International Journal of Multimedia and Ubiquitous Engineering
Vol. 9, No. 11 (2014), pp. 197-208
http://dx.doi.org/10.14257/ijmue.2014.9.11.20

Design of an Efficient Method for Identifying Virtual Machines Compatible with Service Chain in a Virtual Network Environment

Hyeonseok Oh[1], Daeun Yu[1], Yoon-Ho Choi[2*] and Namgi Kim[1]

[1]Department of Computer Science, Kyonggi University, Iui-dong, Yeongtong-gu, Suwon-si, Gyeonggi-do, Korea
[2]Department of Convergence Security, Kyonggi University, Iui-dong, Yeongtong-gu, Suwon-si, Gyeonggi-do, Korea
[1]{lims, deyoo, ngkim}@kyonggi.ac.kr, [2]ychoi@kyonggi.ac.kr

Abstract

With advancements in network technologies, network virtualization has been proposed to efficiently provide a large number of services and flexible management by utilizing limited resources over existing networks as much as possible. Network virtualization has been proposed as a new paradigm for networks, as it simplifies complicated network configurations for convenient maintenance through the maximum utilization of limited network resources. However, problems have arisen due to network virtualization, such as how to assign real network components that are compatible with virtual network components over a large number of virtual machines existing in a network. In particular, the service-chaining concept—where a network flow only passes through needed services—has been newly introduced by combining the recently highlighted Software-Defined Network with a virtualization concept called Network Function Virtualization. As a result, studies on which virtual machines are selected and how to connect them have increased ever more. Accordingly, this paper aims to identify virtual machines that are compatible with service chaining in a virtual network environment where virtual machines are dispersed, and it proposes a method of how to create a path by connecting virtual machines.

Keywords: *Virtual Network, Software-Defined Network, Network Function Virtualization, Service Chaining, Finding Compatible Virtual Machines*

1. Introduction

As the number of network users has increased in recent years, services using networks have also increased, creating many problems, such as limited resources and the construction of new networks for new services. To resolve such problems, a number of studies have been conducted.

As a result of such studies, a host virtualization concept has been proposed to maximize the utilization of limited resources by converting many logical hosts into a single physical host, thereby reducing the idle state as much as possible, or converting a number of physical hosts into a single logical host to enable efficient management. The virtualization technique has been extended to a network layer so that physical devices in a network, such as switches or links, can also be segmented into logical devices, similar to hosts.

Through this process, idle resources in a network can be utilized as much as possible, and an ease-of-management environment can be constructed. In addition, a network environment,

[*] Corresponding Author

which is independent of services and users, can be provided easily without physical installation and modifications by using segmented logical devices. However, since services and paths in a network have become more complicated than existing systems under the virtual network environment, the optimal path search for flows and traffic management will be more difficult. A Software-Defined Network (SDN) has been introduced to perform flow-path and traffic management so that network traffic can move more efficiently. Since a specific flow can be passed through a desired path in the SDN, service chaining—where specific hosts can be passed through to meet the characteristics of a flow— has been proposed. In addition to the conventional concept of an optimal path, which simply indicates a path from source and destination points, service chaining can add nodes that must be passed through. In this paper, an algorithm that can determine an optimal service chain path is proposed to solve the above problems by considering the network performance factors and services of each virtual machine.

This paper is organized in the following way. In Section 2, technologies related to the proposed algorithm are explained, while the algorithm that aims to solve the above problems is explained in Section 3. After we describe the evaluation results under various circumstances in Section 4, we conclude this paper in Section 5.

2. Related Studies

2.1. Virtual Network Embedding (VNE)

Network virtualization is used to virtualize all network devices in a network to alter them into logical devices, thereby creating a virtual network using logical devices [1]. The most basic step is to map the virtual nodes and links in a virtual network to the physical devices in a real network. During such mapping operations, it is important to meet constraints, such as host process capability or bandwidth size, over the virtual network while creating no excessive load on the network. This is called an NP-Hard problem. To solve the virtual network mapping problem, there are two approaches: heuristic programming and integer programming. Both of these two methods aim to reduce the number of network links and loads while producing optimal performance.

