Distributed Control Function Selection Method for Service Function Chaining in NDN

Naoki Yamaguchi*
naoki.11_13@asagi.waseda.jp
Waseda University
Tokyo, Japan

Hidenori Nakazato
nakazato@waseda.jp
Waseda University
Tokyo, Japan

ABSTRACT

In this article, we discuss the function selection method in Function Chaining (NDN-FC) on Named Data Networking (NDN), which is an information-centric networking technology, in order to solve the problems caused by the increase in IoT devices. There is a previous study [10] on the function selection method in NDN-FC, buttheir proposal is a centralized approach and a coordinator overlooking the entire network is required. We propose a distributed control type function selection method. In order to implement this method, it is necessary to extend interest packet, data packet and forwarding of each node. We compared the existing method and the proposed method using ndnSIM, which is an NDN network simulator. The results of the proposed method are close to those of the method that requires the coordinator in terms of service execution delay and load balancing.

CCS CONCEPTS

• **Networks** → **In-network processing**; *Naming and addressing*; *Network simulations*; **Routing protocols**.

KEYWORDS

NDN, function chaining, IoT, function selection

ACM Reference Format:
Naoki Yamaguchi and Hidenori Nakazato. 2020. Distributed Control Function Selection Method for Service Function Chaining in NDN. In *Cloud Continuum Services for Smart IoT Systems (CCIoT '20), November 16–19, 2020, Virtual Event, Japan.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3417310.3431397

1 INTRODUCTION

In recent years, IoT technology is utilized in various fields. Increase in IoT devices put a lot of strain on cloud servers that provide IoT services and on networks that deliver the contents. It is a serious problem because many IoT services, e.g. autonomous driving, smart home, and factory automation, requires low latency.

There is a proposal "Edge Computing" [2] to resolve this problem. Edge Computing is the technology, that reduces network traffic and processes data with low latency, due to processing information individually near the source of information and near the place where it used. Nevertheless, Edge Computing has the problems that a single edge needs to have the required processing power for processing requests, and it is necessary to consider where to process those requests.

To solve this problem, we have applied Service Function Chaining (SFC) [7]. SFC is a technology that defines network functions represented by NAT, Firewall, load balancer, etc. as "functions," allocates them to computing resources in the network, controls packets to go through the node with the desired functions, and perform in-network processing. By creating a function chain, a series of functions are executed to get the desired output. These functions can be started and stopped flexibly on a general-purpose OS like the functions in NFV [3], enabling more efficient processing within the network. We extend this and define function the data processing required by IoT services as a function and place it in network. In order to realize seamless and dynamic SFC, we adopt a protocol, in contrast to typical location-oriented IP protocols, called Named Data Networking(NDN)[11] which is one of content oriented protocols. NDN makes it possible to manage functions and IoT data more intuitively and easily by routing with name. Since functions for IoT services can also be requested by names, it is possible to flexibly respond even if the location of the function is dynamically changed. More efficient communication is expected by using both the two technologies of function chaining and routing by name. This combination is called NDN-FC [4]. We can prepare several instances with the same function name to avoid concentrating the lord on one function instance. Then, we need a mechanism to select an appropriate instance to be used.

In this paper, we consider the method to select the appropriate function instances among the available instances of the same function to be executed in the chain of functions. The rest of the paper is organized as follows. In section 2, NDN-FC, which is the system we assume as the environment of implementing SFC. We also review some previously proposed function selection algorithms there. We propose one function selection method and NDN-FC extension for its implementation in section 3. The proposed method was evaluated from the viewpoint of load distribution and execution delay in section 4. Lastly, section 5 concludes this paper.

2 RELATED WORKS

2.1 VNF Scheduling

There are not many related studies on SFC in a drastic network environment such as NDN. Therefore, we review some related research in the NFV environment, which is more advanced. One of the problems related to function selection is the Virtual Network

CCIoT '20, November 16–19, 2020, Virtual Event, Japan

ACM ISBN 978-1-4503-8131-4/20/11…$15.00
https://doi.org/10.1145/3417310.3431397

Function (VNF), it is network function implemented as software on a general-purpose OS, scheduling problem.

