Master's Thesis

Title

Implementation of M2M applications for demonstrating benefits of ICN architecture

Supervisor

Professor Masayuki Murata

Author

Yuhao Gao

February 8th, 2019

Department of Information Networking

Graduate School of Information Science and Technology

Osaka University

Keywords

ICN (Information-centric Networking)

General purpose framework

M2M (machine-to-machine) communication

Service Migration and Execution Management

NDN (Named Data Networking)

Contents

1 Introduction **5**

2 Related work **7**

2.1 Information-centric networking 7

2.2 Machine-to-machine communication 8

2.3 Service migration and execution management 9

3 ICN-based and IP-based design of operations and ICN benefits for the development **10**

3.1 Application logic of M2M communication applications 10

3.1.1 Abstraction and requirements 10

3.1.2 ICN-based design 16

3.1.3 IP-based design 20

3.1.4 Benefits of ICN compared to IP networks 24

3.2 Service migration and execution management 27

3.2.1 The abstraction and requirements 27

3.2.2 ICN-based design 30

3.2.3 IP-based design 35

3.2.4 Benefits of ICN compared to IP networks 37

4 Proof of concept in a specific scenario **38**

4.1 Scenario overview and requirements 38

4.2 ICN-based design 42

4.3 IP-based design 44

4.4 Experiment 44

4.5 Benefits of ICN compared to IP networks 47

5 Conclusion and future work **51**

Acknowledgements **52**

References **53**

Appendix **53**

A. Implementation of the average value retrieval **53**

B. Implementation of the device control based on the instruction in the guiding system **65**

List of Figures

1 Abstraction of general M2M communication applications 13
2 ICN-based implementation of the service location advertisemen and the service requesting in the M2M communication application 19
3 ICN-based implementation of the service execution and responding in the M2M communication application 20
4 IP-based implementation of the service advertisement and the service requesting in the M2M communication application 22
5 IP-based implementation of the service execution and responding in the M2M communication application 23
6 Abstraction of general service migration and execution management operations 28
7 ICN-based implementation of the service requesting in the service migration and execution management 34
8 IP-based implementation of the service requesting in the service migration and execution management 36
9 Guiding system overview 39
10 Guiding system node behavior overview 41
11 PoC scenario node behavior overview 42
12 ICN-based design of the PoC scenario communication 44
13 IP-based design of the PoC scenario communication 44
14 Robot behaviors in the experiment 47
15 The overview of the average value retrieval communication 53
16 The diagram of the average value retrieval communication in ICN over IP . 53
17 The diagram of the average value retrieval communication in IP 54

List of Tables

1 Details of service requests and the responder behavior in each service category 15

2 Service identities of each service category in M2M communication applications 17

3 Application design differences in ICN and IP 25

4 Details of service requests and the responder behavior of each service category 30

5 Service identities of each service category in the management operation . . 32

6 Name space in the ICN-based design of the PoC scenario 43

7 Software used in the experiment 45

8 Amount of executable instructions in the ICN and the IP program 48

1 Introduction

Information-centric networking (ICN) is a new generation network concept aiming to solve problems altogether in terms of the robustness, the efficiency, the development cost and so on. Compared with IP-based architectures, ICN-based architectures primitively embedded several useful facilities including the name used as the invocation mechanism of services, the in-network caching, the programmable router and so on, which enhances the system performance and makes developers comprehend and solve the problems easily.

Currently, various ICN-based frameworks and ICN benefits for users and developers in specific scenarios have been proposed. ICN is originally designed to realize the efficient content distribution in many-to-many communications, and recent research imply that ICN may have benefits in distributed computing such as the mobile edge computing. Because ICN nodes are programmable and are able to retrieve and to execute services not only in servers but also in routers, which enables the efficient processing. Researchers have found that ICN has benefits in certain specific application scenarios such as controlling Internet of things (IoT) lights, collecting sensor data and processing the data, and intermittent communications using movable routers. ICN developers intend to develop applications in various scenarios rather than previously discussed scenarios, but ICN benefits in general scenarios have not been discussed. If developers know that ICN-based architectures have benefits in certain general scenarios, various applications of general scenarios have opportunities to be discussed and realized in real-world.

In this thesis, we propose benefits of using the ICN-based architecture for the development of general applications and operations compared with the IP-based architecture. Since arbitrary application with operations well-adapted to ICN and arbitrary network optimization operations are able to be implemented in ICN layer and to show that ICN is able to realize easy implementation compared with IP, we discuss the development of general M2M communication applications and the service migration and execution management. We show the benefits by comparing the IP-based and the ICN-based design of those operations. We also present the implementation of specific scenario APIs to confirm the benefits. The specific application scenario is controlling IoT guidance devices such as panels and robots to guide people to reach the destination. Through comparing the IP

and the ICN design of the specific application APIs, we confirm that the ICN benefit is that ICN is able to realize the easier implementation of the application logic of general M2M communications, the service migration and execution management, the caching, and the publish/subscribe communication than IP.

This thesis is organized as follows. We compare this work with related works including the ICN, M2M communication, and the service migration and execution management in Section 2. We show the design and implementation method of M2M application in ICN and IP, and show the ICN benefits in the application implementation in Section 3. We confirm the feasibility of the method by designing and implementing a specific application with the method, and show the ICN benefits in the specific application implementation in Section 4. We summarize this thesis and show the future work in Section 5.

2 Related work

2.1 Information-centric networking

ICN is a new generation network concept aiming to solve problems altogether in efficiency, robustness, development cost and so on. ICN standardly embeds the content-based packet awareness, programmability, publish-subscribe, caching mechanism in the network layer.

In recent years, in terms of specific application scenarios, many architectures and protocols have been proposed and the ICN benefit in those scenarios for service users has been shown. In the scenario of retrieving real-time contents for Intelligent Transportation System and a drone connecting to the internet retrieving sensor data of geographical information, the ICN is able to communicate efficiently by using the primitive publish-subscribe mechanism [**?**, **?**]. In the scenario of retrieving data from distributed databases, ICN is able to reduce the waste service request when the service becomes unavailable and communicate efficiently by using the primitive service availability management mechanism [**?**]. In the scenario of the home IoT device cooperation such as controlling the light based on the sensor data, ICN is able to realize the easy maintenance of the service location without using IP addresses when the developer deploys, resets, and changes the device hardware [**?**]. In the scenario of communications between multiple fragmented networks without the interactive connection with a movable router to switch and forward packets, ICN is able to reduce the moving distance of the movable router and communicate efficiently and robustly by using the primitive caching and the service availability mechanism [**?**, **?**, **?**].

The research about architectures and protocols for optimizing the network resource such as the CPU resource and the bandwidth has been conducted recently. Since the application function is able to be treated as the content and to be migrated to other nodes, the architecture for migrating services and managing the CPU resource has been proposed [**?**, **?**]. By using the ICN primitive publish/subscribe mechanism, ICN is able to enhance the performance of communications between mobile nodes. Various mobility support techniques in ICN including routing-based, mapping-based, trace-based, data spot, and data depot methods have been proposed [**?**, **?**]. The research about the name scheme and the caching protocol in various scenarios have been conducted [**?**, **?**].

In this work, we show the ICN benefit in the application implementation. Since ICN is able to realize the application function in the network layer, ICN can be a general purpose framework in the development of various application domains by embedding the function able to be reused in other domains into the ICN framework. The easy implementation of applications is able to be achieved by assembling the network layer service instead of scratching low-level services.

2.2 Machine-to-machine communication

M2M communications mean communications between machines able to connect to other machines. Massive devices are considered to join the network and massive data will be transmitted. The major use cases include the security and the public safety, smart grid, tracking and tracing, vehicular telematics, payment, healthcare, and remote control.

The vision and the requirement of M2M communication have been indicated [**?**]. The vision is to realize the compute-rich/high-performance hardware, the ultra scalable connectivity, and the cloud-based mass device management and services. The requirement includes the enhancement of transmission, security, operation and resource management. The major architecture for machine type communications includes multiple layers: the device layer, the network layer, and the application layer [**?**]. The device layer is responsible for providing low-level primitive service such as data generation, device control, and data processing. The network layer is responsible for optimizing the resource and the operation. The application layer is responsible for realizing the high-level business logic. Various algorithm and protocols enhancing the efficiency and the scalability of the resource and the operation have been proposed [**?**, **?**].

In this work, we show the M2M communication abstraction including the major M2M communication which is an instance of communications with operations which are well-adapted to ICN and are able to show the ICN benefits in the application implementation. We consider the major communication in M2M communications includes the distributed machine cooperation which is one of the distributed computing and is well-adapted to ICN.

2.3 Service migration and execution management

Service migration and execution management means migrating service functions to the optimal node and instructing the function execution based on the measurement information for the optimization.

Service migration and execution management is considered as one instance of the network optimization operation focusing on the resource allocation and the operation. Since a service function can be considered as a content able to be delivered, the publish-subscribe mechanism is able to be applied to the actual communication for managing service requests and service responses [**?**]. Since the decision-making algorithm of the management should satisfy the requirement of specific scenarios such as the migrated function should keep up with the user mobility, the computing resource usage and so on, various problem formulations and strategies and protocols have been proposed. The migration management operation can be considered as a sequential decision making problem and is able to be formulated with the Markov decision process technique [**?**, **?**].

In this work, we show the operation abstraction including the major migration and execution management operations which is an instance of communications with general purpose operations able to be used in various domains and is able to show the ICN benefits in the application implementation.

3 ICN-based and IP-based design of operations and ICN benefits for the development

In this section, our purpose is to show the design of general operations in certain application domains and ICN benefits for the application development. We describe the ICN-based and the IP-based design of the M2M communication application and the service migration and execution management which are instances of application operations well-adapted to ICN and network optimization operations. The network optimization operation means the operation optimising resources in the network (e.g., computing and transmission resources) and the node processing. Next, we show the ICN benefits for the application development.

Modules of operations well-adapted to ICN and general purpose operations including the network optimization operation is able to be standardly embedded in ICN. Application developers can use the modules in the application development to cut down the development time, which is considered as the ICN benefits for the development. Any instance of these operations can show the ICN benefits. The M2M communication and the service migration and execution management are instances of these operations and are able to make full use of the ICN potential in the distributed cooperation and computing which is considered as one of the important applicable domain in ICN.

In this work, we use terms of the requester and the responder representing nodes related to services.

3.1 Application logic of M2M communication applications

3.1.1 Abstraction and requirements

The service system is a system that when users inform servers to provide them services needed in a scenario, servers accomplish several steps of processes needed in the service logic and provide the services to users.

The system is supposed to include following roles of nodes. A node in the network is able to have multiple roles. Servers are supposed to have a part of roles including application user interface (AUI), application decision-maker (ADM), data processor (DP),

and content provider (CP).

- User: a trigger of services.
- AUI: provide user the access to applications and the application service in application-specific format.
- ADM: make decisions and request services provided by other nodes and send service responses based on the application logic.
- DP: process content data.
- CP: generate and provide contents for the user's retrieval, the data processing, and the decision-making.

Regarding to system requirements, users request application services in a certain frequency, and servers are supposed to give users services in a certain frequency. The entire service logic can be divided into several parts, and each part of the logic can be included in a function. By servers executing functions, servers are able to achieve the service logic and then provide services to users.

In the function execution, functions are supposed to be executed in the scenario-specific order. It means that if a function has been executed, the next function is supposed to be called and be executed. The server finishing the execution is supposed to notify the server with the next function to execute the function. Some functions need input data for processing. The data is produced by servers and devices called responders. Servers with the functions are supposed to send requests to responders for retrieving the input data.

To satisfy the requirements, nodes are supposed to include functions and behaviors shown in the following itemization and Fig.1. An arbitrary application we propose in this section as a general application is able to be realized by including multiple following nodes and combining multiple following node behaviors.

- Users: send contents and content requests and service requests to AUI in a certain frequency.
- AUI: An AUI keeps functions including updating the state of itself when receive interface control commands. An AUI is able to send signal to other AUIs to invoke

the function for updating the state of other AUIs. An AUI is able to send service requests to ADM and to send content requests to CP and to send contents to DP.

- ADM: An ADM keeps functions for the decision-making. An ADM is able to send contents and signals to other ADMs to invoke the function for the further decision-making. An ADM is able to respond to service requests from AUIs with interface control commands. An ADM is able to send content requests to CP and to send contents to DP. The push communications of pull behaviors are able to be conducted.

- CP: A CP is able to send contents and content requests to other CPs for the aggregation. A CP is able to respond to content requests with contents. The push communications of pull behaviors are able to be conducted.

- DP: A DP keeps functions for the data processing. A DP is able to send contents and signals to other DPs to invoke the function for the further processing. A DP is able to respond to contents with processed contents.

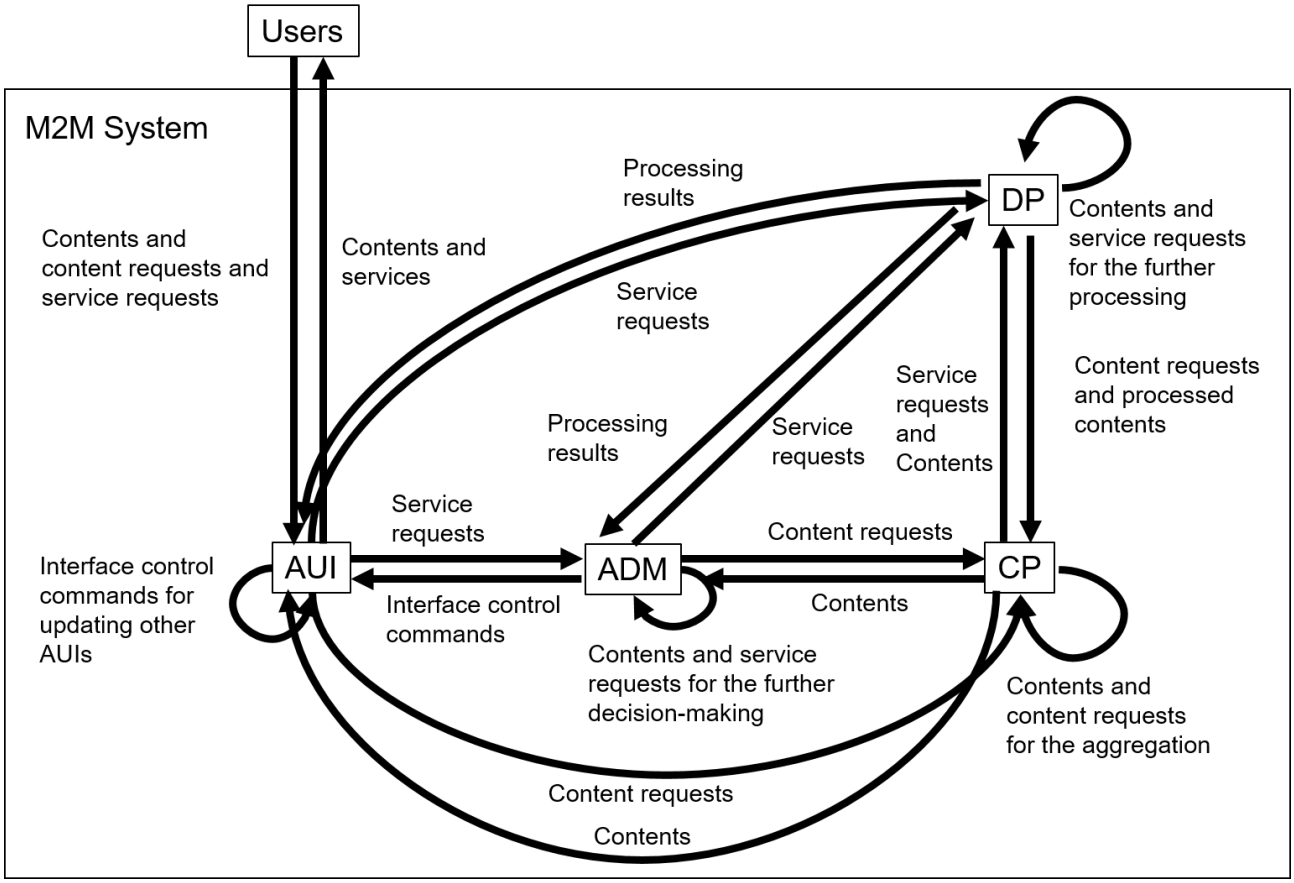

Figure 1: Abstraction of general M2M communication applications

Communications in Fig.1 are able to be classified into pull communications and push communications. Since the research of the push communication general design is still in progress in the ICN research and the push communication's aim is able to be realized in the pull communication, we only discuss the pull communication requirement and the design and the implementation in this work.