2.2. Software-Defined Network (SDN)

A switch that serves packet transmissions over existing networks has two planes: a data plane that is responsible for data transmissions and a control plane that indicates the transmission direction. These two planes are combined into one structure in a switch. However, this combined structure cannot identify the overall network status so it cannot produce a compatible traffic path for a large number of services over a network. To solve this problem, an SDN was proposed. This is to separate the combined structure into two planes; a switch manages the data plane, while a controller collects the control plane of every switch in a centralized manner, thereby ascertaining the overall network topology information. Based on this information, when a flow enters the controller-managed network, a flow with the shortest path is produced. This considers the capacity of each link using the shortest path-finding algorithm, thereby preventing network performance degradation, which is due to the duplication of paths, as previously experienced.

2.3. Network Function Virtualization (NFV)

Due to advancements in network technologies, the frequency of new services has increased gradually. Thus, recurrent deletions of existing network functions or additions of new functions occur. However, since most traditional networks consist of proprietary hardware-based devices, it is difficult to remove existing functions, add new functions, or manage maintenance. To solve these problems, Network Function Virtualization (NFV) was proposed. With NFV, a network structure consists of several simple but high-performance components, such as high-capacity storages, high-performance servers, and high-performance switches, while the required functions are implemented using software and added to each component [2]. Through NFV, a network structure that is flexible to change and does not depend on specific proprietary venders can be created.

2.4. Service Chaining in SDN

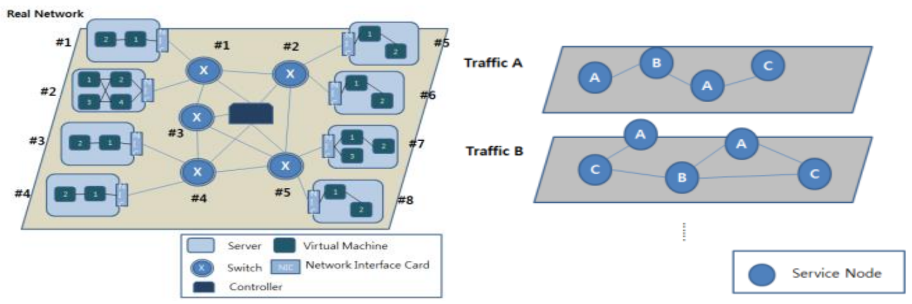
As network flows can be directed to a compatible path in a network using an SDN, a service-chaining concept has been proposed where, according to the flow characteristics, a flow can pass through only the devices that provide required services among the existing network services [3]. Such service chaining is especially effective in the security sector. For example, the service chaining concept has been used to classify a flow by attack type, after which a measure was proposed to improve security performance via service chains that correspond to a certain attack type of a specific flow [4].

2.5. Web Service Composition

There has been a consistent study on how to combine various services over the web, ever since prior to the proposal of a service-chaining concept in the SDN, as mentioned above. As user service requirements have become more and more complicated, a number of studies have been conducted on combining existing web services dispersed over networks to provide new web services. Such a web service composition basically determines whether two web services are required to be linked by using the definition of a user’s requirements as well as the input, the output, the previous state, and the modified state of the final results received after each service is provided. To determine this, a representation of the correlation between web services has to be generated; therefore, a form of graphs [5, 6] or trees [7] was employed to represent the correlation. This represented correlation is used to search for the services—to be linked through search algorithms—that are compatible with each data structure used in the representation of the correlation, thereby being linked with prior services. Also, G. Zayaraz et. al used CPN model for functional testing of Web service composition [8].

3. Proposed Algorithm

In this paper, a method is proposed of how to create an optimal service-chain path, which is compatible with user requirements or flow characteristics in a virtual network. To this end, a network structure—to which this algorithm is applied—is first explained, and the proposed algorithm that can be operated in the above network structure is explained thereafter.



(a) (b)

Figure 1. Topological Overview of Network Architecture for Implementing the Proposed Algorithm : (a) An Example of the Real Network Structure for Virtualization; (b) An Example of the Virtual Network Topology that can be Inserted into a Controller

3.1. Network Structure

The network structure considered in this paper consists of high-performance devices required in NFV as well as an SDN-applied network in which controllers are added. Over the network structure, each high-performance server consists of an internal network, which is composed of a number of virtual machines, while a virtual machine(VM) serves one of the functions of the network components, such as firewalls or detection systems. Switches that are connected to servers in the network are interlinked to a controller, while a controller receives traffic flow information from the switches to properly identify the flow characteristics. In addition, the characteristics of every virtual machine are also provided to a controller according to a specific event or a certain interval, and this information is employed later in the proposed algorithm. Figure 1(a) shows an example of the network that employs this proposed structure.