[9] showed that the VNF scheduling problem can be formulated by a flexible job-shop problem. The job-shop problem is a problem in which when there is a job consisting of multiple tasks in a fixed execution order, all the tasks are assigned to the machine that executes them so that the processing time of the job is the shortest. Think of a job as a VNF, a job as a service consisting of multiple ordered VNFs, and a machine as a VM with a VNF assigned, and this job-shop problem can be translated into a VNF scheduling problem. Flexible means that one work can be processed by two or more machines, which is consistent with the fact that similar VNFs are placed in multiple VMs in VNF scheduling. This problem is an NP-hard problem, and [9] only shows the first step in formalizing the VNF scheduling problem, and no concrete method to solve it is proposed. Further, in this problem, it is premised that the number of service requests corresponding to the number of jobs is clear, and it is a scheduling problem assuming offline. In response, [6] formulated a VNF mapping / scheduling problem premised on online, and proposed three greedy algorithms and tabu search-based heuristics as methods to solve this problem. However, both [9] and [6] do not consider the link delay when forwarding a packet from one function node to another. This delay can be ignored if all chained VNFs are located in the same data center, but if the chained VNFs are located in different data centers, for example in a multi-cloud environment. Therefore, in [8], we showed that the transmission delay cannot be ignored as the size of the network service increases, and formulated a new VNF scheduling problem that takes the transmission / scheduling delay into consideration. Furthermore, this problem was evaluated using an optimal mixed integer linear program (MILP) framework and genetic algorithm-based heuristics. So far,some studies on the scheduling problem of VNF, but all of these problems are very complicated. In the IoT data SFC we discuss, scheduling all of these is a more difficult problem, as instance of the function is located everywhere in the network. Therefore, we aim to reduce the amount of calculation and select the appropriate function instance by continuously selecting the optimal case locally.

2.2 NDN-FC

NDN-FC is a mechanism to implement SFC over NDN. Two types of packets: interest packet and data packet are used for communications in NDN. A user requests a content with an interest packet and the requested content is returned in a data packet. In NDN, a user requesting content is called a *consumer* and a holder of content is called a *producer*. In forwarding, each router decides the destination of the packet based on the content name of the interest packet. NDN-FC enables routing to functions by adding a new field called "Function Name" to interest packets. In NDN, data packets are sent back through the same path that the interest packet is forwarded through but in the reverse order. In order to apply the requested function chain to the content in a data packet, the corresponding interest packet has to pass through the function in reverse order in advance. For instance, when SFC is applied to a certain content in the order of F1, F2, and F3, the Function Name field in the interest packet needs to carry the specification /F3/F2/F1. If a router receives this interest packet, it refers to the prefix at the beginning of the Function Name and routes the packet to F3. And if an interest packet arrives at a function, each function refers to the Function Name of the interest packet and removes the leading prefix corresponding to its function name. That is, the Function Name field /F3/F2/F1 in the interest packet that reaches the function F3 is replaced with the Function Name /F2/F1. By repeating this process, the interest packet goes through all the requested functions, and the Function Name becomes empty eventually. Then finally the packet is transferred to the data producer. The data producer prepares a data packet including the requested content and the data packet is forwarded back through the path of the interest packet. The data packet reaches each function in the reverse order of the interest, and the data packet is processed by the function and delivered to the original sender of the interest packet, or consumer (Fig. 1).

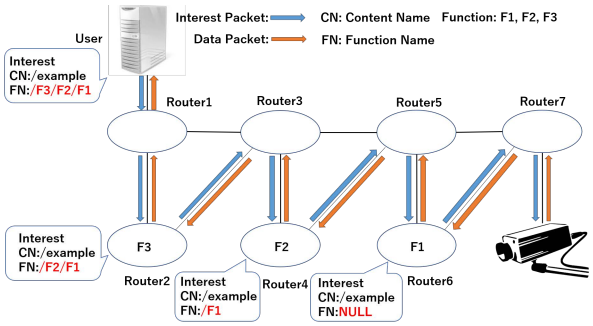

Figure 1: Forwarding in NDN-FC

2.3 Previously Proposed Function Selection Methods

[10] proposed a function selection method assuming NDN-FC focusing on the network load and function utilization in order to distribute the load of functions and minimize the service execution delay. It was shown that these objectives can be achieved by considering each function call count as the indicator of function utilization rate and the number of hops passed during SFC execution as the indicator of network load. This method requires a coordinator where load information of all functions and network topology information need to be available. Let us call this method *DL-MD-C method* for load distribution and minimum delay in centralized setting.