The pull communications in Fig.1 are able to be considered as instances of the service request operation that requesters request the responders to execute service behaviors and to provide service responses. The services are able to be classified into 4 categories including the decision-making, the content retrieval, the data processing, the AUI control. Any node in Fig.1 can be a requester. A responder can be an ADM or a CP or a DP or an AUI.

As for operation requirements, when a requester requests services, the requester needs

the service identity information, and the service location information or the location information of a responder keeping the service location information. If a node with a cache of the requested service decides to provide the service, the node becomes a responder. A requester keeps the identity information of services requester intends to request since the application developer knows what service the requester is supposed to request in an application scenario. Therefore, only the service location information is needed to be provided by responders. The following itemization show requirements of the service requesting.

1. A responder is supposed to advertise the service location information to let a requester know the service location.

2. A requester is supposed to send service requests to the location. The details of service requests in each service category is shown in Table 1.

3. A responder receiving a request of services is supposed to execute functions corresponding to requested services and to return the service response to the requester. The responder behaviors in each service category are shown in Table 1.

Table 1: Details of service requests and the responder behavior in each service category

Service category	Details of the service request	responder behavior
Decision-making	A decision-making request for the high-level API execution in the ADM	The responder is an ADM. The ADM is supposed to execute the high-level API kept in the ADM and return the API response, which conducts actions including sending service requests to other nodes and forwarding service responses from other nodes to the requester.
Content retrieval	A content request including identities of requesting contents	The responder is a CP. The CP is supposed to lookup requested contents at the buffer or generate contents, and then return contents to the requester.
Data processing	A processing request including the identities of processing functions	The responder is a DP. The DP is supposed to send content requests to retrieve input data and to process the data and to return the processed data to a CP and to return the processing result (ACK/-NACK).
AUI control	A control request including AUI control commands	The responder is an AUI. The service request includes AUI control commands. The AUI is supposed to update its state based on the commands and to return the control result (ACK/NACK).

3.1.2 ICN-based design

As for the ICN application implementation principle, in order to cut down the time and resources for application implementation works, the operations well-adapted to ICN (e.g., distributed computing, in-network processing, and service mobility) or able to be reused in other application domains are supposed to be implemented in the network layer by service developers. The application layer program is supposed to invoke network layer services to realize the application and is implemented by application developers.

Fig.2, 3 show node behaviors and node communications of the service requesting operation.

Service location advertisement

A responder is supposed to advertise the service location information. The service advertisement is able to be realized by the responder sending the advertisement Interest including service identities and the requester updating service identities and incoming interfaces in the FIB after receiving the Interest. The routers receiving the advertisement Interest will also update the routing of advertised services by updating the FIB.

As for the implementation, the codes of sending the advertisement Interest can be in the application layer or in the network layer since the operation can be invoked by the application layer function (e.g., initialization function) or the network layer function (e.g., scenario-specific function). Updating the FIB of the requester receiving the advertisement Interest is an ICN primitive operation.

Table 2: Service identities of each service category in M2M communication applications

Service category	Service identity
Decision-making	Include the high-level API identity and API parameters. Name example: ServiceIdentity = /{APIIdentity}/{APIParameter}* /Guidance/Bob
Content retrieval	Include requesting content identities. Name example: ServiceIdentity = /{ContentIdentity} /apple.jpg
Data processing	Include the processing API identity and API parameters. Name example: ServiceIdentity = /{APIIdentity}/{APIParameter}* /zip/apple.jpg
AUI control	Include control API identity and API parameters. Name example: ServiceIdentity = /{APIIdentity}/{APIParameter}* /TurnOnLight/KitchenLight

Sending service requests

A requester keeps the service identity information since the application developer knows what service the requester is supposed to request in a scenario. The codes of sending the request can be in the application layer or in the network layer since the operation can be invoked by the application layer function (e.g., initialization function) or the network layer function (e.g., scenario-specific function). In the requester application layer program, by the requester sending an Interest whose name includes the service identity, the service location lookup can be done in the network layer and the Interest will be sent to the next hop since the ICN framework primitively manages the service location information in the network layer.

Executing services and responding to requests

When an responder receives service requests, the responder should respond to requests to execute requirement behaviors mentioned in Table 1. Each service category design is shown in the following itemization. The codes of executing services and responding to requests are supposed to be in the network layer since the network computing resource is able to be used for more efficient processing than the processing only in the endpoint.

- Decision-making: the responder is supposed to execute the high-level API kept in the responder and return the API response.
- Content retrieval: the responder is supposed to generate the requested content based on the request name and to return the content.
- Data processing: the responder is supposed to execute the processing API kept in the responder and return the processing response (e.g., ACK/NACK, the state of the responder).
- AUI control: the responder is supposed to execute the control API kept in the responder and return the control response (e.g., ACK/NACK, the state of the responder).

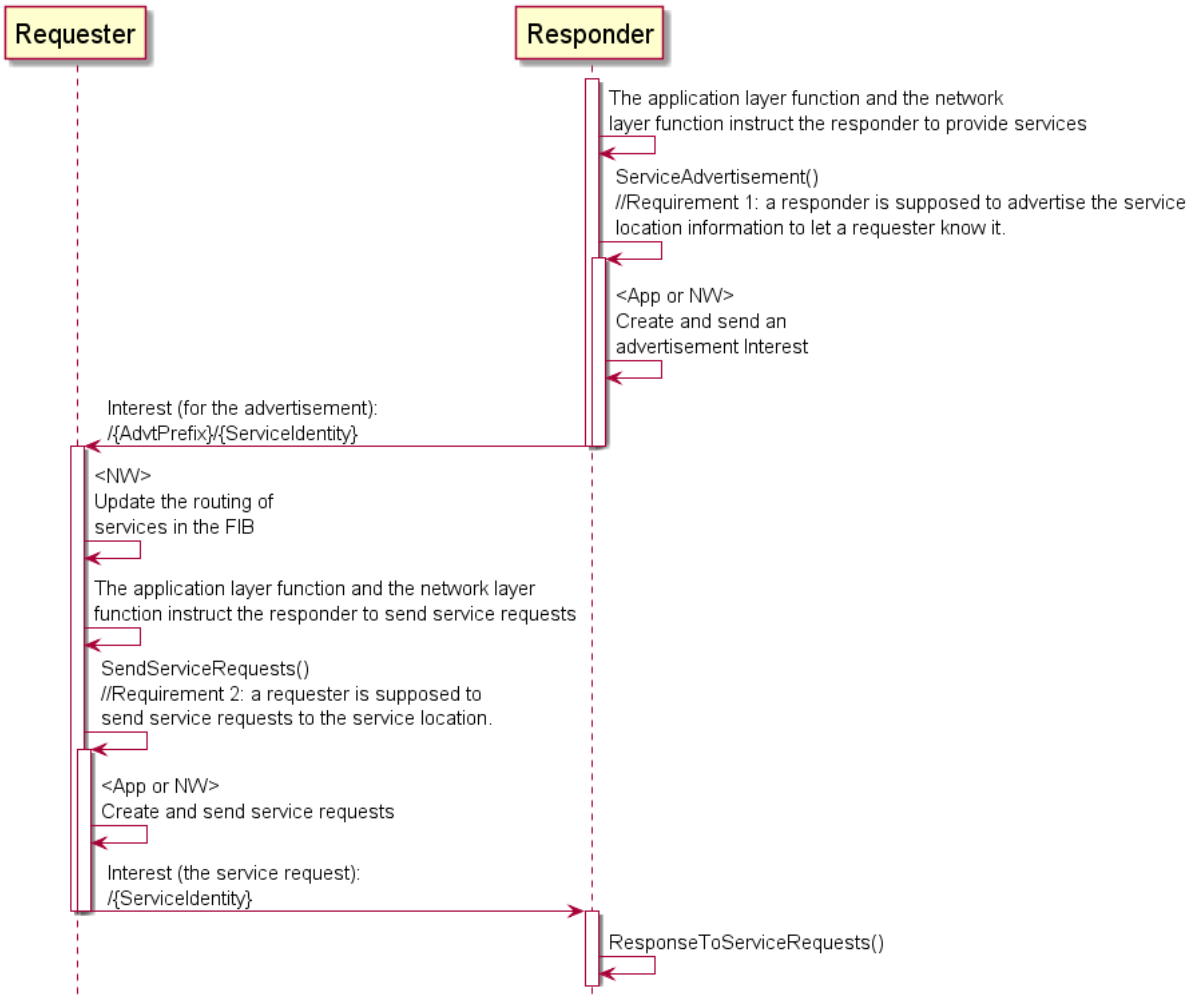

Figure 2: ICN-based implementation of the service location advertisemen and the service requesting in the M2M communication application

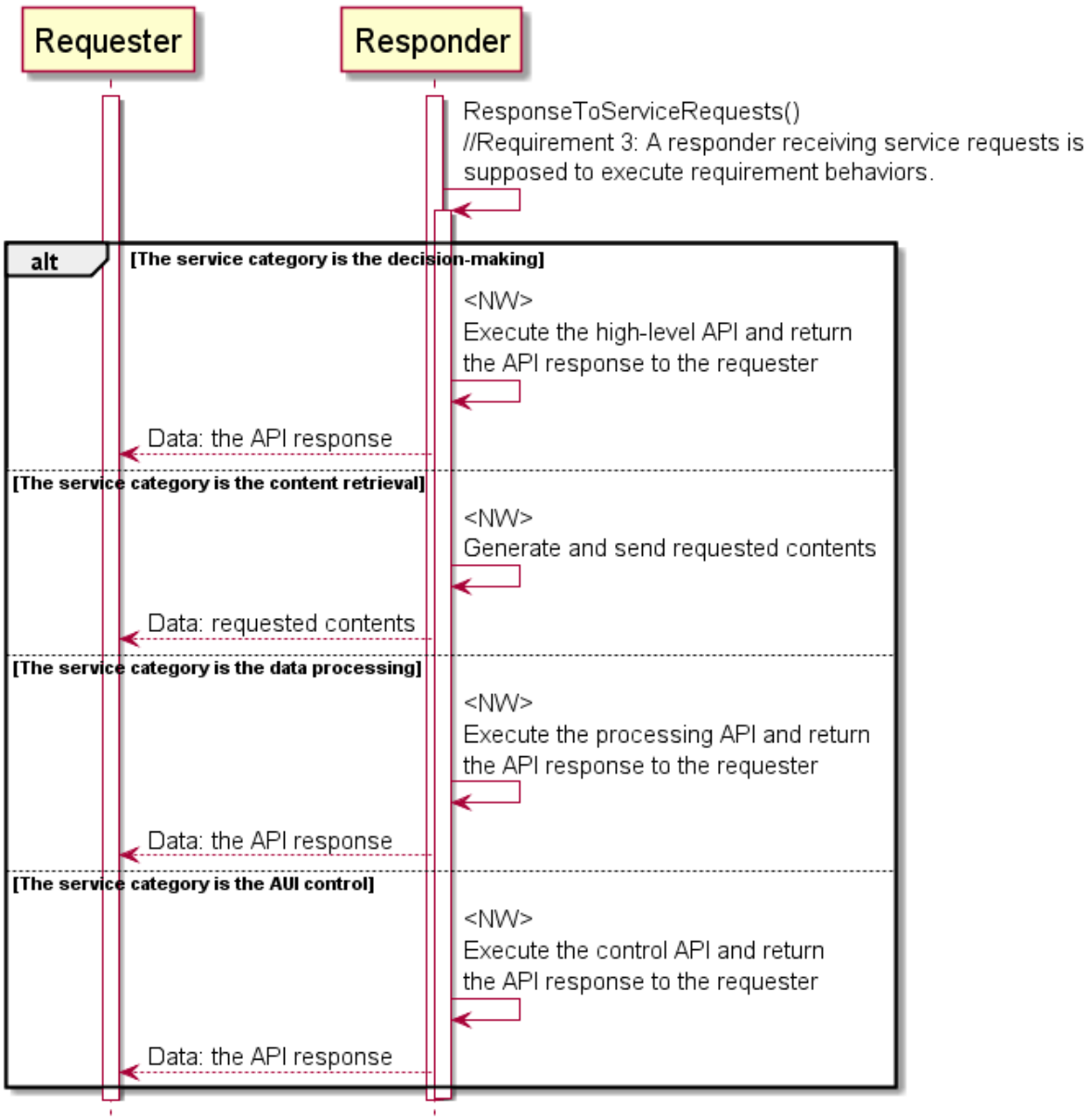

Figure 3: ICN-based implementation of the service execution and responding in the M2M communication application

3.1.3 IP-based design

All application operations are implemented in the application layer by application developers. Since application layer services used as assembling components in the application de-

velopment are general purpose service able to be used in multiple application domains, the service development depends on requirements from the multiple application domains, and this development is independent with a specific application development project. Therefore, the application developer comes to implement all application operations.

Service location advertisement

A responder is supposed to advertise the service location information. The service location advertisement is able to be realized by the responder sending the advertisement packet including service identities and service location information (IP address), and the requester managing the service information including service identities and the service location information.

Sending service requests

A requester keeps the service identity and the service location information since the application developer knows what service the requester is supposed to request in a scenario. . In the requester application program, the sending packet operation is able to be realized by the requester specifying the service location and sending a packet containing the service identity.

Executing services and responding to requests

When a responder receives service requests, the responder should respond to requests to execute requirement behaviors mentioned in Table 1. Each service category design is shown in the following itemization.

- Decision-making: the responder is supposed to execute the high-level API kept in the responder and return the API response.
- Content retrieval: the responder is supposed to generate the requested content based on the request name and to return the content.
- Data processing: the responder is supposed to execute the processing API kept in the responder and return the processing response (e.g., ACK/NACK, the state of the responder).

