and-VM migration for optimizing throughput and energy in SDN-based cloud datacenter," in *Proc. IEEE 5th Int. Conf. Cloud Comput. Technol. Sci.*, Bristol, U.K., Dec. 2013, pp. 206–211.

[20] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, and B. Ohlman, "A survey of information-centric networking," *IEEE Commun. Mag.*, vol. 50, no. 7, pp. 26–36, Jul. 2012.

[21] T. Koponen *et al.*, "A data-oriented (and beyond) network architecture," in *Proc. Conf. Appl. Technol. Archit. Protocols Comput. Commun.*, Kyoto, Japan, Aug. 2007, pp. 181–192.

[22] M. Ain, "Architecture definition, component descriptions, and requirements, deliverable D-2.3," PSIRP 7th FP EU-Funded Project, Rep., Feb. 2009. [Online]. Available: http://www.psirp.org/files/Deliverables/FP7-INFSO-ICT-216173-PSIRPD2.3_ArchitectureDefinition.pdf

[23] B. Ahlgren, "Second NetInf architecture description, D-6.2," 4WARD EU FP7 Project, Rep., Apr. 2010. [Online]. Available: http://www.4ward-project.eu/

[24] E. Aubry, T. Silverston, and I. Chrisment, "SRSC: SDN-based routing scheme for CCN," in *Proc. 1st IEEE Conf. Netw. Softw. (NetSoft)*, London, U.K., Apr. 2015, pp. 1–5.

[25] X. N. Nguyen, D. Saucez, and T. Turletti, “Providing CCN functionalities over OpenFlow switches,” INRIA, Rocquencourt, France, Res. Rep. hal-00920554, Aug. 2013. [Online]. Available: https://hal.inria.fr/hal-00920554

[26] J. Son, D. Kim, H. S. Kang, and C. S. Hong, “Forwarding strategy on SDN-based content centric network for efficient content delivery,” in *Proc. Int. Conf. Inf. Netw. (ICOIN)*, Kota Kinabalu, Malaysia, Jan. 2016, pp. 220–225.

[27] J. Pan, L. Ma, R. Ravindran, and P. TalebiFard, “HomeCloud: An edge cloud framework and testbed for new application delivery,” in *Proc. 23rd Int. Conf. Telecommun. (ICT)*, Thessaloniki, Greece, May 2016, pp. 1–6.

[28] R. Ravindran, X. Liu, A. Chakraborti, X. Zhang, and G. Wang, “Towards software defined ICN based edge-cloud services,” in *Proc. IEEE 2nd Int. Conf. Cloud Netw. (CloudNet)*, San Francisco, CA, USA, Nov. 2013, pp. 227–235.

[29] P. TalebiFard *et al.*, “An information centric networking approach towards contextualized edge service,” in *Proc. 12th Annu. IEEE Consum. Commun. Netw. Conf. (CCNC)*, Las Vegas, NV, USA, Jan. 2015, pp. 250–255.

[30] A. Gopalasingham, D. G. Herculea, C. S. Chen, and L. Roullet, “Virtualization of radio access network by virtual machine and docker: Practice and performance analysis,” in *Proc. IFIP/IEEE Symp. Integr. Netw. Service Manag. (IM)*, Lisbon, Portugal, May 2017, pp. 680–685.

[31] *MQTT Protocol*. Accessed: May 2018. [Online]. Available: http://mqtt.org/

[32] *AMQP Protocol*. Accessed: May 2018. [Online]. Available: https://www.amqp.org/

[33] *Stomp Protocol*. Accessed: May 2018. [Online]. Available: https://stomp.github.io/

[34] X. Lyu *et al.*, “Selective offloading in mobile edge computing for the green Internet of Things,” *IEEE Netw.*, vol. 32, no. 1, pp. 54–60, Jan. 2018.

[35] X. Chen, L. Jiao, W. Li, and X. Fu, “Efficient multi-user computation offloading for mobile-edge cloud computing,” *IEEE/ACM Trans. Netw.*, vol. 24, no. 5, pp. 2795–2808, Oct. 2016.

[36] J. Redmon. *DarkNet: Open Source Neural Networks in C*. Accessed: May 2018. [Online]. Available: http://pjreddie.com/darknet/

[37] *The OpenCellID Database*. Accessed: May 2018. [Online]. Available: https://www.opencellid.org

Tien-Dung Nguyen (M’18) received the B.Eng. degree in computer engineering from the Ho Chi Minh University of Technology, Ho Chi Minh City, Vietnam, in 2008, and the Ph.D. degree from the Computer Engineering Department, Kyung Hee University, Seoul, South Korea, in 2014.

He is currently with Kyung Hee University as Research Professor with the Department of Computer Science and Engineering. His current research interests include cloud computing, Internet of Things, future Internet, distributed real-time system, and mobile computing.

Eui-Nam Huh (M’18) received the B.S. degree from Busan National University, Busan, South Korea, the master’s degree in computer science from the University of Texas, Austin, TX, USA, in 1995, and the Ph.D. degree from Ohio University, Athens, OH, USA, in 2002.

He is currently with Kyung Hee University, Seoul, South Korea, as a Professor with the Department of Computer Science and Engineering. His current research interests include cloud computing, Internet of Things, distributed real time system, mobile computing, and big data and security.

Dr. Huh has served many community services for ICCSA, WPDRTS/IPDPS, APAN Sensor Network Group, ICUIMC (IMCOM), ICONI, APIC-IST, ICUFN, SoICT as various types of chairs. He is the Vice-Chairman of Cloud/Bigdata Special Technical Group of TTA, and an Editor of ITU-T SG13 Q17&Q18.

Minho Jo (SM’16) received the B.A. degree from the Department of Industrial Engineering, Chosun University, Gwangju, South Korea, in 1984 and the Ph.D. degree from the Department of Industrial and Systems Engineering, Lehigh University, Bethlehem, PA, USA, in 1994.

He is currently a Professor with the Department of Computer Convergence Software, Korea University, Sejong, South Korea. He is one of the founders of the Samsung Electronics LCD Division, Seoul, South Korea. He has published over 100 publications in reputed journals/magazines and international conferences. His current research interests include Internet of Things (IoT), LTE-unlicensed, artificial intelligence and big data in the IoT, blockchain and security, HetNets, cloud computing, wireless energy harvesting, autonomous cars, and optimization and probability in networks.

Dr. Jo was a recipient of the 2018 IET Best Paper Premium Award and the Headong Outstanding Scholar Prize in 2011. He was a Vice President of the Institute of Electronics of the Korea Information Processing Society. He is the Founder and the Editor-in-Chief of *KSII Transactions on Internet and Information Systems* (http://itiis.org. SCI and Scopus indexed). He is currently the Vice President of the Korean Society for Internet Information. He is currently an Editor for the IEEE WIRELESS COMMUNICATIONS, an Associate Editor of the IEEE ACCESS, the IEEE INTERNET OF THINGS JOURNAL, *Security and Communication Networks*, and *Wireless Communications and Mobile Computing*.