Electronic Preprint for Journal of Information Processing Vol.26

Regular Paper

Automatic Construction of Name-Bound Virtual Networks for IoT and its Management

Kenji Fujikawa[1,a)] Ved P. Kafle[1,b)] Pedro Martinez-Julia[1,c)] Abu Hena Al Muktadir[1,d)]
Hiroaki Harai[1,e)]

Received: November 10, 2017, Accepted: June 8, 2018

Abstract: In this paper, we propose a mechanism for automatic configuration of name-bound virtual networks (NBVNs) for Internet of Things (IoT). Generally, IoT devices indicate their correspondent nodes by names. However, current technologies for the construction of virtual networks (VNs) rely on VLANs, IP routing, and OpenFlow control, thus they do not provide a name-based solution. Our proposal fills this gap. We first define business players and their roles for constructing and using the NBVNs. The players are application service provider (ASP), virtual network operator (VNO), and infrastructure provider (InP). Subsequently, we propose a system to automatically construct NBVN. It automatically allocates and assigns the IPv6 addresses required by the network nodes, including IoT devices, of each NBVN. Moreover, it auto-configures the mechanisms for data forwarding and name resolution. Thus, the proposed system constructs area- and/or time-bound VNs for offering network services to event-centric IoT applications, such as outdoor concerts and sporting events. Furthermore, we apply our proposed system to automatically construct wide-area management networks. Finally, we demonstrate that the constructed NBVNs are capable of configuring thousands of addresses and name entries within a minute, thus IoT devices can communicate with each other by using names instead of addresses.

Keywords: NBVN, name-bound, autoconfiguration, area-bound, time-bound, virtual network, IoT, SDN, NFV

1. Introduction

It is estimated that tens of billion devices will be connected in the Internet of Things (IoT) era [1]. The diversity of devices found in IoT includes sensors, cameras, vehicles, and robots, as well as other devices carried by humans such as PCs and smartphones.

A key aspect of IoT devices is that they can be used both outdoors and indoors. For instance, a vehicular network is an outdoor IoT network, in which vehicles equipped with sensors exchange information such as traffic, accidents, and public transportation status. This type of network exists permanently and widely. In contrast, we define a time-bound and area-bound virtual network (VN) for IoT devices, and focus on an automatic construction method of time-/area-bound VNs. They are supposed to be used at outdoor concerts, sporting events and so on. IoT devices communicate with each other by names on the VNs. For these use cases, the current VN configuration methods are insufficient.

Current computer and network virtualization technologies are not capable of delivering functions of automatic addressing, routing and naming. Cloud computing [2] only provides computational resources placed somewhere, but is not able to provide network resources in a specified area. Software Defined Networking (SDN) [3] configures network resources in the specified area. However, it does not automatically construct VNs that support end-to-end communications for IoT devices. In such cases, human network managers have to manually set up the required IP addresses, IP routing and name resolution systems for the VN. Provision of time-bound VNs is difficult without automation of constructing VNs.

Service provider, virtual network operator, virtual network provider, and physical infrastructure provider are defined in Ref. [4], as business players that construct and use VNs. However, it does not provide a clear explanation about the information flow among the players and the tasks performed by a network manager of each player. Presently, the human network managers of the networks owned by such business players must construct the VNs manually. Such operation can take a few days to be completed, so it is impossible for them to provide time-bound VNs. Thus, the interactions among the business players must be clearly defined in order to automate the construction of VNs. Furthermore, the current approaches for VN construction only provide the links as *pipes*, but they do not provide a method for automatically configuring name resolution and data forwarding mechanisms. Therefore, the network manager of each player has to configure them manually.

Naming is a mandatory function for IoT devices to communicate with each other. Several Application Programming Interfaces (APIs) for IoT device communication have been defined at

[1] National Institute of Information and Communications Technology, Koganei, Tokyo 184–8795, Japan
[a)] hudikaha@nict.go.jp
[b)] kafle@nict.go.jp
[c)] pedro@nict.go.jp
[d)] muktadir@nict.go.jp
[e)] harai@nict.go.jp

organizations such as Open Connectivity Foundation (OCF) [5] and Open Mobile Alliance (OMA) [6]. On them, the IoT devices address their correspondent nodes and resources by using their names, also in the Representational State Transfer (REST) protocol used by such APIs. Naming is also important for the network managers of the aforementioned players. Thus, the network manager of each player gives unique names to the network devices that he/she operates. In essence, the network only needs to provide the necessary communication functions for nodes and resources to be specified by names. From this perspective, there is a gap between the required VN and the VN based on the current technologies.

We also focus on management networks for wide areas. Laying physical lines in a wide area costs much, therefore the management channel and the data channel tend to share the same laid lines. Traditionally, the management channel is separately constructed by VLAN on the shared lines. IP addresses are assigned to the network nodes on it manually, or automatically with DHCP. In either case, in order to access to the nodes by names, the network manager must manually prepare a name/address translation table, or manually configure a name resolution system.

In this paper, we propose a method to automatically and instantly deploy a Name-Bound Virtual Network (NBVN) at a given place for a given time to offer network services to event-centric IoT applications, such as outdoor concerts and sporting events. This is proposed in Ref. [7]. We define the Application Service Provider (ASP), Virtual Network Operator (VNO) and Infrastructure Provider (InP) as the business players. We then specify their roles and the tasks for the network manager of them, and propose interactions among them.

We exploit the specification of such interactions in the method for the automation of the NBVN construction. We also propose a simple network layer model to construct multiple NBVNs and distinguish their data packets. The NBVN natively includes a name resolution system and an address assignment mechanism that allow the NBVN nodes to automatically configure their addresses and setup the data forwarding mechanism.