- AUI control: the responder is supposed to execute the control API kept in the responder and return the control response (e.g., ACK/NACK, the state of the responder).

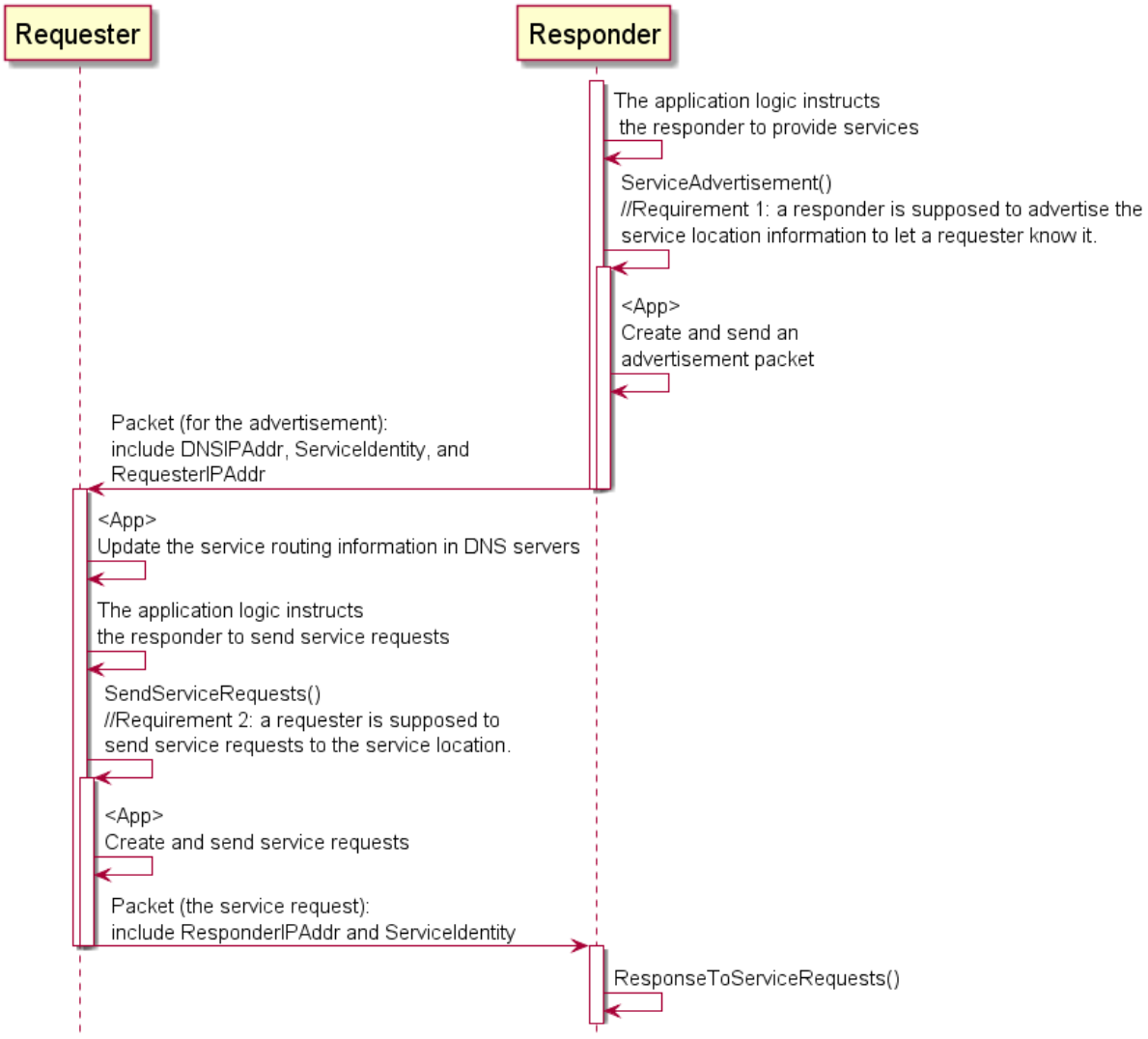

Figure 4: IP-based implementation of the service advertisement and the service requesting in the M2M communication application

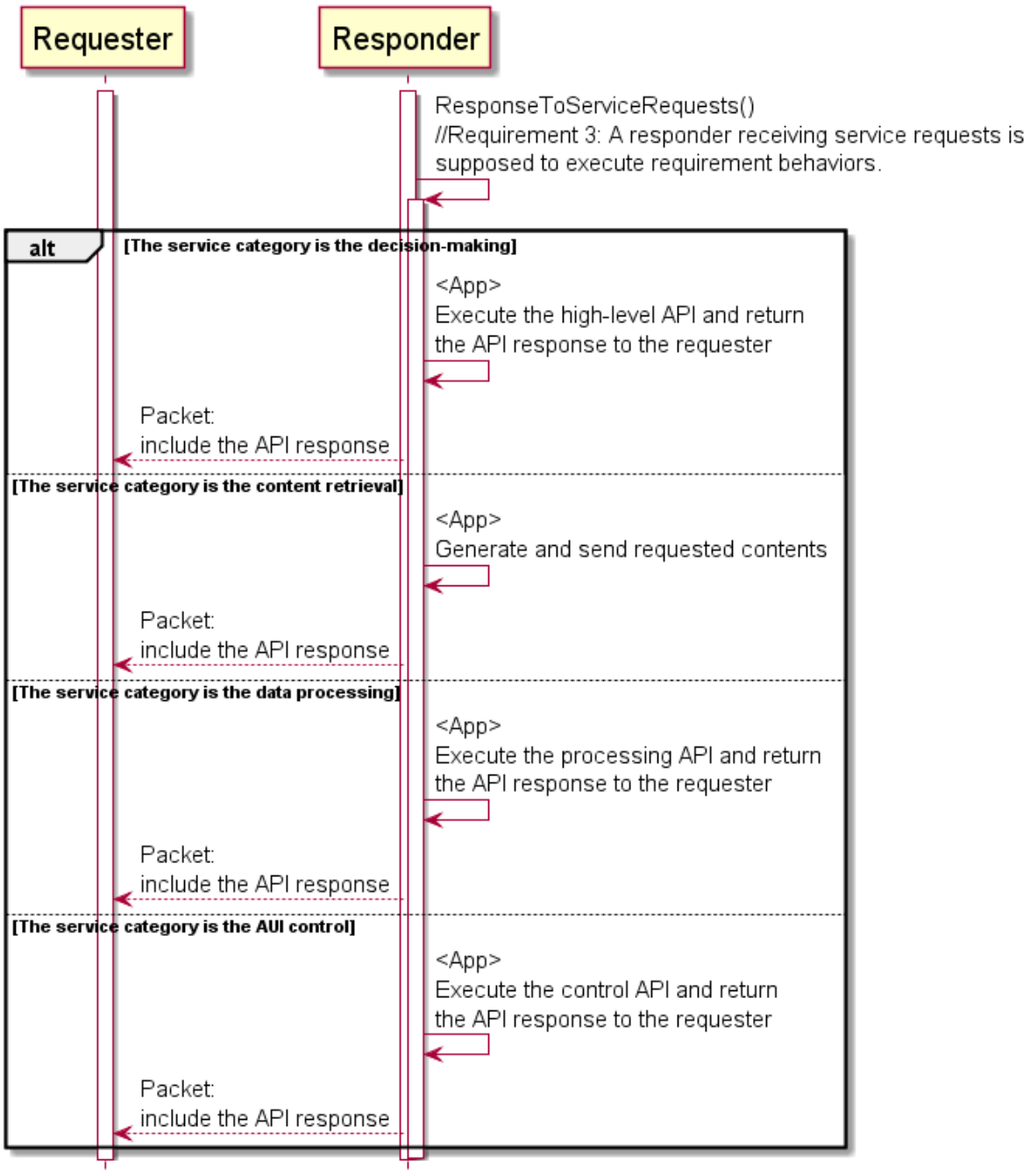

Figure 5: IP-based implementation of the service execution and responding in the M2M communication application

3.1.4 Benefits of ICN compared to IP networks

Application logic realized with primitive named services in the network layer

In the ICN-based design, the operations well-adapted to ICN (e.g., distributed computing, in-network processing, and service mobility) or able to be reused in other application domains are supposed to be implemented in the network layer by service developers. The application layer program is supposed to invoke network layer services to realize the application and is implemented by application developers.

In the IP-based design, application developers are generally responsible for implementing all application operations in the application layer.

This difference simplifies the implementation work of application developers.

Table 3: Application design differences in ICN and IP

Layer and development responsibility	ICN codes	IP codes
High-level application logic: · Application project ·Application developers	Combination of network layer services	Combination of general purpose services
Primitive component: · Standardization · Network layer developers · Library/framework/-platform developers	Operations well-adapted to ICN: · Distributed computing · Machine cooperation Network optimization operations in ICN layer: · Caching ·Service migration and execution management · Mobility support General purpose services in ICN layer: ·Components for communication including the interface, the routing	General purpose services: ·Components for communication including the interface, the routing

We confirm this benefit by showing the application layer implementation of the average value retrieval and the guiding system in the IP and the ICN over IP architecture. The average value retrieval implementation is shown in the appendix of this thesis and the guiding system implementation is shown in section 4.

Built-In caching function in individual ICN nodes

ICN nodes have the primitive caching module in the network layer. Since the module is embedded in the network layer forwarding pipeline, Interests and Data are able to be cached based on the default caching policy and there needs no implementation work of building the caching module.

IP nodes and the IP-based framework generally do not have the primitive caching module. If the caching module is required in a application, the module is supposed to be implemented in the application layer.

ICN does not need the implementation and therefore ICN-based frameworks simplify the implementation.

Efficient pub/sub implementation

The Pub/Sub communication is that the service publish information from responders and service subscribe information from requesters are able to be stored in pub/sub management modules. Requesters and responders are able to receive the information when they access the management module. By using the pub/sub management module, the time of sending and receiving the information is able to be separate, and the sender does not need to know the location of receivers. The pub/sub management module is able to deal with communications where publishing and subscribing are dynamic.

In IP-based design, the pub/sub module is not primitively provided and is supposed to be implemented in the application layer program of a proxy server or requesters and responders.

In ICN-based design, the pub/sub module is primitively provided in the network layer (specifically, the PIT) of requesters and responders and there does not need the application layer implementation.

Therefore, the ICN-based design simplified the implementation of pub/sub modules.

Unified service invocation mechanism

The service invocation mechanism means the protocol and the description and the service discovery mechanism for invoking a service.

In IP, the invocation mechanism is usually different with the mechanism of other application domains. For example, a part of Web sites use FTP while another site uses HTTP for transfering a file. The method for retrieving a file is different in GitHub and RPM repositories. A developer can not request a service from a repository that the developer does not know.

In ICN, all service is invoked by using the Interest and the Data packet. For requesting a service, the invocation protocol is to only send Interest packets. The name resolution of service identities will be done in the network layer, and a requester is able to request a service without knowing and inputing the service location.

The unified invocation mechanism cut down the time for developers learning an invocation mechanism in other application domains, which cuts down the development costs.

3.2 Service migration and execution management

3.2.1 The abstraction and requirements

The service migration means that a copy of a service agent moves from the migration source to the migration destination for the load balance optimization of the computing and the transmission. There needs a decision-maker which collects the information for the migration (e.g., states of each node computing resources, service demand information from clients and servers) from content providers and make decisions and instruct the migration destination and the migration source to accomplish the migration. The availability of the original service agent and the copy can be extinguished or provided based on the configuration of the scenario-specific optimal policy.

The general migration and execution management operation is supposed to involve roles shown in the following itemization.

- User: a trigger of the migration and execution instruction and provide the configuration to the migration operation.
- Management decision-maker (MDM): provide the user the access to the service of the service migration and execution instruction. Collect the information for the migration from content providers, and make migration and execution decisions based

on the optimal policy and the configuration, and instruct the migration destination and the migration source to accomplish the migration, and instruct service executors to execute requested services.

- Content provider (CP): provide the information for the migration.
- Migration destination (MD): request a service agent from the migration source. After retrieving the agent, the service can be advertised based on the optimal policy.
- Migration source (MS): provide a copy of a service agent to the migration destination. After sending the agent, the original service agent can be extinguished based on the optimal policy.
- Service executor (SE): keep service agents, and when receiving service execution requests, execute the service logic and respond with the service response.

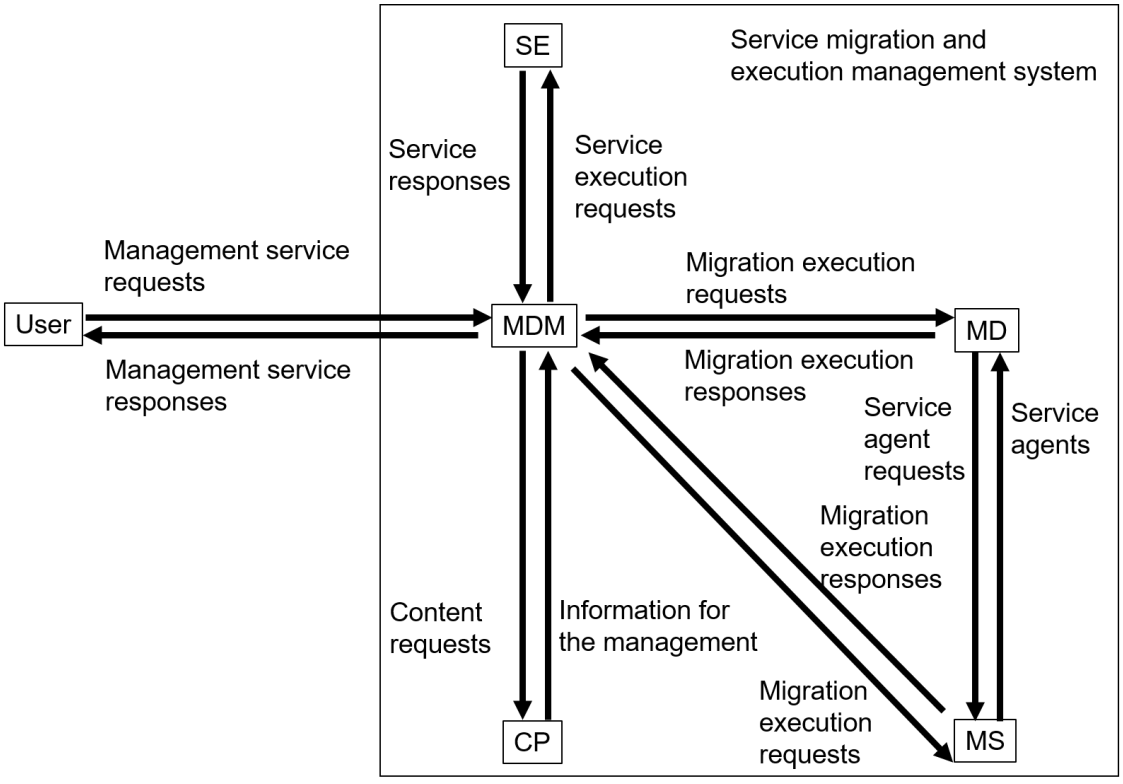

Figure 6: Abstraction of general service migration and execution management operations

Major interactions in the management operation is shown in Fig. 6. An arbitrary management service is able to be realized by containing all kinds of operations shown in the following itemization and combining the operations in an arbitrary order.