DL-MD-C adopts a function selection criteria which combines the number of requests on a function instance and the number of hops between functions in SFC.

Assume IoT service S is composed of a sequence of i functions deployed in a network node. The network is made of n nodes. There are m types of functions and multiple instances of the same type functions are deployed over the n nodes. Let $N = \{n_1, n_2, \ldots, n_n\}$ be the set of nodes in the network. F is the set of function types. Let F_S express the sequence of function types to be executed to implement service S, i.e., $F_s : f_1 \rightarrow f_2 \rightarrow \cdots \rightarrow f_i$ where $f_j \in F$ for $1 \leq j \leq i$.. Also, let $N(f_j)$ be the set of nodes where an instance of function type f_j is deployed.

Then, the function selection criteria is defined as follows. If C_{jk} is the number of times f_j at node n_k is called and H_{kl} is the number of hops between node n_k and n_l, the function selection criteria for service S for node k is defined as

$$U_k = \sum_{j=1}^{i} (\alpha H_{kl} + \beta C_{jk}), \quad (1)$$

where $n_l \in N(f_x)$ and f_x is the directly succeeding function after f_j, i.e., $f_j \rightarrow f_x$ in F_s. α and β are weight coefficient.

DL-MD-C method selects an instance of f_j that minimizes U_k as the component of SFC. To find the minimum U_k, the coordinator needs to manage all parameters such as the number of hops between nodes and the number of function calls to select function instances, and it will become the bottleneck. Also, to find the minimum U_k, all combinations of function instances to compose the function sequence must be examined. The computation cost of DL-MD-C is quite high.

3 FUNCTION SELECTION METHOD BY DISTRIBUTED CONTROL

Our function selection method being proposed in this paper selects one function at a time. An instance of the function to be executed next is chosen when the instance is not selected yet. The selection is performed either at the consumer node generating an interest packet with filled Function Name field or at the node of a function instance through which the interest packet goes through. In other words, the selection is performed in a distributed manner.

Our solution is an approximate algorithm and does not guarantee a strict choice. In the discussion presented in this paper, we assume one node hosts at most one function instance to verify the availability of this algorithm. As long as the number of function instances in the network is less than the number of nodes, this assumption is valid for load balancing. The method presented in this paper can be applied even if multiple function instances are hosted by one node, however.

To make the selection, some information is exchanged among consumers and functions but no coordinator which collects the view of the entire network is necessary. Each consumer node and function instance node maintains a table that stores the number of calls to functions and the number of hops to functions. The table is populated by information piggybacked on data packets as described below.

3.1 Algorithm

As mentioned in section 2.2, if Function Name field in an interest packet is filled with a sequence of function names, the interest packet is forwarded to the first function in the sequence. So, the functions after the first one in the sequence can be disregarded for forwarding purpose.

Every time a function name which is not allocated a specific function instance yet appears at the beginning of the Function Name field of an interest packet for SFC, that is, when the consumer sends the interest packet, or when the interest packet is sent out from an function instance node after execution of the function, the node n_k refers to Function Information Table (FIT) described below. The node finds H_{kl}, the number of hops from node n_k to the node n_l hosting the function instance requested as the next function in the Function Name field, and C_{jl}, the number of calls on the function instance of f_j at node n_l during the past T seconds from FIT. Then, the function instance at node n_k that minimizes V_k in eq.(2) is selected. Here, α and β represent the weight of the number of times a function is called and the number of hops to the function, and are arbitrary constants.

Assume a consumer node n_0 transmits an interest packet with its Function Name field set as F3/F2/F1 (Fig. 2). The consumer obtains H_{0l} and C_{3l} ($l \in N(f_3)$) of the nodes n_l hosting the function instances of function F3 from FIT, and evaluates them by the eq. (2). Once F3a is selected as the instance to be used among the instances F3a, F3b, and F3c of function F3, Function Name will be replaced with F3a/F2/F1 and the interest is further forwarded. When the interest packet is received by the node hosting the function instance F3a, the F3a at the beginning of the Function Name field is deleted. When the interest packet is further forwarded after execution of F3a, the function instances of the next function F2 is evaluated again by eq. (2) and replaced with the selected function instance identifier.