Traffic identified in this structure can have a logically reconfigured virtual network according to the traffic characteristics; therefore, an administrator can configure a virtual network topology that is optimized for this specific flow, and the administrator can enter this information into a controller. Figure 1(b) shows a graphic form of a virtual network that can be configured according to the traffic characteristics. Here, a labeled mark in each node represents a specific service. When services are added, information on the service addition is transmitted to a controller to be used during path creation, which is compatible with future service-chaining.

3.2. Detailed Explanation of the Algorithm

The proposed algorithm in this paper largely consists of two steps. The first step is to find a virtual machine that is compatible with the required services, and the second step is to determine a path. To solve the problem in each step, first, a method of how to select optimal virtual machines using a vector space model—in which various pieces of information about a virtual machine are used—is provided. Second, using the selected virtual machine, a service-chaining path is virtually created by using a search via the shortest distance-finding algorithm, which is then converted into a path in the real network environment.

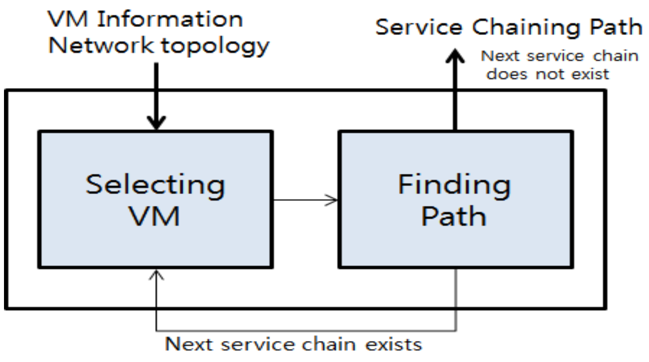


Figure 2. A Schematic Diagram of the Proposed Algorithm

3.2.1. Virtual Machine Selection

To find virtual machines that are compatible with user requirements or flow characteristics, this algorithm uses a virtual network topology that exists in a controller. The information inside a controller includes a set of values received and requested from each host during the system initialization, which is necessary to calculate the optimal value compatible with the requirements. Table 1 explains these values.

The upper three items in Table 1 are necessary to find the virtual machines that are compatible with the requirements, such as performance or power consumption, while the last item is an element considered for the service chain. To find compatible virtual machines using such items, all elements are represented by an integer format followed by a 4D vector. Each element's value can be updated by the initial request or through periodic communication with the devices as well as through controller operations. All items in a single 4D vector produced through the above operation can be arranged in a 3D vector space, other than the last item, which refers to service number. A compatible virtual machine can be found using Algorithm 1 and the arranged items. Table 2 explains the parameters used in the algorithm 1.

The algorithm 1 is developed based on the Euclidian distance, and it does not aim to calculate real distance but to simply compare distances. Therefore, the computation excluded root values to compare distances for the purpose of a reduction in computational load.

A virtual machine that has the shortest distance between the requirement vector and the VM vector in the aforementioned 3D vector model is selected as the optimal virtual machine. This algorithm sorts and stores the distances between the requirement vector and virtual machines in an ascending order upon initial request, and it returns the lowest value on the far left side whenever the same request arrives. No modifications are found in the values in the network topology. The following is an example of the above algorithm.

The distances to the VM are shown in Table 3 where the requirement vector is (0, 0, 0) and the VM vectors indicating the required items without providing service numbers are arranged in the 3D space, as shown in Figure 3. Based on the results, the virtual machine most compatible with the user or the requirement vector is No. 3 VM, so that No. 3 VM is returned as the result of the request. Administrators can use the algorithm for security as well as for traffic or resource management.

Table 1. Information Required for Processing from a Virtual Machine

Item name	Description
session_num	The number of sessions that are currently connected: the available number of sessions is limited in general, and the greater the number of sessions, the worse the throughput becomes.
proc_pkt	Throughput by packet size: the maximum packet throughput by a processer per second.
protocol_proc	Throughput by protocols: throughput by protocol of the input flow.
service_num	Provided service number: service number provided by a virtual machine.