- Management decision-making: an user invokes the management service by sending the configuration to the MDM and invokes the management service and receive the response.
- Content retrieval: MDM sends content requests to CPs and receives the information for the management.
- Service migration: MDM instructs the service migration by sending migration instruction to an MD, and the MD executes the migration by sending service agent requests to an MS based on the migration instruction of MDM.
- Service execution management: MDM instructs the service execution by sending service execution requests to SEs and receiving the service response from SEs.

All operations in the above itemization are able to be considered as instances of the service requesting operation. Services in the management are able to be classified into four categories of operations including the management decision-making, the content retrieval, the service migration, and the service execution management. In the service requesting operations, there are two types of nodes: requesters and responders. The requester can have any type of roles in Fig. 6. The responder can be the MDM or the CP or the MD or the MS or the SE.

Requirements of the service requesting operation is shown in Sec. 3.1.1. The difference of requirements between the M2M communication application and the management is that the responder receiving service requests executes corresponding behaviors of the management decision-making, content retrieval, the service migration, the service execute management. Table 1 shows the details of service requests and the responder behavior of each service category.

Table 4: Details of service requests and the responder behavior of each service category

Service category	Details of the service request	responder behavior
Management decision-making	A decision-making request for the high-level API execution in the DM	The responder is an MDM. The MDM is supposed to execute the high-level API and return the API response, which conducts behaviors including the content retrieval, the service migration, and the service execution management.
Content retrieval	A content request including identities of requesting contents	The responder is a CP. The CP is supposed to lookup requested contents at the buffer or generate contents, and then return contents to the requester.
Service migration	A migration execution request including identities of the MD and the MS and the migrated service	The responder is an MD. The MD is supposed to respond to the migration request with the response (e.g., ACK/NACK) and to send service agent requests to the MS. The MS receiving requests is supposed to return the service agent.
Service execution management	A execution request including SE identities and execution instructions	The responder is an SE. The SE is supposed to execute the service and to respond to the request with the ACK/-NACK response.

3.2.2 ICN-based design

The management service belongs to the network optimization operation. The network optimization operation is able to be applied to various application domain and it is the general purpose operation. In order to cut down the development time and to avoid

application developers to implement the same operation every time in different projects, modules for general purpose operations should be embedded in ICN layer. In this case, the ICN role is a framework for the application development.

According to the requirement, the service requesting operation needs operations including advertising the service, sending service requests, and responding to requests. The design differences between designs of the M2M communication application and the management service is that the service category and the service identity and the responder behavior in the management service are different with those of M2M communications. The service identity of each service category is shown in Table 5. The design of advertising services and sending service requests is the same as the design of those operations in the M2M communication application. The design of responding to service requests is shown in the following itemization and Fig. 7.

Table 5: Service identities of each service category in the management operation

Service category	Service identity
Management decision-making	Include the high-level API identity and API parameters. Name example: ServiceIdentity = /{APIIdentity}/{APIParameter}* /Migration/ServiceA/NodeA/NodeB
Content retrieval	Include requesting content identities. Name example: ServiceIdentity = /{ContentIdentity} /NodeA/CPUState
Service migration	Include the identity of the migrating service, the MD, and the MS. Name example: ServiceIdentity = /{MDIdentity}/migration/ {MigratingServiceIdentity}/{MSIdentity} /NodeA/migration/CalcAverageValue/NodeB
Service execution management	Include the execution instruction and the identity of the SE and the service identity. Name example: ServiceIdentity = /{SEIdentity}/ExecManagement/ {ServiceIdentity}/{ExecutionInstruction} /NodeA/ExecManagement/GenerateCameraData/Stop

- Management decision-making: the responder is supposed to execute the high-level API kept in the responder and return the API response.
- Content retrieval: the responder is supposed to generate the requested content based on the request name and to return the content.
- Service migration: the responder (MD) is supposed to return the migration response (e.g., ACK/NACK) and to send service agent requests to the MS based on the migration instruction from the responder. The MS receiving service agent requests

is supposed to send the agent to the MD.

- Service execution management: the responder is supposed to execute the requested service and return the management response (e.g., ACK/NACK, the state of the responder).

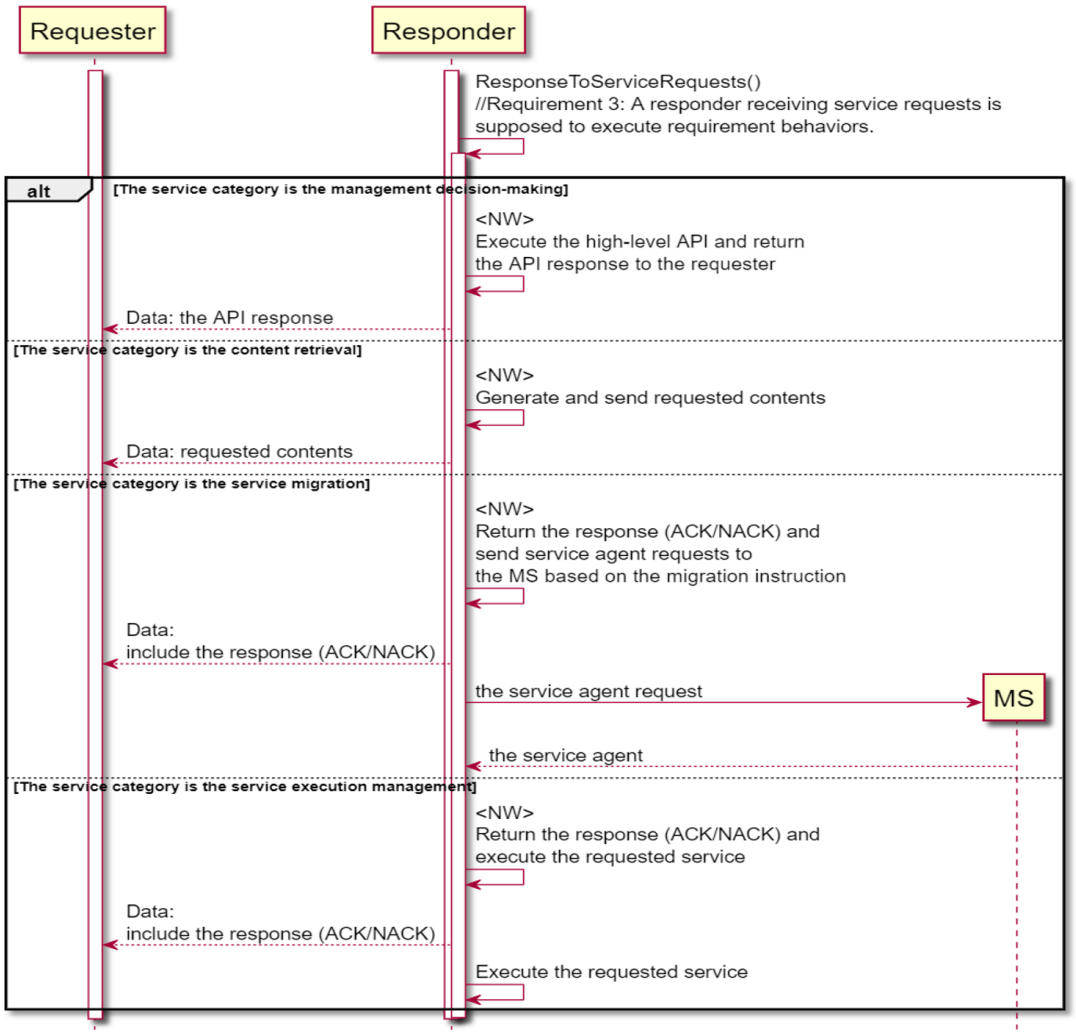

Figure 7: ICN-based implementation of the service requesting in the service migration and execution management

3.2.3 IP-based design

The management service is supposed to be implemented in the application layer by application developers. Since there are no IP-based general purpose framework, the management service can not be primitively provided in a framework and the application developers have to implement the management service.

The design differences between designs of the M2M communication application and the management service is that the service category and the service identity and the responder behavior in the management service are different with those of M2M communications. In the service requesting operation, the design of advertising services and sending service requests is the same as the the design of the M2M communication application. The design of responding to service requests is shown in the following itemization and Fig. 8.

- Management decision-making: the responder is supposed to execute the high-level API kept in the responder and return the API response.
- Content retrieval: the responder is supposed to generate the requested content based on the request name and to return the content.
- Service migration: the responder (MD) is supposed to return the migration response (e.g., ACK/NACK) and to send service agent requests to the MS based on the migration instruction from the responder. The MS receiving service agent requests is supposed to send the agent to the MD.
- Service execution management: the responder is supposed to execute the requested service and return the management response (e.g., ACK/NACK, the state of the responder).

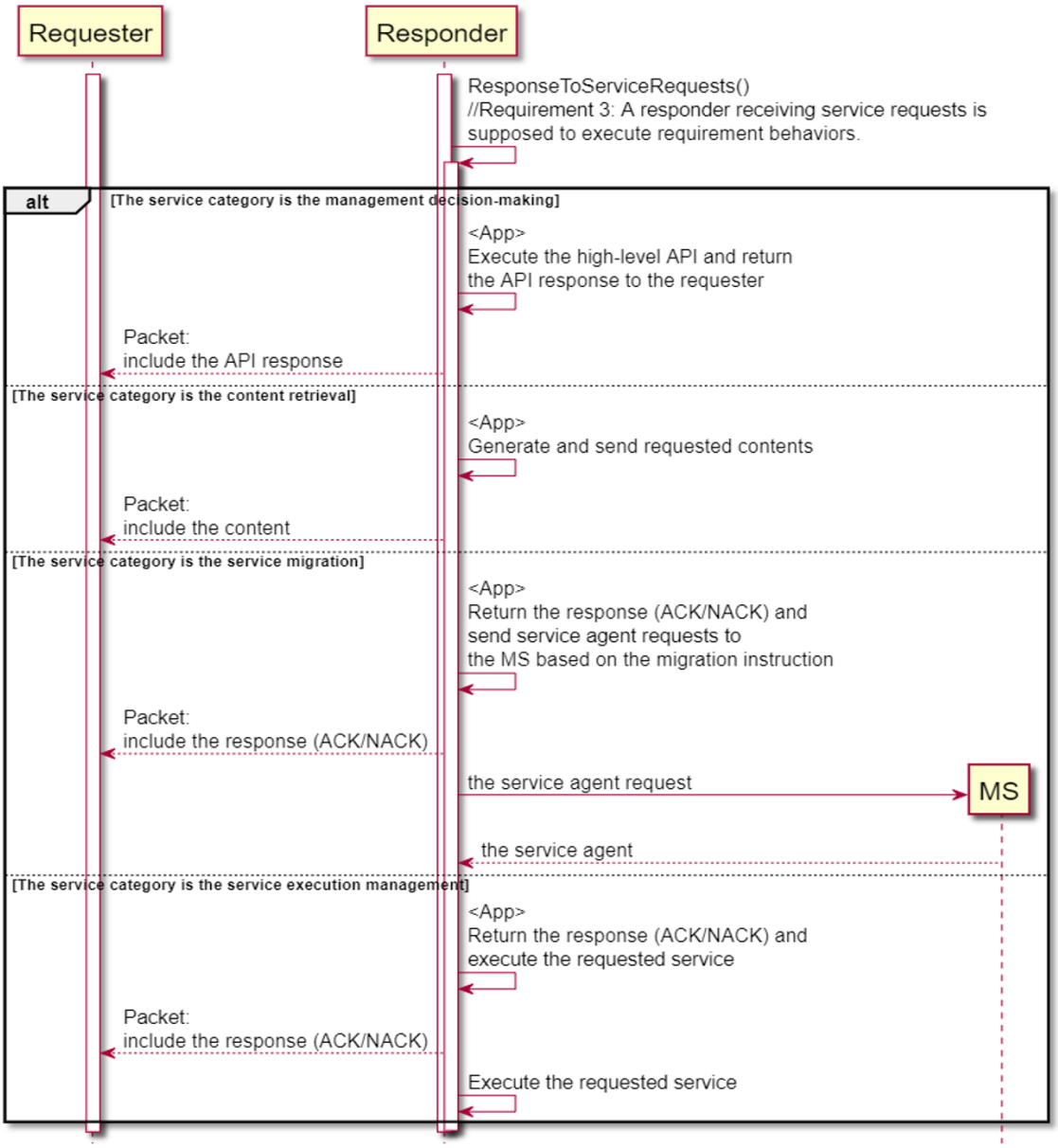

Figure 8: IP-based implementation of the service requesting in the service migration and execution management

3.2.4 Benefits of ICN compared to IP networks

ICN as a general purpose framework with modules of the network optimization operation

The service migration and execution management mentioned in Sec. 3.2 is one instance of the network optimization operation. If modules of the network optimization operation are provided in a framework for application development, application developers do not need to develop the modules, which reduces the software size in the application layer and simplifies the application development.

In IP, there does not have a general purpose framework for various application domains and application developers of the domains have to develop modules of the network optimization operation.

While ICN is able to be used as a general purpose framework with modules of the network optimization operation, application developers do not need to develop the modules, which reduces the software size in the application layer and simplifies the application development.

4 Proof of concept in a specific scenario

In this section, our purpose is shown in the following itemization.

- show a practical application functionality is able to be realized with the design mentioned in Sec. 3.
- confirm the ICN benefits mentioned in Sec. 3 by comparing the ICN-based and the IP-based design of the guiding system.

The reason we choose the guiding system as the practical application is that the guiding system is one instance of M2M communication applications and needs the service migration and execution management for the optimization when users increase. The application is able to show the ICN benefits.

In addition, the design of another specific scenario which calculates the average value of sensor data and is able to show the ICN benefits is shown in the appendix. We have not conducted the experiment to confirm the feasibility of the design but the we have confirmed the feasibility of the design in the guiding system scenario.

In this section, we describe the scenario overview and requirements and the ICN-based and the IP-based design of the guiding system.

4.1 Scenario overview and requirements

The guiding system is a system controlling IoT guidance devices including panels and robots to guide users to the movable or the stationary target. This system is able to be used in scenarios involving the guidance to movable devices and places in the city or the tourist spot or the indoor environment. Users can be the the tourist visitor, the student and the stuff attending to an event and so on. The merit is that the guidance with robots is able to provide more accurate guidance with the less misleading compared the existing service such as Google Map in some situation.

In the network, all nodes in the edge network except the global are ICN nodes. The nodes include the user's and the target's mobile terminal, routers, and guidance devices. The node role is shown in the following itemization. The router and the device is deployed

near the physical path which the user passes and they can provide the guidance service to the user connecting to the router.

- User: a trigger of the guidance service and provide the user location information to the system.
- Target: the destination of the guidance and provide the target location information to the system.
- Router: A router has multiple roles including calculating the physical path between the user and the target and the guidance area that the physical path passes and the optimal device control instruction, and controlling the device to provide the service.
- Guidance device: be controlled by the router to provide the guidance service.