By repeating this operation, the interest packet is delivered to the producer specified in the content name field in the packet while selecting appropriate function instances.

$$V_k = (\alpha H_{kl} + \beta C_{jl}) \quad (2)$$

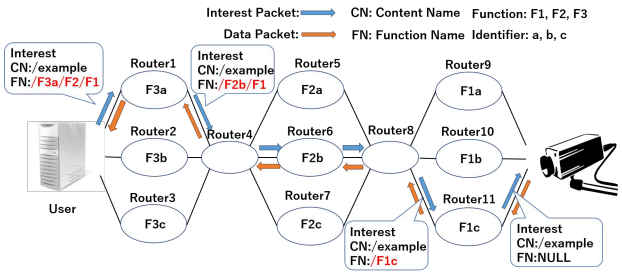

Figure 2: Forwarding by Distributed Control

3.2 Function Information Table (FIT) and Packet Format Extension

In order to implement the above method, FIT holding information of hop count and function call count is created. Also, a mechanism for updating FIT is explained in this section. As described in section 3.1, the consumer and function nodes maintain FIT.

The FIT in a node hosting function instances stores the function call counts of instances hosted by itself in addition to the information of other function instances. Its own call count is initialized in a predetermined period T. Furthermore,to avoid browsing unupdated information and sending a large number of packets to the same instance, the forwarder increments the function call count of the function instance selected when shipping the Interest. Fig.3 shows an image of the table of F3a in the example given in Fig. 2.

To update the information in FIT, it is necessary to exchange information between function nodes. We added one field to interest

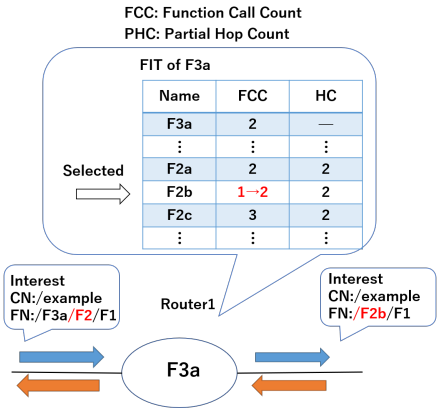

Figure 3: Image of The Table of F3a

packet format and three fields to data packet format defined in NDN-FC. The field added to interest packet is Function Full Name field. Function Name field, Partial Hop Count field, and Function Call Count filed are added to data packet.

Function Name field added in data packet is used to know the identifier of the previous function instance in the updating of FIT, and Function Full Name filed added in interest packet helps it.

Function Full Name field is initialized with *NULL*, and every time a function instance is selected at a consumer or function node, the identifier of the selected function instance is appended at the end of this field. By doing so, the sequence of function instance identifiers for the executed SFC is written in this field. When a data packet is created by the producer, the complete sequence of function instance identifiers is acquired from the Function Full Name field of the interest packet. The value in the Function Full Name field in the interest packet is copied to the Function Name field in the created data packet.

So the Function Name field added in the data packet initially has the full name. This function name field is not referenced when executing the first function of the sequence because the data packet does not have the information to update the FIT. On the other hand, when a data packet arrives at the second and subsequent instances of the assigned function in the sequence, that is, when the data Packet has the information to update FIT's information, the function instance identifier at the end of the Function Name field is extracted and removed. The extracted function instance identifier expresses the previous function executed while forwarding back the data packet. Figure 4, Figure 5 show the operations of these fields,the Function Full Name field and Function Name field,respectively.

Partial Hop Count field added in data packet is a field to measure hop count between function nodes. Partial Hop Count is NULL when a Data packet is created. When a data packet reaches a function node or consumer node, if the value of Partial Hop Count is not NULL, the hop count field of the entry that indicated by the identifier at the end of the Function Name field in its FIT is updated using that value in the Partial Hop Count field. The entry to be updated is the one for the last function instance in the of Function Name field in the data packet. When sending a Data packet from the Function node, the Partial Hop Count field is initialized to 0