Table 2. Parameters for Algorithm 1

Parameter	Description	Parameter	Description
vm_set	A set of inputted virtual machines (VMs)	proc_pkt	Packet throughput per second
req_vec	Optimal VM requirement vector inputted by a user	distance	Distance between VM vector and user requirement vector
service_num	One of the attributes of req_vec, which refers to service number preferred by a user	session_num	Number of sessions connected to a VM
protocol_proc	Protocol throughput		

Algorithm 1

```
1: Input: Information vector about VMs (the number of
2: sessions currently connected, throughput by packet size, average packet
size, provided service number), User requirement vector
3: Output: a set of VMs compatible with the requirements
4:
5: PROC FIND_VM:
6: vm_set   = get_vm(req_vec.service_num)
7: FOR i 0 TO vm_set.length:
8:   distance := pow(req_vec.session_num
9:                       - vm_set[i].session_num, 2)
10:               + pow(req_vec.proc_pkt_num
11:                       - vm_set[i].proc_pkt, 2)
12:               + pow(req_vec.protocol_proc
13:                       - vm_set[i].proctocol_proc, 2)
14: sorting by distance in vm_set
15: RETURN vm_set
```

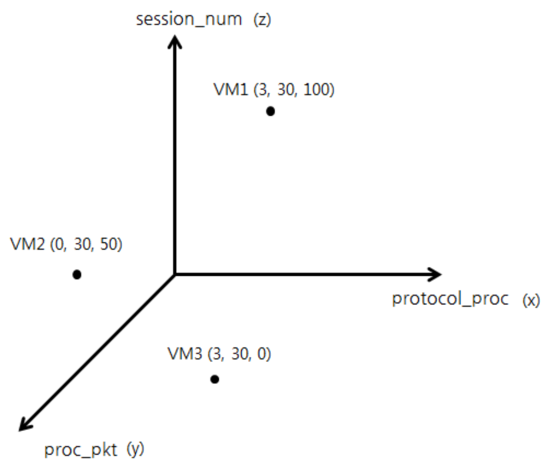


Figure 3. An Example of Virtual Machine Selection

Table 3. An Example for Finding the Optimal VM

VM No.	Distance to the origin	Rank
1	10909	3
2	3400	2
3	909	1

3.2.2. Step of Path Determination

Using a VM set obtained through the previous process, an optimal service-chain path is calculated using Algorithm 2. Table 4 explains the parameters used in the proposed algorithm. Algorithm 2 selects the optimized virtual machines in terms of distances among the virtual machines selected using the previous optimal VM selection algorithm. This thereby creates a path through which a flow should pass. The path of a flow can be generated by means of one of the shortest distance-finding algorithms in a graph, such as Dijkstra's algorithm, Floyd's algorithm, or the Bellman-Ford algorithm. Through one of these algorithms, the shortest path is searched for all pairs that have services in a graph and a logical path. In addition, only a service sequence is substituted with a physical path. First, a logical service path is generated through the optimal VM selection algorithm, and then, a VM set that is compatible with the requirement is obtained using Algorithm 1 (Lines 9 to 10). After this, a VM set that is compatible with the next service chain step is obtained by the same manner as above (Lines 11 to 12). Then, the shortest value between the pairs in the two sets is searched using the table with the shortest distances between nodes, which was made in advance to search the shortest path between two nodes contained in the two sets of VMs. A pair of nodes that has the least distance is selected and put into the logical path (Lines 13 to 22). After this, the VM with the shortest distance between the elements in an optimal VM set is selected using Algorithm 1, and the last node in the currently stored logical path is found and stored again in the logical path. This process continues until all service chains are processed to find VMs compatible with each step in the service chain. This is followed by a search for the shortest path between the nodes (Lines 29 to 30), which can be done using the table identifying the shortest paths of each pair, as produced in the earliest step.

Table 4. Parameters for Algorithm 2

Parameter	Description	Parameter	Description
vm_set	A VM set compatible with the first service chain	select_idx, select_idx2	VM No. that has the least and shortest distance after the SPF result
vm_set2	A VM set compatible with the second service chain	D[][]	Distance between row and col
total_path	Shortest path finally returned	path[]	VM Node list