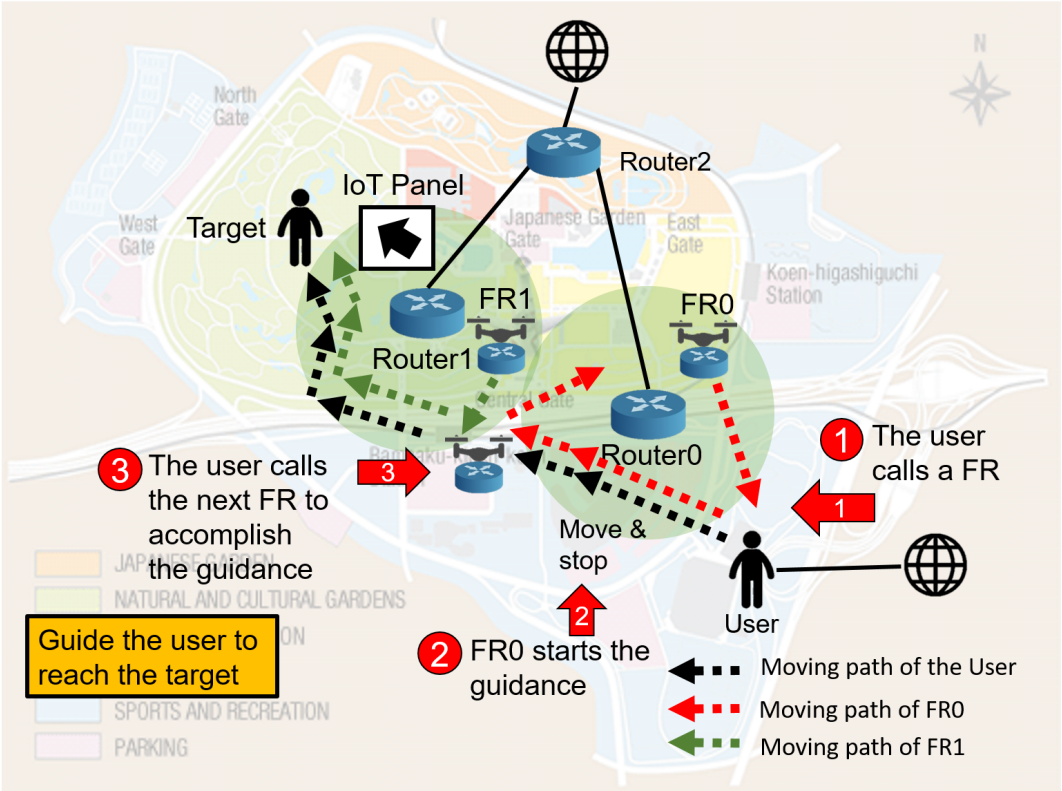

Figure 9: Guiding system overview

The system is supposed to make guidance devices to execute the optimal behavior based on the data provided by the user and the target and the CP. In the CP, CP keeps the user and the target location data, the data of the physical path that the user and the movable target are able to pass, the data of the guidance area that the router and the guidance device belongs to, the device information including the device type and the position. In the following itemization, we show that behaviors the system is supposed to achieve and node roles that each behavior is assign to. We show the example interaction in the actual communication order in the Fig. 10

1. User: a trigger of the guidance service and provide the user location information to the system.

2. Target: the destination of the guidance and provide the target location information to the system.

3. Content provider (CP): provide the cached result of the calculation and the data for the calculation.

4. Physical path calculation server (PS): calculate the physical path between the user and the target. The physical path means the path that the user and the movable target passes (e.g., the sidewalk and the vehicle).

5. Guidance area calculation server (GS): calculate guidance areas that the physical path passes. The guidance area means the area within the communication range of the router keeping the connection with multiple guidance devices.

6. Device control instruction calculation server (GCS): calculate the device control instruction based on the guidance area and the physical path. The device control instruction means the high-level control command which will be transferred into multiple primitive library APIs for communicating with the guidance device hardware.

7. Guidance device (GD): communicate and control the guidance device hardware based on the device control instruction.

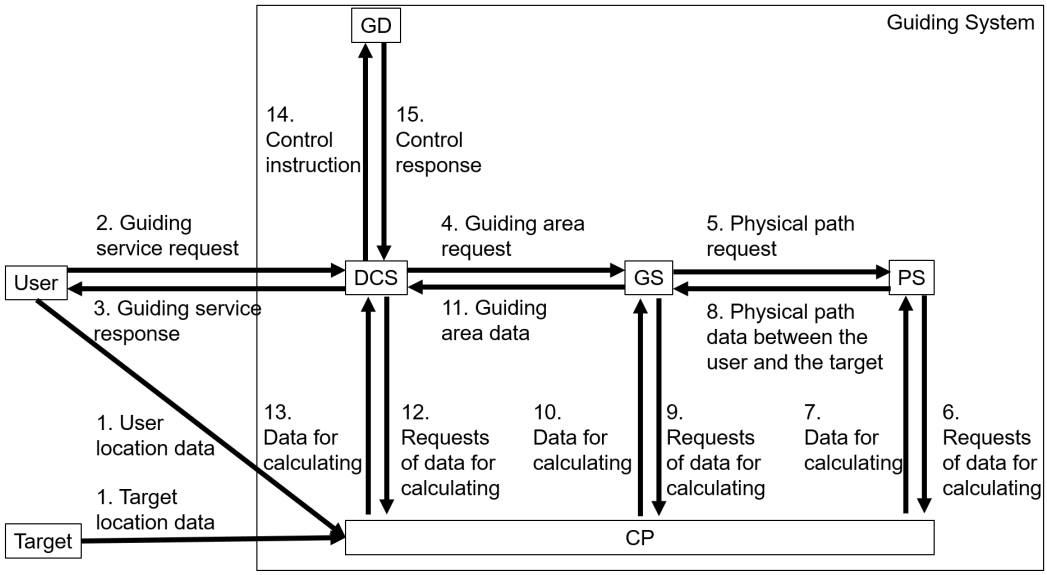

Figure 10: Guiding system node behavior overview

As for the PoC scenario, we discuss the subset of the guiding system. In this scenario, we assume that the DCS already keeps the identity of the guidance area and the physical path. The user request the guidance service. The DCS receiving the request retrieve the device information from the CP and calculate the device control instruction and send the instruction to the GD to provide the guidance service.

The reason we choose the scenario as the PoC scenario is shown in the following itemization.

- This scenario is able to show that the general design mentioned in the Sec. 3 is able to develop a specific application.
- This scenario is able to show the ICN benefits mentioned in the Sec. 3. Because the operation of the content retrieval and the instruction calculation and the device control can be embedded in the ICN layer, and these operations have to be implemented in the IP application layer.

The following itemization shows the behavior requirement of the PoC scenario communication.

1. The user is supposed to send the guidance service request to the DCS and receive the response.

2. DCS receiving the service request is supposed to send content requests to CP to retrieve the device information based on the guidance area ID that the DCS keeps.

3. DCS receiving the device information is supposed to calculate the device control instruction based on the physical path that the DCS keeps and the device information.

4. DCS finishing the instruction calculation is supposed to send the control instruction to the GD.

5. GD receiving the instruction is supposed to communicate with the device hardware to provide the guidance service.

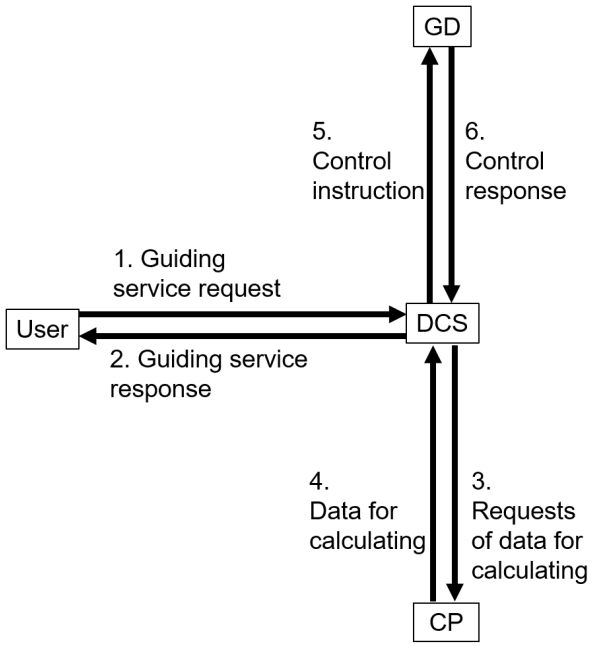

Figure 11: PoC scenario node behavior overview

4.2 ICN-based design

In the ICN-based design, modules of DCS, CP, GD are implemented in the network layer and the user module is implemented in the application layer. The DCS module is supposed

to be in network layer since the function requires computing resources and can make full use of the computing resource in the network. The CP and GD module are supposed to be in network layer since the module can be reused in other applications. The namespace of ICN packets is shown in Table 6. Fig. 12 shows the module diagram.

Table 6: Name space in the ICN-based design of the PoC scenario

Name	Description
/Advt/{ServiceID} e.g., /Advt/Guidance/User0/Target0 /Advt/DeviceInfo/Area0	The namespace of service advertisement packets. A node receiving the packet is supposed to update the pair information of the service ID and the service location in the network layer.
/Guidance/{UserID}/ {TargetID}	The namespace of the guidance service request. A user can request the guidance service from the current location of the user to the target location by sending the packet with this name.
/DeviceInfo/ {GuidanceAreaID}	The namespace of the request for retrieving the device information. A CP receiving the pakcet is supposed to return the device information including the device ID in the corresponding guidance area.
/{DeviceID}/ {GuidanceInstruction} e.g., /FR0/FRPathMove/10,10/ 20,20	The namespace of the control instruction for controlling the device hardware. A GD receiving the packet is supposed to invoke the library API able to communicate with the device hardware.

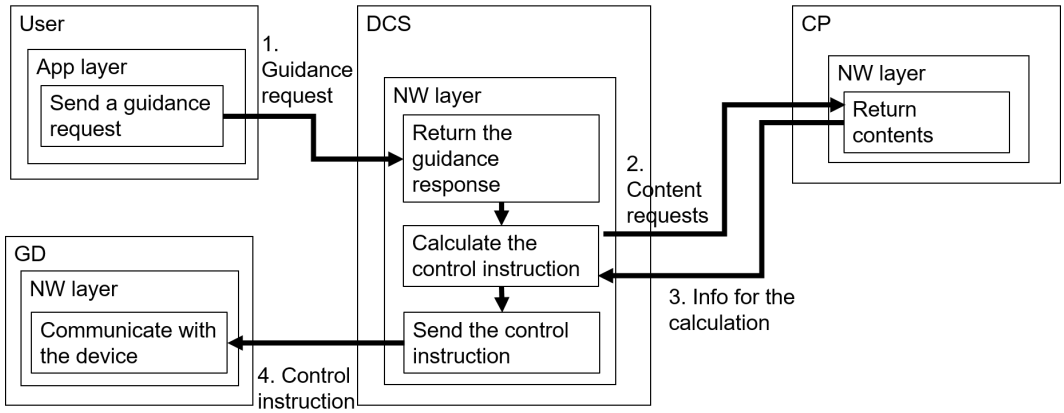

Figure 12: ICN-based design of the PoC scenario communication

4.3 IP-based design

In the IP-based design, all modules are supposed to be implemented in the application layer since the application in IP is implemented with the scratch technique. The behavior of each module is the same as the ICN-based design. Fig. 13 shows the module diagram.

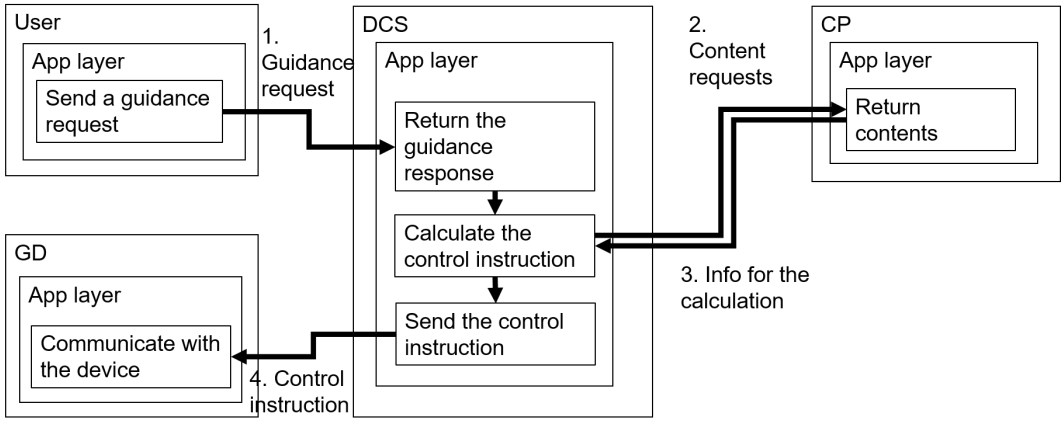

Figure 13: IP-based design of the PoC scenario communication

4.4 Experiment

The purpose of the experiment is to show the design of PoC scenario is feasible and the design of M2M communication applications mentioned in Sec. 3 is feasible to realize a

specific application.

In the experiment, we execute the code of the ICN-based and the IP-based design mentioned in Sec. 4.2, 4.3 in the local environment to confirm the feasibility. Although the design can be realized in multiple nodes, the experiment in the local environment is also able to show the feasibility of codes and to show the ICN benefit since a part of functions can be implemented in the ICN layer. We build the ICN program with named data networking (NDN) [**?**] architecture which is one of the ICN-based architecture implementation. We show codes of PoC scenario communications in the appendix of this paper.

The software used in the experiment is shown in the following table.

Table 7: Software used in the experiment

Software	Description
jndn 0.15	The Java NDN library
NFD 0.5.0	The NDN forwarding engine
DroneKit-Python 2.0	The simulator of the drone mobility
Mission Planner 1.3.44	The GUI tool monitoring the state of drones
java 8	The language used in the IP and the ICN program
Ubuntu 14.04 LTS	The OS executing the program

In the experiment, we assume that the DCS holds the data of the guidance area identity and the physical path between the target and the user. The actual guidance area identity is Area0. The physical path is expressed with the group of waypoints. We express the waypoint with a pair of two dimension coordinates (x, y) that the x is the horizontal direction coordinate and the y is the vertical direction coordinate. The actual waypoints are (0, 0), (20, 0), (20, 20). The DCS receiving the guidance service request retrieves the device information including the device ID in the corresponding guidance area from the CP. The actual device ID is FR0. The DCS receiving the device information calculates the device control instruction. We have implemented an algorithm calculating the instruction by finding out waiting waypoints that two neighbor waypoints are in the same distance in

the original physical path. We consider the robot provides the guidance service by moving and waiting at the waypoint until the user comes to the waypoint. The actual waypoint is (0, 0), (10, 0), (20, 0), (20, 10), (20, 20).

The following list shows the output message for confirming the program behavior. Fig. 14 shows the actual robot behavior. The ICN-based and the IP-based program print the similar output message and the robot does the same behavior and we only show the message and the robot behavior in ICN.