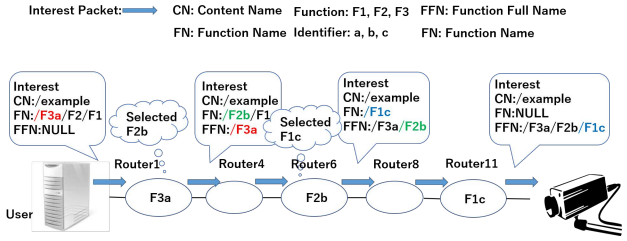

Figure 4: The Operation for The Function Full Name Field

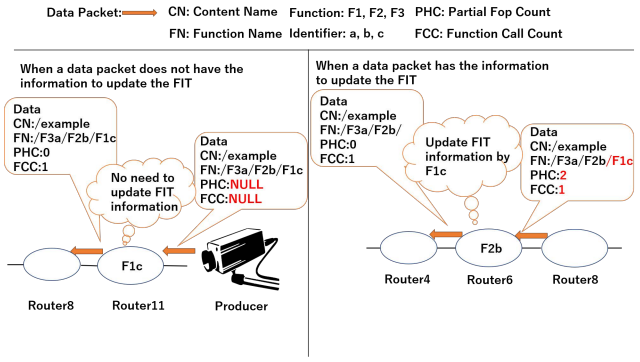

Figure 5: The Operation for The Function Name Field in Data Packet

regardless of whether the Partial Hop Count is NULL or not. If the Partial Hop Count field is not NULL, the value is incremented by 1 at every node on reception of the data packet regardless of the node's function instance hosting status.

On the other hand, Function Call Count is a field that conveys the number of times each function instance has been called. Function Call Count field is also set to NULL when a data packet is created. When a node hosting function instance or a consumer node is reached, if the value of Function Call Count field is not NULL, the information of function call count field of an entry in its FIT is updated using that value. The entry to be updated is the same entry where hop count field is updated. Furthermore, when a data packet reaches a function node, the number of times its own function instance has been called in FIT is described in Function Call Count field. The process of these series of Data packets is shown in Figure 6 using a part of the example of Figure 2.

4 EVALUATION

4.1 Simulation Environment

We performed simulations to evaluate the effectiveness of the proposed method. In the simulation, distribution of function instance load and the execution delay of services made of chains of functions were measured using an NDN simulator ndnSIM [5]. A 24-node US nation-wide topology[1] was used for the measurement (Figure 7).

Three instances of each of five different function types F1, F2, F3, F4, and F5 were deployed in this network. The instances of the same function type are distinguished by appending 'a,' 'b,' and 'c'

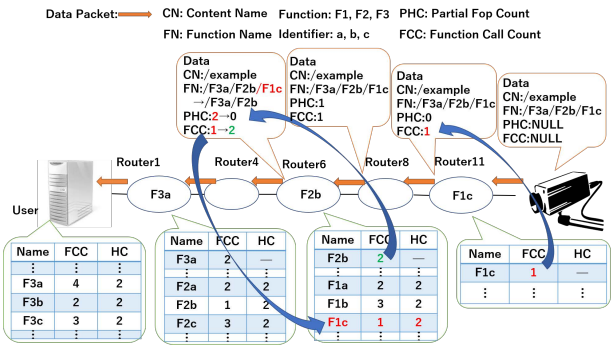

Figure 6: The Process of Data Packet

at the end of the function type name such as F1a, F1b, and F1c. Four consumers are connected to the network. The consumers generate SFC requests. Two data producers are also attached to the network. The consumers periodically select one from 12 types of SFC requests as shown in Table 1) and send out an interest packet. In this environment, we conducted a test to make 300 SFC requests from consumers.Table 2 shows the parameters used in our proposed algorithm for simulation.

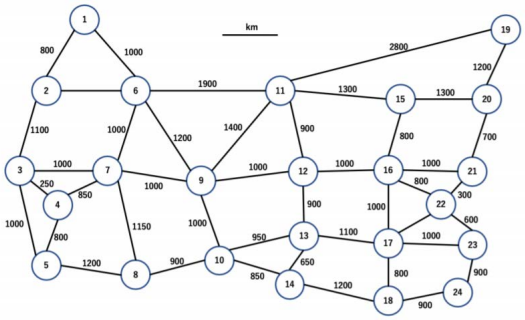

Figure 7: U.S.-24- Node Topology

Table 1: SFC Request Types

Type	Request
1	F1→F2→F4
2	F1→F2→F5
3	F2→F1→F4
4	F2→F1→F5
5	F1→F3→F4
6	F1→F3→F5
7	F3→F1→F4
8	F3→F1→F5
9	F2→F3→F4
10	F2→F3→F5
11	F3→F2→F4
12	F3→F2→F5

Table 2: Parameters

Parameter	DL-MD-C	Distributed Control
α	1	1
β	1	1
x	30[times]	-
T	-	50[ms]

For comparison, the following two function selection methods: hop-first and lord-first were prepared in addition to the proposed method and DL-MD-C.