Algorithm 2

```
INPUT: User requirement vector, VM Info., Network topology
OUTPUT: Service chain path optimized for the user requirements

PROC CHAINING_VM:
Init matrix D[N][N]
Using the shortest distance finding algorithm, search the shortest path for all node pairs

FOR i 0 TO chain length of the flow:
  IF i = 0:
    vm_set   := a VM set compatible with the requirement in the first chain step
  ELSE IF i = 1:
    vm_set2  := a VM set compatible with the requirement in the second chain step
    min_val  := 9999
    min_i    := 0, min_j    := 0
    FOR j 0 TO vm_set.length:
      FOR k 0 TO vm_set2.length:
        IF min_val > D[vm_set[j]][vm_set2[k]]:
          min_val = D[vm_set[j]][vm_set2[k]]
          min_i = j, min_j = k
    path.add(vm_set[min_i])
    path.add(vm_set2[min_j])
    select_idx = vm_set2[min_j]
  ELSE:
    vm_set := a VM set compatible with the requirement
    FOR j 0 TO vm_set.length:
      select_idx2 := index where D[select_idx][vm_set[j]] is the least
    path.add(select_idx2)

for i 0 TO path.length - 1:
    total_path.add(shortest path between path[i] and path[i+1])

RETURN total_path
```

Table 5. Simulation Environment

Category	Description
CPU	Intel Core2 Quad 3.7Ghz
RAM	4GB
OS	Windows 7 Professional
LANGUAGE	Java

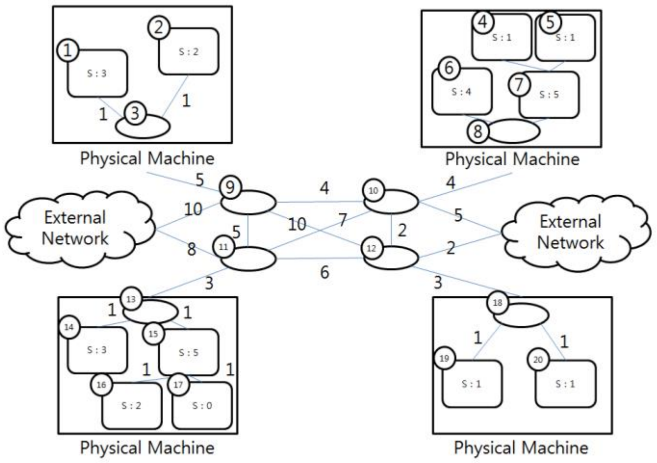


Figure 4. Network Topology for the Experiment

4. Experiment

In this section, an experiment is presented to evaluate the performances of the algorithms proposed in this paper. The experimental environment is shown in Table 5, and a simulator was used to verify the performance.

In this experiment, the times spent on creating a path in two shortest-path algorithms using the proposed algorithm were compared. Figure 4 shows the network environment for the experiment. All the implementations were the same, except for the algorithm used to find the shortest path in the earlier step. The maximum length of the applied service chain was six, while the number of service types was limited to six for each node. Under these circumstances, only the representative types were recorded out of a variety of possible service-chain types.

The experiment results are as follows: the measure of result time value is nano second. Also, each value is average value of many experiments. As shown in Figure 5, as the number of service chains increased, all algorithms showed an increase in overall processing time. The processing time per algorithm showed that the Floyd algorithm was more efficient than the Dijkstra and Bellman-Ford algorithms. This was because the Dijkstra and Bellman-Ford algorithms should find the shortest path for all pairs during single path finding, while the proposed algorithm, including the service-chaining concept, was applied; although, the complexity of Floyd's algorithm was the greatest.