Listing 1: The output message when executing the PoC scenario communication program

```
1. GCS Service invocation
[User] Sent packet: Guidance User0 Target0
[User] Received packet: ACK
[DCS] Received packet: Guidance User0 Target0
[DCS] Sent packet: ACK

2. Content retrieval and 3. Data processing
[CP] Received packet: DeviceInfo Area0
[CP] Sent DeviceInfo: FR0
[DCS] Received DeviceInfo: FR0
[DCS] Send Device Control Request: FR0 FNPath 0.0,0.0 10.0,0.0 20.0,0.0
 20.0,10.0 20.0,20.0

4. Device control API invocation
[GD] Received packet: FR0 FNPath 0.0,0.0 10.0,0.0 20.0,0.0 20.0,10.0
20.0,20.0
[GD] Sent the Device Control response: ACK
Now Location: (10.00322558, 0.4122134499)
The remains of waypoints: (3)
Remained HoverTime:4.0
Now Location: (20.01231418, 0.1451367656)
The remains of waypoints: (2)
Remained HoverTime:5.0
Now Location: (20.02325234, 9.4599345392)
The remains of waypoints: (1)
Remained HoverTime:4.0
Now Location: (20.00322558, 15.4029904499)
The remains of waypoints: (0)
Remained HoverTime:4.0
```

Figure 14: Robot behaviors in the experiment

By confirming that the ICN-based and the IP-based program realizes the application functionality in the PoC scenario, we show that the design and the implementation method of M2M applications in ICN and IP are able to realize specific applications.

4.5 Benefits of ICN compared to IP networks

The effort of application developer writing application layer codes becomes less

Table 8 shows the amount of executabe instructions which is the executable step in the main and other methods in the ICN and the IP program. We show that the amount of executable instructions in the ICN program is 14.2% of instructions in the IP program. The IP program and ICN program is listed in the appendix of this thesis.

The reason why the instruction in the ICN program is less than the IP program is that the ICN application layer program is implemented by assembling the network layer API while IP program is implemented by the scratch technique which implements each module from high-level codes to low-level codes.

Table 8: Amount of executable instructions in the ICN and the IP program

ICN modules (lines)	IP modules (lines)
User 37	User 28
	DCS 137
	GD 56
	CP 34
Total 37	Total 261
Ratio: 14.2%	

We show an implementation example of requesting services. The following lists show the ICN-based and the IP-based content generation operation in CP. In the ICN-based program, the content generation function is implemented in the strategy class of the network layer, and the association operation of the service ID and the service function is done in the application layer or the network layer. Codes of association operation does not generally written in the application layer. In the IP-based program, codes of handling the interface and packets and generating contents have to be implemented in the application layer.

Listing 2: IP-based codes of the content generation

```
1  package IPPackage;
2
3  import java.io.BufferedReader;
4  import java.io.BufferedWriter;
5  import java.io.InputStreamReader;
6  import java.io.OutputStreamWriter;
7  import java.io.PrintWriter;
8  import java.net.ServerSocket;
9  import java.net.Socket;
10
11 public class CP_IP {
12
13   public static void main(String[] args) {
14
15     try {
16       ServerSocket ConnectionWaitingSocket = new ServerSocket(Common.
         CPPort);
17       Socket NewConnectionSocket = ConnectionWaitingSocket.accept();
18       BufferedReader reader_NewConnectionSocket = new BufferedReader(
19           new InputStreamReader(NewConnectionSocket.getInputStream()));
```

```
20        PrintWriter writer_NewConnectionSocket = new PrintWriter(
21            new BufferedWriter(new OutputStreamWriter(NewConnectionSocket.
              getOutputStream())));
22
23        String str;
24        while (true) {
25          if ((str = reader_NewConnectionSocket.readLine()) != null) {
26            System.out.println("[CP] Received packet: " + str);
27
28            if (str.contains("DeviceInfo Area0")) {
29              writer_NewConnectionSocket.println("FR0");
30              writer_NewConnectionSocket.flush();
31              System.out.println("[CP] Sent DeviceInfo: FR0");
32            }
33          }
34          Thread.sleep(1000);
35        }
36      } catch (Exception e) {
37        e.printStackTrace();
38      }
39    }
40  }
```

Accessing services with the service ID without the explicit service location

The user program is supposed to send the guidance service request to DCS. The IP socket needs the explicit service location information such as the IP address and the port number of DCS while the ICN face does not needs the information. This difference means that accessing the service in the network only needs the service ID without considering the current available service location.

Listing 3: ICN-based codes of the service accessing operation

```
1 Face GuidanceRequestFace = new Face();
2 Interest GuidanceRequest = new Interest(new Name("/Guidance/User0/Target0
  "));
3 GuidanceRequestFace.expressInterest(GuidanceRequest, new
  onGuidanceResponse());
4 GuidanceRequestFace.processEvents();
```

Listing 4: The IP-based user program of the PoC scenario

```
1 Socket socket = new Socket("localhost", Common.DCS_RecepitonPort);
2 PrintWriter writer = new PrintWriter(socket.getOutputStream(), true);
3 String GuidanceRequestName = "Guidance User0 Target0";
```

```
4 writer.println(GuidanceRequestName);
5 writer.flush();
```

5 Conclusion and future work

In this work, we have proposed an ICN-based framework for M2M applications and we have showed the ICN benefits that the effort of the application implementation with the ICN-based framework is less than the effort with IP-based framework. In order to confirm the feasibility of the ICN-based framework, we have conducted an experiment realizing a specific application with the ICN-based framework and the IP-based framework. In order to confirm the benefits of the ICN-based framework compared with the IP-based framework, we have showed the ICN and the IP codes realizing the same service function. Actual ICN codes are shorter than IP codes and we have showed that the effort of the application implementation in the ICN framework is less than the effort in the IP framework.

As the future work, we plan to discuss the methodology of the ICN application development work including the requirement analysis, design, implementation, test, deployment, maintenance.

Acknowledgments

This thesis would not accomplish without the united efforts and supports of a number of people. First, I would like to express my the deepest appreciation for my supervisor, Professor Masayuki Murata of Osaka University, for his valuable time, advice, and continual support. Furthermore, I am tremendously grateful to Professor Shingo Ata of Osaka City University for his significant guidance, precise advice, and helpful discussions. I am also thankful to Associate Professor EUM Suyong of Osaka University for his instructive advice. Moreover, I would like to express gratitude to Associate Professor Shin'ichi Arakawa, Assistant Professor Yuchi Ohsita, Daichi Kominami, and Appointed Assistant Professor Tatsuya Otoshi for their insightful comments and feedback.

Finally, I owe my gratitude to all the members of the Advanced Network Architecture Research Laboratory at the Graduate School of Information Science and Technology, Osaka University.

A. Implementation of average value retrieval

Fig.15 shows the scenario overview of average value retrieval. A user connecting to DP sends average value service requests and receives the average value from DP. DP receiving the service request sends data requests to all connecting CPs and retrieves CP data. DP receiving all CP data processes the data and return the average value to the user.

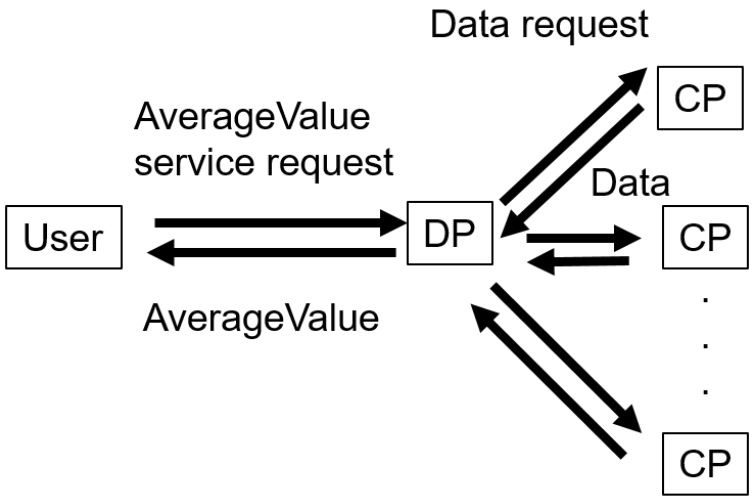

Figure 15: The overview of the average value retrieval communication

Communication diagrams in ICN over IP and IP are shown in Fig.16 and Fig.17

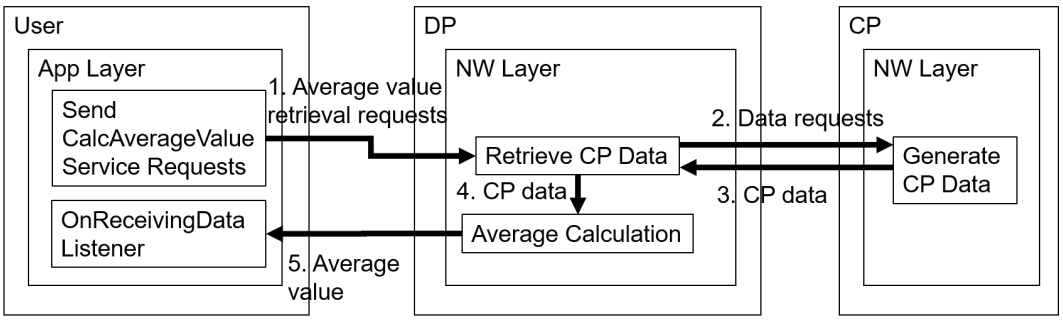

Figure 16: The diagram of the average value retrieval communication in ICN over IP

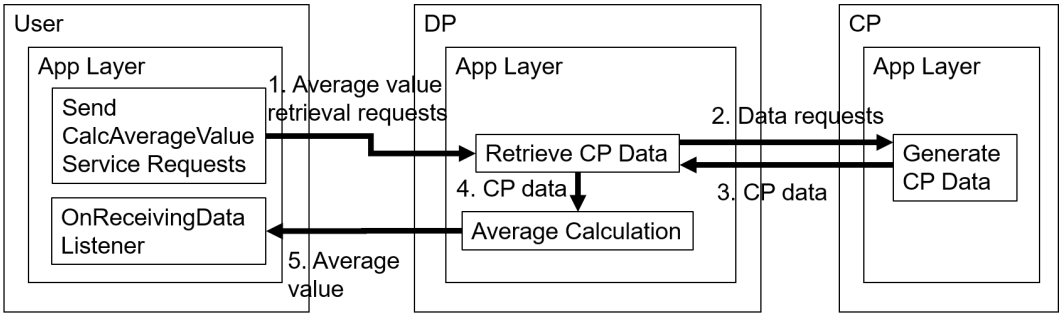

Figure 17: The diagram of the average value retrieval communication in IP

Operation requirements are shown in the following itemization.

- DP and CPs are supposed to advertise their services.
- The user is supposed to sends a service request to DP.
- DP receiving a service request from the user is supposed to retrieve sensor data from multiple CPs, and to process the data, and to respond to the service request with the process result.
- A CP receiving a data request from DP is supposed to respond with sensor data.

The required behavior sequence is shown in the following itemization.

1. Service advertisement in DP and CPs
   (a) (App in ICN) DP and CPs updates its FIB for forwarding the advertisement packets to neighbor nodes.
   (b) (App in ICN and IP) In ICN, DP and CPs send the advertisement packet to other nodes. In IP, DP and CPs send the advertisement packet to the DNS server.
   (c) (NW in ICN and App in IP) In ICN, nodes receiving the advertisement packet update their FIB in the network layer. In IP, the DNS server update the DNS records.
2. Sending service requests: (App in ICN and IP) A user sends a service request to DP.

3. Sensor data retrieval and processing and responding in DP

   (a) (NW in ICN and App in IP) If DP receives a service request from the user, DP sends data requests to all CPs and receives sensor data.

   (b) (NW in ICN and App in IP) If DP receives all sensor data, DP processes the data and sends the service response to the user.

4. Sensor data generation in CP: (NW in ICN and App in IP) If a CP receives a data request from DP, the CP sends sensor data to DP.

Behaviors of each node are shown in the following itemization.

- User: (App in ICN and IP) the user sends a service request and receives the response.
- DP

  1. Service advertisement

     (a) (App in ICN) DP appends FIB entries directing to CPs to its FIB for forwarding the advertisement packets to neighbor nodes.

     (b) (App in ICN and IP) In ICN, DP and CPs send the advertisement packet to other nodes. In IP, DP and CPs send the advertisement packet to the DNS server.

     (c) (NW in ICN and App in IP) In ICN, nodes receiving the advertisement packet update their FIB in the network layer. In IP, the DNS server update the DNS records.

  2. Sensor data retrieval and processing and responding

     (a) (NW in ICN and App in IP) If DP receives a service request from the user, DP sends data requests to all CPs and receives sensor data.

     (b) (NW in ICN and App in IP) If DP receives all sensor data, DP processes the data and sends the service response to the user.

- CP

  1. Service advertisement

(a) (App in ICN) CP appends an FIB entry directing to DP to its FIB for forwarding the advertisement packets to neighbor nodes.

(b) (App in ICN and IP) In ICN, CP sends the advertisement packet to neighbor nodes. In IP, DP and CPs send the advertisement packet to the DNS server.

(c) (NW in ICN and App in IP) In ICN, nodes receiving the advertisement packet update their FIB in the network layer. In IP, the DNS server update the DNS records.

2. (NW in ICN and App in IP) If a CP receives a data request from DP, the CP sends sensor data to DP.

Application layer developers generally divides the application logic into multiple high-level functions and divides the high-level functions into low-level functions until the low-level functions come to be library APIs or service APIs. In the application layer program, the application logic is divided into the invocation of multiple high-level functions in the application-specific order. Each high-level function is supposed to be realized by invoking multiple low-level functions.