Hop-first
Hop-first is a method that refers only to the information of hop count when selecting a function, and selects the function instances that form the path with the smallest number of hops. Hop-first is equivalent to setting $(\alpha, \beta) = (1, 0)$ in eq. (2).

Lord-first
Lord-first is a method that refers only to the information of function call count when selecting a function, and selects the one with minimum function call count. Lord-first is exactly alike $(\alpha, \beta) = (0, 1)$ in eq. (2).

4.2 Function Distribution

We evaluated how much the load of the functions is distributed. Figure 8 shows the function call count to each function instance in each method. In order to show the degree of dispersion more clearly, the *function distribution degree* defined in [10] is obtained for each method. The function distribution degree is the sum of the absolute value of the skew from the ideal number of function calls.The function distribution degree D is defines as

$$D = \sum | < \textit{observed call counts} > - < \textit{optimal call counts} > |.$$

The closer the function distribution degree is to 0, the more uniformly the function instances are called. Figure 9 compares the function distribution degrees of the four methods.

Looking at the figure, the performance of the proposed function selection method shows a value between that of Lord-First and Hop-First, which is slightly different from DL-MD-C. In particular, function call counts from F1 to F3 of proposed method shows similar characteristics to those of DL-MD-C. On the other hand, F4 and F5 are slightly different from F1 to F3. This is because F4 and F5 are called less frequently than F1 to F3, so the tables were not sufficiently updated by the Data packet. That is a major factor in the slight difference between the function dispersion of the proposed method and that of DL-MD-C.

4.3 Service Execution Delay

The delay from the user sending an SFC request to the user receiving the processed data packet for each function was measured. The time required for the function to process the data packet was set to $40ms$. The request frequency was changed from $10/s$ to $30/s$ to compare different network load. Figure 10 shows the average service execution time. The proposed method showed almost the same performance as DL-MD-C. However, when the request frequency

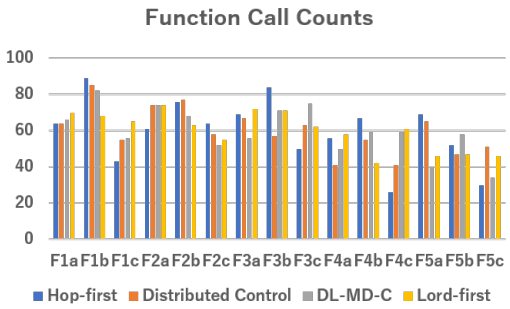

Figure 8: Function Call Counts

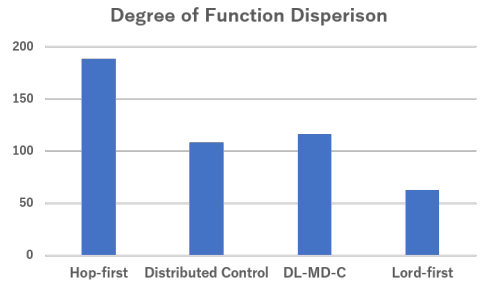

Figure 9: The Degree of Function Dispersion

is increased to 30/s, the performance of the proposed method is slightly inferior to DL-MD-C. This is because the centralized control adopted by DL-MD-C assumes function call counts of all function instances in the entire network are up-to-date, whereas the proposed method refreshes the function call count at regular intervals and the information may be tardy. The proposed method cannot adapt to excessive requests arriving in a short period of time.