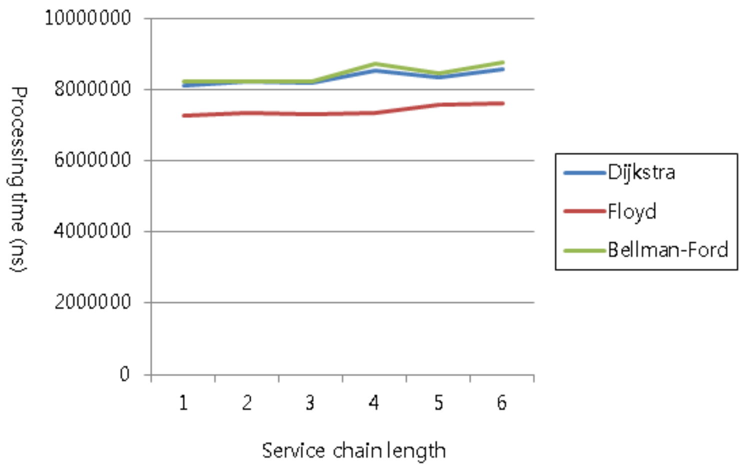


Figure 5. Processing Time *vs.* Service Chain Length

5. Conclusion

This paper proposed a service-chaining method based on finding suitable virtual machines according to user requirements in NFV- and SDN-based network environments in which a large number of virtual machines are dispersed over a network and a single controller manages a number of virtual machines. To select suitable virtual machines, the required elements are proposed. These elements are converted into an N-dimensional space vector where virtual machines are arranged to determine similarities based on the distances to the requested vector, thereby acquiring a virtual machine set that meets the requirements. To generate the optimal service-chaining path, VMs that are compatible with each step in the service chain according to the flow characteristics were found, and a method using the shortest distance-finding algorithm between VMs was proposed.

A future study is scheduled to select optimal virtual machines using more elements that can reflect the network administrator or user requirements more specifically than the current one, and algorithms that can construct an efficient service chain path will be developed.

Acknowledgements

This paper is supported by the Industrial Core Technology Development Program (Next-Generation Communication Network) in the Ministry of Science, ICT & Future Planning [10047541, Development of the auto-defense and auto-scalable SDN smart security networking system using NFV-based service chaining].

References

[1] Asma BEN LETAIFA, Amel HAJI, Maha JEBALIA, Sami TABBANE, State of the Art and Research Challenges of new services architecture technologies: Virtualization, SOA and Cloud Computing", IJGDC Vol. 3, No. 4, **(2010)** December, pp 69-88
[2] ETSI, "Network Function Virtualization (NFV); Prof of Concepts, Framework," **(2013)**
[3] Cisco, "Enabling Service Chaining on Cisco Nexus 1000V Series," White Paper, **(2013)**
[4] Woosik Lee, Hyeonseok O, Namgi Kim, and Yoon-Ho Choi, "Security Service Chaining Mechanism For Preventing Various High-Volume Attack Traffic", **(2014)**
[5] Ruoyan Zhang, I. Budak Arpinar, Boanerges Aleman-Meza, "Automatic Composition of Semantic Web Service", ICWS, **(2003)**, pp 38-41

[6] Xiao Peng and Liu Changsong , "ESCA: Evolution-strategy based Service Composition Algorithm for Multiple QoS Constrained Cloud Applications", IJFGCN Vol. 7, No.1, **(2014)** February, pp 249-260
[7] Jianquiang Hu, Juanzi Li, "Searching and Selecting the Best Web Service Chain" ISECS, Volume 1, **(2008)**, pp 627-630
[8] G. Zayaraz and Poonkavithai Kalamegam, "A test framework based on CPN model for functional testing of Web Service Composition", IJAST, Volume 53, **(2013)** April, pp 135-150

Authors

Hyeon seok Oh is an master grade student at department of Computer Science in Kyonggi university, Suwon, Korea. He received the undergraduate degree from school of computer science, Kyonggi university. S.Korea, in Feb 14, 2013. His research interests include Software-Defined Network, Network Function Virtualization, and so on.

Daeun yu is an master grade student at department of Computer Science in Kyonggi university, Suwon, Korea. She received the undergraduate degree from school of computer science, Kyonggi university. S.Korea, in Feb 21, 2014. Her research interests Wireless Body Area Network and so on.

Yoon-Ho Choi is an assistant professor at department of convergence security in Kyonggi university, Suwon, Korea. He received the M.S. and Ph.D. degrees from school of electrical and computer engineering, Seoul National University, S. Korea, in Aug. 2004 and Aug. 2008, respectively. He was a postdoctoral scholar in Seoul National University from Sep. 2008 to Dec. 2008 and in Pennsylvania State University, University Park, PA, USA, from Jan. 2009 to Dec. 2009. He has served as TPC members in various international conferences and journals. His research interests include Deep Packet Inspection(DPI) for high-speed intrusion prevention, mobile computing security, vehicular network security, big data analysis and so on.

Namgi Kim received the B.S. degree in Computer Science from Sogang University, Korea, in 1997, and the M.S. degree and the Ph.D. degree in Computer Science from KAIST in 2000 and 2005, respectively. From 2005 to 2007, he was a research member of the Samsung Electronics. Since 2007, he has been a faculty of the Kyonggi University. His research interests include sensor system, wireless system, and mobile communication.