User program

Listing 5: The user program of requesting the average value in ICN

```
1  package net.named_data.jndn.tests;
2
3  import java.nio.ByteBuffer;
4  import net.named_data.jndn.Data;
5  import net.named_data.jndn.Face;
6  import net.named_data.jndn.Interest;
7  import net.named_data.jndn.Name;
8  import net.named_data.jndn.OnData;
9
10 public class User_RequestingAverageValue_ICN{
11   public static void
12   main(String[] args){
13     try{
14       Face face = new Face();
15       face.expressInterest("/CPData/0..N/CalcAverageValue", new
         onReceivingDataListener());
16       face.processEvent();
17     }
```

```
18      catch (Exception e){
19        System.err.println(e);
20      }
21    }
22  }
23
24  class onReceivingDataListener implements OnData{
25    public void onData(Interest interest, Data data) {
26      ByteBuffer content = data.getContent().buf();
27      for (int i = content.position(); i < content.limit(); ++i)
28        System.out.print((char) content.get(i));
29    }
30  }
```

Listing 6: The user program of requesting the average value in IP

```
1  package RequestingAverageValue;
2
3  import java.io.BufferedReader;
4  import java.io.IOException;
5  import java.io.InputStreamReader;
6  import java.io.PrintWriter;
7  import java.net.InetAddress;
8  import java.net.Socket;
9
10 public class User_RequestingAverageValue_IP {
11
12   final static int DPReceptionPort = 10000;
13
14   public static void main(String args[]) {
15
16     String DNSAddr = args[0];
17     String UserAddr;
18     String IncomingMessage;
19
20     try {
21       UserAddr = InetAddress.getLocalHost().getHostAddress();
22       Runtime.getRuntime().exec("ssh " + DNSAddr + " dnscmd /RecordAdd
         CalcAverage.com User A " + UserAddr);
23
24       Socket socket = new Socket("DP.CalcAverageValue.com",
         DPReceptionPort);
25       BufferedReader IncomingStreamFromNW = new BufferedReader(new
         InputStreamReader(socket.getInputStream()));
26       PrintWriter OutgoingStreamToNW = new PrintWriter(socket.
         getOutputStream());
27
28       OutgoingStreamToNW.print("RequestAverageValue");
29       OutgoingStreamToNW.flush();
30       while (true) {
31         while ((IncomingMessage = IncomingStreamFromNW.readLine()) != null
           )
```

```
32            System.out.print(IncomingMessage + "\n");
33        }
34      }
35      catch (IOException e) {
36        System.err.println(e);
37      }
38    }
39  }
```

DP program

Listing 7: The DP program of requesting the average value in ICN

```
1  package net.named_data.jndn.tests;
2
3  import java.nio.ByteBuffer;
4  import net.named_data.jndn.Data;
5  import net.named_data.jndn.Face;
6  import net.named_data.jndn.Interest;
7  import net.named_data.jndn.Name;
8  import net.named_data.jndn.OnData;
9
10 public class DP_RequestingAverageValue_ICN{
11   public static void
12   main(String[] args){
13     String UserAddr = args[0];
14     try {
15       Runtime.getRuntime().exec("nfdc route add prefix /Advt nexthop " +
         UserAddr);
16       Face face = new Face();
17       face.expressInterest("/Advt/CalcAverageValue");
18       face.processEvent();
19     } catch (IOException e) {
20       e.printStackTrace();
21     }
22   }
23 }
```

Listing 8: The DP program of requesting the average value in IP

```
1  package RequestingAverageValue;
2
3  import java.io.BufferedReader;
4  import java.io.IOException;
5  import java.io.InputStreamReader;
6  import java.io.PrintWriter;
7  import java.net.DatagramPacket;
8  import java.net.InetAddress;
9  import java.net.MulticastSocket;
```

```
10 import java.net.ServerSocket;
11 import java.net.Socket;
12 import java.util.concurrent.BlockingQueue;
13 import java.util.concurrent.LinkedBlockingQueue;
14
15 public class DP_RequestingAverageValue_IP {
16
17   final static int DPReceptionPort = 10000;
18
19   public static
20   void main(String[] args) {
21     try {
22       String DNSAddr = args[0];
23       InetAddress CPMCastAddr = InetAddress.getByName(args[1]);
24       int CPSize = Integer.parseInt(args[2]);
25       String DPAddr = InetAddress.getLocalHost().getHostAddress();
26       BlockingQueue<String> queue = new LinkedBlockingQueue<>();
27
28       // Service location advertisement
29       // register the DNS record of DP
30       Runtime.getRuntime().exec("ssh " + DNSAddr + " dnscmd /RecordAdd
         CalcAverage.com DP A " + DPAddr);
31
32       // Wait for the handshake from the user and CPs and create sockets
33       ServerSocket ServerSocket = new ServerSocket(DPReceptionPort);
34       String UserAddr = InetAddress.getByName("User.CalcAverageValue.com")
         .getHostAddress();
35       while (true) {
36         Socket socket = null;
37         socket = ServerSocket.accept();
38         if (socket.getInetAddress().getHostAddress() == UserAddr)
39           new UserSocketBehaviorThread(socket, CPMCastAddr, queue).start()
             ;
40         else
41           new CPSocketBehaviorThread(socket, CPSize, queue).start();
42       }
43     } catch (IOException e) {
44       System.err.println(e);
45     }
46   }
47 }
48
49 class UserSocketBehaviorThread extends Thread {
50
51   final static int CPMCastPort = 10001;
52   public static final int PACKET_SIZE = 1024;
53   // private MulticastSocket CPMCastSocket;
54   Socket SocketToUser;
55
56   byte[] buf = new byte[PACKET_SIZE];
57   DatagramPacket packet = new DatagramPacket(buf, buf.length);
58   String IncomingMessage;
```

```
59  MulticastSocket CPMCastSocket;
60
61  InetAddress CPMCastAddr;
62  BlockingQueue<String> queue;
63
64  UserSocketBehaviorThread(Socket t_socket, InetAddress t_MCastAddr,
    BlockingQueue<String> t_queue) {
65    this.SocketToUser = t_socket;
66    this.CPMCastAddr = t_MCastAddr;
67    this.queue = t_queue;
68  }
69
70  // Process CP data and respond to the service request with the average
    value
71  // after receiving all CP data
72  @Override
73  public
74  void run() {
75    try {
76      // Wait for receiving a packet
77      BufferedReader IncomingStream = new BufferedReader(new
        InputStreamReader(SocketToUser.getInputStream()));
78      PrintWriter OutgoingStream = new PrintWriter(SocketToUser.
        getOutputStream());
79
80      // Wait for response of CP data
81      // Send data requests and process data and respond to the
82      // service request
83      CPMCastSocket = new MulticastSocket();
84      while (true) {
85        while ((IncomingMessage = IncomingStream.readLine()) != null){
86          // Send CP data requests after receiving a service
87          // request
88          if (IncomingMessage.equals("CalcAverageValue")) {
89            String CPDataRequestMessage = "CPDataRequest";
90            byte[] bytes = CPDataRequestMessage.getBytes();
91            DatagramPacket DataRequest = new DatagramPacket(bytes, bytes.
              length, CPMCastAddr,
92                CPMCastPort);
93            CPMCastSocket.send(DataRequest);
94          }
95        }
96        //If average value has been calculated, send the value to the user
97        for(String entry : queue){
98          if(entry.contains("AverageValue")) OutgoingStream.write(entry);
99        }
100       Thread.sleep(100);
101     }
102   }
103   catch (IOException | InterruptedException e) {
104     System.err.println(e);
105   }
```

```
106   }
107 }
108
109
110 class CPSocketBehaviorThread extends Thread {
111
112   public static final int PACKET_SIZE = 1024;
113   //private MulticastSocket CPMCastSocket;
114   Socket socket;
115
116   byte[] buf = new byte[PACKET_SIZE];
117   DatagramPacket packet = new DatagramPacket(buf, buf.length);
118   String IncomingMessage;
119   MulticastSocket CPMCastSocket;
120   BlockingQueue<String> queue;
121
122   int CPSize;
123   float Average;
124
125   CPSocketBehaviorThread(Socket t_socket,int t_CPSize, BlockingQueue<
      String> t_queue) {
126     this.socket = t_socket;
127     this.CPSize = t_CPSize;
128     this.queue = t_queue;
129   }
130
131   // Process CP data and respond to the service request with the average
      value
132   // after receiving all CP data
133   @Override
134   public
135   void run() {
136     try{
137       //Wait for receiving a packet
138       BufferedReader IncomingStream = new BufferedReader(new
          InputStreamReader(socket.getInputStream()));
139       CPMCastSocket = new MulticastSocket();
140
141       //Wait for response of CP data
142       //Send data requests and process data and respond to the service
          request
143
144       while(true){
145         while ((IncomingMessage = IncomingStream.readLine()) != null){
146           if(IncomingMessage.contains("CPData")){
147               //Packet payload example: CPData CP0 100
148               //if the data has been received from a certain CP, drop the
                  packet
149               if(queue.contains(IncomingMessage)) continue;
150               //if not, add to the queue
151               else queue.put(IncomingMessage);;
152
```

```
153             //if DP receives all data from CPs in the list, DP responds to
                 the user with the average value
154             if(queue.size() == CPSize){
155               int Total = 0;
156               for(String entry : queue){
157                 String[] message = entry.split(" ");
158                 Total+= Integer.parseInt(message[2]);
159               }
160               Average = Total / CPSize;
161               queue.clear();
162               queue.put("AverageValue " + String.valueOf(Average));
163             }
164           }
165         }
166         Thread.sleep(100);
167       }
168     }
169     catch (IOException | InterruptedException e) {
170       System.err.println(e);
171     }
172   }
173 }
```

CP program

Listing 9: The CP program of requesting the average value in ICN

```
1  package net.named_data.jndn.tests;
2
3  import java.nio.ByteBuffer;
4  import net.named_data.jndn.Data;
5  import net.named_data.jndn.Face;
6  import net.named_data.jndn.Interest;
7  import net.named_data.jndn.Name;
8  import net.named_data.jndn.OnData;
9
10
11 public class CP_RequestingAverageValue_ICN{
12   public static void
13   main(String[] args){
14     String DPAddr = args[0];
15     String CPID = args[1];
16     try{
17       Runtime.getRuntime().exec("nfdc route add prefix /Advt nexthop " +
         DPAddr);
18       Face face = new Face();
19       face.expressInterest("/Advt/CPData/" + CPID);
20       face.processEvent();
21     }
```

```
22      catch (Exception e) System.out.println("exception: " + e.getMessage()
        );
23    }
24  }
```

Listing 10: The CP program of requesting the average value in IP

```
1  package RequestingAverageValue;
2
3  import java.io.IOException;
4  import java.io.PrintWriter;
5  import java.net.DatagramPacket;
6  import java.net.InetAddress;
7  import java.net.MulticastSocket;
8  import java.net.Socket;
9
10 public class CP_RequestingAverageValue_IP {
11
12   // Multicast socket Thread: receive data request
13   // Socket: send data
14
15   public static void main(String[] args) {
16     // advertisement service
17     // update the DNS record
18     String DNSAddr = args[0];
19     String CPID = args[1];
20     String CPMCastAddr = args[2];
21     String CPAddr;
22     try {
23       CPAddr = InetAddress.getLocalHost().getHostAddress();
24       Runtime.getRuntime().exec("ssh " + DNSAddr + " dnscmd /RecordAdd
         CalcAverage.com " + CPID + " A " + CPAddr);
25
26       new MCastSocketBehaviorThread(CPMCastAddr).start();
27     } catch (IOException e) {
28       e.printStackTrace();
29     }
30   }
31 }
32
33 class MCastSocketBehaviorThread extends Thread {
34
35   final static int CPMCastPort = 10001;
36   final static int PACKET_SIZE = 1024;
37   String CPMCastAddr;
38
39   MCastSocketBehaviorThread(String t_CPMCastAddr) {
40     this.CPMCastAddr = t_CPMCastAddr;
41   }
42
43   public
44   void run() {
```

```
45
46      MulticastSocket CPMCastSocket;
47
48      try {
49        CPMCastSocket = new MulticastSocket(CPMCastPort);
50        InetAddress mcastAddress = InetAddress.getByName(CPMCastAddr);
51        CPMCastSocket.joinGroup(mcastAddress);
52
53        byte[] buf = new byte[PACKET_SIZE];
54        DatagramPacket packet = new DatagramPacket(buf, buf.length);
55        while (true) {
56          CPMCastSocket.receive(packet);
57          String MCastMessage = new String(buf, 0, packet.getLength());
58          if(MCastMessage.equals("CPDataRequest")){
59            Socket DPSocket = new Socket("DP.CalcAverageValue.com",
              CPMCastPort);
60            PrintWriter OutgoingStream = new PrintWriter(DPSocket.
              getOutputStream());
61
62            OutgoingStream.print(GenerateCPData());
63            OutgoingStream.flush();
64          }
65        }
66      } catch (IOException e) {
67        e.printStackTrace();
68      }
69
70    }
71
72    private
73    String GenerateCPData(){
74      return "Dummy";
75    }
76
77  }
```

B. Implementation of the device control based on the instruction in the guiding system

User program

Listing 11: The user program of the PoC scenario communication in the guiding system in ICN

```
1  package ICNPackage;
2  import java.io.IOException;
3  import java.nio.ByteBuffer;
4  import java.util.jar.Attributes.Name;
5
6  import javax.xml.crypto.Data;
7
8  import net.named_data.jndn.Face;
9  import net.named_data.jndn.Interest;
10 import net.named_data.jndn.OnData;
11 import net.named_data.jndn.encoding.EncodingException;
12
13 public class User_ICN {
14   public static void main(String[] args){
15     Face GuidanceRequestFace = new Face();
16     try {
17       GuidanceRequestFace.expressInterest(new Name("/Guidance/User0/
         Target0"), new onGuidanceResponse());
18       while(true) {
19         GuidanceRequestFace.processEvents();
20         Thread.sleep(1000);
21       }
22     } catch (IOException | EncodingException | InterruptedException e) {
23       e.printStackTrace();
24     }
25   }
26 }
27
28 class onGuidanceResponse implements OnData{
29     public void
30     onData(Interest interest, Data data)
31     {
32       System.out.println
33         ("[User] Got data packet with name " + data.getName().toUri());
34       System.out.print
35         ("Payload:");
36       ByteBuffer content = data.getContent().buf();
37       for (int i = content.position(); i < content.limit(); ++i)
38         System.out.print((char)content.get(i));
39       System.out.println("");
40     }
```

```
41 }
```

Listing 12: The user program of the PoC scenario communication in the guiding system in IP

```
1
2  package IPPackage;
3
4  import java.io.BufferedReader;
5  import java.io.InputStreamReader;
6  import java.io.PrintWriter;
7  import java.net.Socket;
8
9  public class User_IP {
10
11   public static void main(String[] args) {
12     Socket socket = null;
13     PrintWriter writer = null;
14     BufferedReader reader = null;
15
16     try {
17       socket = new Socket("localhost", Common.DCS_RecepitonPort);
18
19       writer = new PrintWriter(socket.getOutputStream(), true);
20       reader = new BufferedReader(new InputStreamReader(socket.
         getInputStream()));
21
22       String GuidanceRequestName = "Guidance User0 Target0";
23       writer.println(GuidanceRequestName);
24       writer.flush();
25       System.out.println("[User] Sent packet: " + GuidanceRequestName);
26
27       String line = null;
28       while (true) {
29         if( (line = reader.readLine()) != null) System.out.println("[User
           ] Received packet: " + line );
30         Thread.sleep(1000);
31       }
32
33     } catch (Exception e) {
34       e.printStackTrace();
35     }
36
37   }
38 }
```