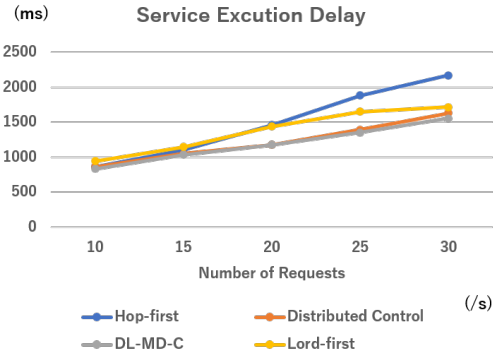

Figure 10: Service Execution Delay

5 CONCLUSION AND FUTURE WORK

In this paper, we have discussed how to implement the function selection method in order to perform in-network processing of IoT data by SFC using NDN on an actual network. By distributed control that selects a function instance each time a request goes to a function node, we aimed to avoid the problem caused by setting a coordinator which becomes a bottleneck. Our distributed approach brings the performance of function instance selection closer to the ideal function selection. When the function distribution degree and service execution delay are evaluated for the proposed method, the performance of the proposed method is close to DL-MD-C which assumes existence of a coordinator where all information need to be gathered.

As a future extension of this work, we are going to consider the effect of cache at network nodes which is a feature of NDN. It is considered that the service execution delay can be shortened by caching the data processed by the function in advance. However, if the proposed method is applied as it is, there is a concern that the information on function call count and hop count is out of date when using the cache. Moreover, although only the function selection is considered in this paper, where to place the function is also an important issue in order to use the function efficiently. In the future, it will be necessary to coordinate the allocation and selection of functions.

ACKNOWLEDGMENTS

The research leading to these results has been supported by the Ministry of Internal Affairs and Communications under Grant No. JPJ000254, "R & D of fundamental technologies for effective use of wired/wireless optimal control type radio waves corresponding to increased IoT equipment" for the research and development of the expansion of radio wave resources.

REFERENCES

[1] Dragos Andrei, Biswanath Mukherjee, and Dipak Ghosal. 2020. Online Scheduling of Large File Transfers over Lambda Grids. (09 2020).
[2] GE Digital. [n.d.]. *Edge Computing and Cloud Give Intelligent Machines a Balanced Load.* Retrieved September 11, 2020 from https://www.ge.com/digital/blog/edge-computing-and-cloud-give-intelligent-machines-balanced-load
[3] Network Functions Virtualisation ETSI. 2014. Network Functions Virtualisation (NFV). *Management and Orchestration* 1 (2014), V1.
[4] Yohei Kumamoto, Hiroki Yoshii, and Hidenori Nakazato. 2020. Real-World Implementation of Function Chaining in Named Data Networking for IoT Environment. In *2020 IEEE ComSoc International Communications Quality and Reliability Workshop (CQR).* 1 – 6.
[5] Spyridon Mastorakis, Alexander Afanasyev, Ilya Moiseenko, and Lixia Zhang. 2016. *ndnSIM 2: An updated NDN Simulator for NS-3.* Technical Report NDN-0028. Named Data Networking Project.
[6] R. Mijumbi, J. Serrat, J. Gorricho, N. Bouten, F. De Turck, and S. Davy. 2015. Design and evaluation of algorithms for mapping and scheduling of virtual network functions. In *Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft).* 1–9.
[7] NOIA. [n.d.]. *Service Function Chaining (SFC).* Retrieved September 14, 2020 from https://docs.noia.network/v1.0/noia/service-function-chaining--sfc-
[8] L. Qu, C. Assi, and K. Shaban. 2016. Delay-Aware Scheduling and Resource Optimization With Network Function Virtualization. *IEEE Transactions on Communications* 64, 9 (2016), 3746–3758.
[9] J. F. Riera, E. Escalona, J. Batallé, E. Grasa, and J. A. García-Espín. 2014. Virtual network function scheduling: Concept and challenges. In *2014 International Conference on Smart Communications in Network Technologies (SaCoNeT).* 1–5.
[10] Y. Shiraiwa and H. Nakazato. 2019. Function Selection Algorithm for Service Function Chaining in NDN. In *2019 IEEE ComSoc International Communications Quality and Reliability Workshop (CQR).* 1–5.
[11] Lixia Zhang, Alexander Afanasyev, Jeffrey Burke, Van Jacobson, kc claffy, Patrick Crowley, Christos Papadopoulos, Lan Wang, and Beichuan Zhang. 2014. Named Data Networking. *SIGCOMM Comput. Commun. Rev.* 44, 3 (July 2014), 66–73.