DCS program

Listing 13: The DCS program of the PoC scenario communication in the guiding system in IP

```
1  package IPPackage;
2
3  import java.io.BufferedReader;
4  import java.io.BufferedWriter;
5  import java.io.IOException;
6  import java.io.InputStreamReader;
7  import java.io.OutputStreamWriter;
8  import java.io.PrintWriter;
9  import java.net.ServerSocket;
10 import java.net.Socket;
11 import java.util.ArrayList;
12
13 public class DCS_IP {
14
15   public static void main(String[] args) {
16
17     try {
18       new ConnectionRequestWaitingThread().start();
19     } catch (Exception ex) {
20       ex.printStackTrace();
21     }
22   }
23 }
24
25 class ConnectionRequestWaitingThread extends Thread {
26
27   public void run() {
28
29     try {
30       ServerSocket ConnectionWaitingSocket = new ServerSocket(Common.
         DCS_RecepitonPort);
31       Socket NewConnectionSocket = ConnectionWaitingSocket.accept();
32       BufferedReader reader_NewConnectionSocket = new BufferedReader(
33           new InputStreamReader(NewConnectionSocket.getInputStream()));
34       PrintWriter writer_NewConnectionSocket = new PrintWriter(
35           new BufferedWriter(new OutputStreamWriter(NewConnectionSocket.
             getOutputStream())));
36
37       String str;
38       while (true) {
39         if ((str = reader_NewConnectionSocket.readLine()) != null) {
40           System.out.println("[DCS] Received packet: " + str);
41
42           if (str.contains("Guidance")) {
43             writer_NewConnectionSocket.println("ACK");
44             writer_NewConnectionSocket.flush();
45             System.out.println("[DCS] Sent packet: ACK");
46
47             // Send the DeviceInfo request to CP
```

```
48              Socket DeviceInfoRequestSocket = new Socket("localhost",
                Common.CPPort);
49              BufferedReader reader_DeviceInfoRequestSocket = new
                BufferedReader(
50                  new InputStreamReader(DeviceInfoRequestSocket.
                    getInputStream()));
51              PrintWriter writer_DeviceInfoRequestSocket = new PrintWriter(
52                  new BufferedWriter(new OutputStreamWriter(
                    DeviceInfoRequestSocket.getOutputStream())));
53
54              writer_DeviceInfoRequestSocket.println("DeviceInfo Area0");
55              writer_DeviceInfoRequestSocket.flush();
56              new DeviceInfoResponseWaitingThread(
                reader_DeviceInfoRequestSocket).start();
57            }
58          }
59          Thread.sleep(1000);
60        }
61      } catch (Exception e) {
62        e.printStackTrace();
63      }
64    }
65  }
66
67  class DeviceInfoResponseWaitingThread extends Thread {
68
69    public DeviceInfoResponseWaitingThread(BufferedReader br) {
70      br_ = br;
71    }
72
73    public void run() {
74      try {
75        String DeviceInfo;
76        while (true) {
77          // if DCS receives the device info, then calculate the device
78          // control instruction
79          if ((DeviceInfo = br_.readLine()) != null) {
80            System.out.println("[DCS] Received DeviceInfo: " + DeviceInfo);
81
82            ArrayList<Point> OriginalPointArray = new ArrayList<Point>();
83            OriginalPointArray.add(new Point(0, 0));
84            OriginalPointArray.add(new Point(20, 0));
85            OriginalPointArray.add(new Point(20, 20));
86
87            ArrayList<Point> CompletedPointArray = CalculateFRWaypoint(
              OriginalPointArray);
88
89            String DeviceControlRequest = "FR0 FNPath";
90            for (Point p : CompletedPointArray) {
91              DeviceControlRequest = DeviceControlRequest + " " + p.x_ +
                "," + p.y_;
92            }
```

```
93
94          Socket DeviceControlRequestSocket = new Socket("localhost",
            Common.GDPort);
95          PrintWriter writer_DeviceControlRequestSocket = new PrintWriter(
96              new BufferedWriter(new OutputStreamWriter(
                DeviceControlRequestSocket.getOutputStream())));
97
98          writer_DeviceControlRequestSocket.println(DeviceControlRequest);
99          writer_DeviceControlRequestSocket.flush();
100
101         System.out.println("[DCS] Send Device Control Request: " +
            DeviceControlRequest);
102       }
103       Thread.sleep(1000);
104     }
105   } catch (InterruptedException | IOException e) {
106     e.printStackTrace();
107   }
108 }
109
110 public static ArrayList<Point> CalculateFRWaypoint(ArrayList<Point>
    OriginalPointArray) {
111   int i = 0;
112   double CompletedDistance_Next = 0f;
113   ArrayList<Point> CompletedPointArray = new ArrayList<Point>();
114   int UnitMovingDistance = 10;
115   Point OriginalStartPoint;
116   Point OriginalEndPoint;
117   Point NextPoint;
118   Point CurrentPoint;
119
120   CompletedPointArray.add(OriginalPointArray.get(0));
121   while (i + 1 < OriginalPointArray.size()) {
122     OriginalStartPoint = OriginalPointArray.get(i);
123     OriginalEndPoint = OriginalPointArray.get(i + 1);
124
125     // calculate the unit vector between OriginalStartPoint and
126     // OriginalEndPoint
127     // calculate the distance between
128     double DeltaX = OriginalEndPoint.x_ - OriginalStartPoint.x_;
129     double DeltaY = OriginalEndPoint.y_ - OriginalStartPoint.y_;
130     double OriginalDistance = Math.sqrt(DeltaX * DeltaX + DeltaY *
        DeltaY);
131     Vector UnitVector = new Vector(DeltaX / OriginalDistance, DeltaY /
        OriginalDistance);
132     CurrentPoint = OriginalStartPoint;
133
134     CompletedDistance_Next += UnitMovingDistance;
135     while (CompletedDistance_Next < OriginalDistance) {
136       NextPoint = new Point(CurrentPoint.x_ + UnitMovingDistance *
          UnitVector.x_,
137           CurrentPoint.y_ + UnitMovingDistance * UnitVector.y_);
```

```
138         CompletedPointArray.add(NextPoint);
139         CompletedDistance_Next += UnitMovingDistance;
140         CurrentPoint = NextPoint;
141       }
142       if (CompletedDistance_Next >= OriginalDistance) {
143         CompletedPointArray.add(OriginalEndPoint);
144       }
145
146       CompletedDistance_Next = 0;
147       i++;
148     }
149
150     return CompletedPointArray;
151
152   }
153
154   private BufferedReader br_;
155 }
156
157 class Point {
158
159   public double x_;
160   public double y_;
161
162   Point(double t_x, double t_y) {
163     x_ = t_x;
164     y_ = t_y;
165   }
166 }
167
168 class Vector {
169   public double x_;
170   public double y_;
171
172   Vector(double t_x, double t_y) {
173     x_ = t_x;
174     y_ = t_y;
175   }
176 }
```

CP program

Listing 14: The CP program of the PoC scenario communication in the guiding system in IP

```
1 package IPPackage;
2
3 import java.io.BufferedReader;
4 import java.io.BufferedWriter;
```

```
5  import java.io.InputStreamReader;
6  import java.io.OutputStreamWriter;
7  import java.io.PrintWriter;
8  import java.net.ServerSocket;
9  import java.net.Socket;
10
11 public class CP_IP {
12
13   public static void main(String[] args) {
14
15     try {
16       ServerSocket ConnectionWaitingSocket = new ServerSocket(Common.
         CPPort);
17       Socket NewConnectionSocket = ConnectionWaitingSocket.accept();
18       BufferedReader reader_NewConnectionSocket = new BufferedReader(
19           new InputStreamReader(NewConnectionSocket.getInputStream()));
20       PrintWriter writer_NewConnectionSocket = new PrintWriter(
21           new BufferedWriter(new OutputStreamWriter(NewConnectionSocket.
             getOutputStream())));
22
23       String str;
24       while (true) {
25         if ((str = reader_NewConnectionSocket.readLine()) != null) {
26           System.out.println("[CP] Received packet: " + str);
27
28           if (str.contains("DeviceInfo Area0")) {
29             writer_NewConnectionSocket.println("FR0");
30             writer_NewConnectionSocket.flush();
31             System.out.println("[CP] Sent DeviceInfo: FR0");
32           }
33         }
34         Thread.sleep(1000);
35       }
36     } catch (Exception e) {
37       e.printStackTrace();
38     }
39   }
40
41 }
```

GD program

Listing 15: The GD program of the PoC scenario communication in the guiding system in IP

```
1  package IPPackage;
2
3  import java.io.BufferedReader;
4  import java.io.BufferedWriter;
```

```
5  import java.io.File;
6  import java.io.FileWriter;
7  import java.io.IOException;
8  import java.io.InputStreamReader;
9  import java.io.OutputStreamWriter;
10 import java.io.PrintWriter;
11 import java.net.ServerSocket;
12 import java.net.Socket;
13
14 public class GD_IP {
15   public static void main(String[] args) {
16
17     try {
18       ServerSocket ConnectionWaitingSocket = new ServerSocket(Common.
         GDPort);
19       Socket NewConnectionSocket = ConnectionWaitingSocket.accept();
20       BufferedReader reader_NewConnectionSocket = new BufferedReader(
21           new InputStreamReader(NewConnectionSocket.getInputStream()));
22       PrintWriter writer_NewConnectionSocket = new PrintWriter(
23           new BufferedWriter(new OutputStreamWriter(NewConnectionSocket.
             getOutputStream())));
24
25       String DeviceControlRequest;
26       while (true) {
27         if ((DeviceControlRequest = reader_NewConnectionSocket.readLine())
           != null) {
28           System.out.println("[GD] Received packet: " +
             DeviceControlRequest);
29
30           if (DeviceControlRequest.contains("FRO")) {
31             writer_NewConnectionSocket.println("ACK");
32             writer_NewConnectionSocket.flush();
33             System.out.println("[GD] Sent the Device Control response: ACK
               ");
34           }
35
36           String[] SplittedRequest = DeviceControlRequest.split(" ");
37
38
39               try {
40                 File newfile = new File("./path.txt");
41                 newfile.createNewFile();
42                   FileWriter fw = new FileWriter("./path.txt",false);
43                   for(int i = 2; i < SplittedRequest.length; i++) {
44                     String[] SplittedPoint = SplittedRequest[i].split(",")
                       ;
45                       fw.write("/" + SplittedPoint[0] + "/" +
                         SplittedPoint[1] + "/10\n");
46                   }
47                   fw.close();
48
```

```
49                  Process p = Runtime.getRuntime().exec("python ./
                    FRControlScript.py -c tcp:127.0.0.1:5760 -f FNPathRun")
                    ;
50                  BufferedReader br = new BufferedReader(new
                    InputStreamReader(p.getInputStream()));
51                  String line;
52                  while((line = br.readLine()) != null) {
53                    System.out.println(line);
54                  }
55              } catch (IOException ex) {
56                  ex.printStackTrace();
57              }
58        }
59        Thread.sleep(1000);
60      }
61    } catch (Exception e) {
62      e.printStackTrace();
63    }
64  }
65 }
```

References

[1] M. Losciale, P. Boccadoro, G. Piro, G. Ribezzo, L. A. Grieco, and N. Blefari-Melazzi, "A novel ICN-based communication bus for intelligent transportation systems," in *Proceedings of 2018 IEEE International Conference on Communications Workshops (ICC Workshops)*, pp. 1–6, May. 2018.

[2] P. Boccadoro, M. Losciale, G. Piro, and L. A. Grieco, "A standard-compliant and information-centric communication platform for the internet of drones," in *Proceedings of European Wireless 2018; 24th European Wireless Conference*, pp. 1–6, May. 2018.

[3] A. Detti, M. Orru, R. Paolillo, G. Rossi, P. Loreti, L. Bracciale, and N. B. Melazzi, "Application of information centric networking to NoSQL databases: The spatio-temporal use case," in *Proceedings of 2017 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*, pp. 1–6, Jun. 2017.

[4] U. D. Silva, A. Lertsinsrubtavee, A. Sathiaseelan, C. Molina-Jiménez, and K. Kanchanasut, "Deploying an information centric smart lighting system in the wild," *CoRR*, vol. abs/1607.05784, Jul. 2016.

[5] Y. Gao, T. Kitagawa, S. Eum, S. Ata, and M. Murata, "Realization of mobility-controlled flying router in information-centric networking," *Journal of Communications and Networks*, vol. 20, pp. 443–451, Oct 2018.

[6] T. Kitagawa, S. Ata, and M. Murata, "Mobility-controlled flying routers for information-centric networking," in *Proceedings of 2018 IEEE Consumer Communications & Networking Conference (CCNC)*, pp. 1–2, Jan. 2018.

[7] T. Kitagawa, S. Ata, and M. Murata, "Retrieving information with autonomously-flying routers in information-centric network," in *Proceedings of 2016 IEEE International Conference on Communications (ICC)*, pp. 1–6, May. 2016.

[8] C. Tschudin and M. Sifalakis, "Named functions and cached computations," in *Proceedings of Consumer Communications and Networking Conference (CCNC), 2014 IEEE 11th*, pp. 851–857, Jan. 2014.

[9] M. Król and I. Psaras, "NFaaS: Named function as a service," in *Proceedings of the 4th ACM Conference on Information-Centric Networking*, pp. 134–144, Sep. 2017.

[10] Y. Zhang, A. Afanasyev, J. Burke, and L. Zhang, "A survey of mobility support in named data networking," in *Proceedings of 2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 83–88, Apr. 2016.

[11] D. An and D. Kim, "ICN-based light-weighted mobility support in IoT," in *Proceedings of 2018 27th International Conference on Computer Communication and Networks (ICCCN)*, pp. 1–2, Jul. 2018.

[12] S. Arshad, M. A. Azam, S. H. Ahmed, and J. Loo, "Towards information-centric networking (ICN) naming for internet of things (IoT): The case of smart campus," in *Proceedings of the International Conference on Future Networks and Distributed Systems*, ICFNDS '17, pp. 41:1–41:6, Jul. 2017.

[13] G. Zhang, Y. Li, and T. Lin, "Caching in information centric networking: A survey," *Computer Networks*, vol. 57, pp. 3128–3141, Apr. 2013.

[14] G. Wu, S. Talwar, K. Johnsson, N. Himayat, and K. D. Johnson, "M2M: From mobile to embedded internet," *IEEE Communications Magazine*, vol. 49, pp. 36–43, Apr. 2011.

[15] H. Shariatmadari, R. Ratasuk, S. Iraji, A. Laya, T. Taleb, R. Jantti, and A. Ghosh, "Machine-type communications: current status and future perspectives toward 5G systems," *IEEE Communications Magazine*, vol. 53, pp. 10–17, Sep. 2015.

[16] N. Zhang, G. Kang, J. Wang, Y. Guo, and F. Labeau, "Resource allocation in a new random access for M2M communications," *IEEE Communications Letters*, vol. 19, pp. 843–846, May 2015.

[17] Y. Liu, C. Yuen, J. Chen, and X. Cao, "A scalable hybrid MAC protocol for massive M2M networks," in *Proceedings of 2013 IEEE Wireless Communications and Networking Conference (WCNC)*, pp. 250–255, Apr. 2013.

[18] M. P. Papazoglou, "Service-oriented computing: Concepts, characteristics and directions," in *Proceedings of Web Information Systems Engineering, 2003. WISE 2003. Proceedings of the Fourth International Conference*, pp. 3–12, Sep. 2003.

[19] S. Wang, R. Urgaonkar, M. Zafer, T. He, K. S. Chan, and K. K. Leung, "Dynamic service migration in mobile edge-clouds," *CoRR*, vol. abs/1506.05261, Jun. 2015.

[20] M. Bienkowski, A. Feldmann, D. Jurca, W. Kellerer, G. Schaffrath, S. Schmid, and J. Widmer, "Competitive analysis for service migration in VNets," in *Proceedings of the Second ACM SIGCOMM Workshop on Virtualized Infrastructure Systems and Architectures*, VISA '10, pp. 17–24, Sep. 2010.

[21] NDN Project, "Named Data Networking (NDN)." `http://named-data.net/`. Accessed: 2019-02-07.