As an extension of our previous work [7], we design a method for applying NBVNs to construct management networks over wide areas. Our proposed system automates most of the construction procedures of the management networks, and requires only very few tasks to be performed manually by the network managers.

Finally, we implement a proof-of-concept experimental system by using Virtual Machines (VMs) running Linux to automatically construct the NBVNs. We validate that the system we implemented constructs area/time-bound NBVNs and enables IoT devices to communicate with each other by using their names. It also enables ASP and VNO to operate network nodes and/or servers in NBVNs by using their names. In addition, a network manager can manage the nodes on the management networks that are mostly automatically constructed.

The reminder of this paper is organized as follows. We first describe limitations for the current approaches for the construction of VNs in Section 2. We then define ASP, VNO and InP, propose their roles and interactions between them, and propose a network layer model of NBVN in Section 3. We describe our design and simple proof-of-concept implementation used to actually construct NBVNs in Section 4. We apply our proposed system to construct management networks in Section 5. We evaluate our system in Section 6. We conclude this paper in Section 7.

2. Limitations of the Current Virtual Network Construction Technologies

In this section, we discuss the limitations of the current technologies from the viewpoint of constructing area/time-bound networks.

2.1 Cloud, Fog and Edge Computing

Cloud computing [2] has been commercially developed such as Amazon Web Services (AWS). In cloud computing, ASP requests resources as needed, which are located somewhere in the Internet. However, when it requires an area/time-bound network, a cloud provider provides computing resources, but does not provide area-bound network resources, such as connectivity and required amount of bandwidth within a certain area. Fog computing [8] and edge computing [9] move a part of computing resources to local area networks and end terminals. Those approaches help constructing area-bound networks, but do not provide network resources either.

2.2 SDN and NFV

Software Defined Networking (SDN) [3] and Network Function Virtualization (NFV) [10] technologies implement Virtual Networks (VNs) over physical networks. OpenFlow is the best known protocol that implements SDN. However, the currently proposed approaches only control flows in the pre-constructed network, or construct only pipes of the network.

Dedicated flows for network services are created over a pre-constructed network in Ref. [11]. Similarly, SDN is used as a traffic engineering tool on a pre-constructed network [12]. They are not construction technologies of VNs, but flow control technologies over pre-constructed networks.

GENI [13] and AutoVFlow [14] construct VNs using OpenFlow. Constructed VNs can be utilized for application service networks over InPs, isolated networks on a campus network, and multi-tenant data centers [3]. They actually construct VNs identified by VLAN IDs, although the proposals do not restrict their applications to VLAN. Thus, those approaches construct only pipes for VNs. For end-to-end communication, a network manager must allocate and assign addressees to network devices over the pipes, configure a data forwarding mechanism, and set up a name resolution system, e.g., DNS. Nevertheless, tasks of network managers are not clearly mentioned in the existing approaches.

So far, the current SDN/NFV approaches only construct VNs where network managers are responsible for setting addressing, forwarding and naming. In addition, as mentioned in the previous section, the interactions between the business players are not clear. Automation of constructing VNs is difficult without a clear definition of the interactions. Thus, provision of time-bound VNs becomes difficult. Our objective is to provide IoT devices with

area/time-bound VNs where addressing, forwarding and naming are automatically configured. The network manager is released from the tasks of these configurations.

3. Proposal of Name-Bound Virtual Networks

For the purpose of providing area/time-bound networks, we propose Name-Bound Virtual Networks (NBVNs) and their dynamic autoconfiguration method [7]. An NBVN is restricted to a certain area and lasts for hours or days. Such NBVNs are very useful to promptly provide IoT applications in area/time-bound outdoor events such as concerts, sports and seasonal festivals.

In this section, we clarify the roles of ASP, VNO and InP when constructing NBVNs. Then, we define the interactions between ASP, VNO and InP. We also define the protocol stacks of NBVNs and show that the protocol stack is simpler than the current VLAN-bound VNs. IoT devices communicate with each other by names on the constructed NBVN. The network manager of VNO only has to assign unique names to network nodes in order to construct the NBVN. He/she does not have to assign IP addresses or VLAN IDs to the nodes, configure a data forwarding mechanism, or configure a name resolution system, which is configured in a server provided by InP.

3.1 Name-Bound Virtual Network Construction Model

We have envisioned a virtual network construction model for future IoT applications [15]. We extend this model to the NBVN construction model (**Fig. 1**).

Infrastructure Providers (InPs) provide physical networks. Edge networks and data centers are connected to a large-scale core network. The edge networks are composed of equipment collecting/processing data from various IoT devices such as PCs, smartphones, vehicles, sensors, and robots. An NBVN is constructed over physical networks of InPs with computational resources (CPU, memory, and storage) and network resources (connectivity, link bandwidth and delay guarantee).

Application Service Provider (ASP) provides network services to IoT devices over the NBVN. However ASP does not know how and from which InP it can obtain the resources that satisfy its request. Thus, ASP sends a request to Virtual Network Operator (VNO) for the NBVN. The request includes information of required computational and network resources.

VNO receives requests from ASP. VNO knows rough locations of computational and network resources, and which InPs can provide them. VNO prepares requests that indicate physical network nodes according to the request from ASP, and sends each request to the correspondent InP. For this, VNO must know node information of InPs. Finally, the NBVN is constructed using physical resources of InPs. VNO manages the NBVN, and delegates the operation of the allocated servers in the NBVN to ASP.

ASP sometimes prepares an NBVN in response to a request from Event Organizer (EO). EO organizes events such as outdoor concerts and sporting events, and wants to make use of NBVNs for them, but is not professional in network management and operation. Currently, preparing VNs for some events is a hard task for EOs. Our proposed system enables EO to easily prepare an NBVN for the event by sending a request to VNO.

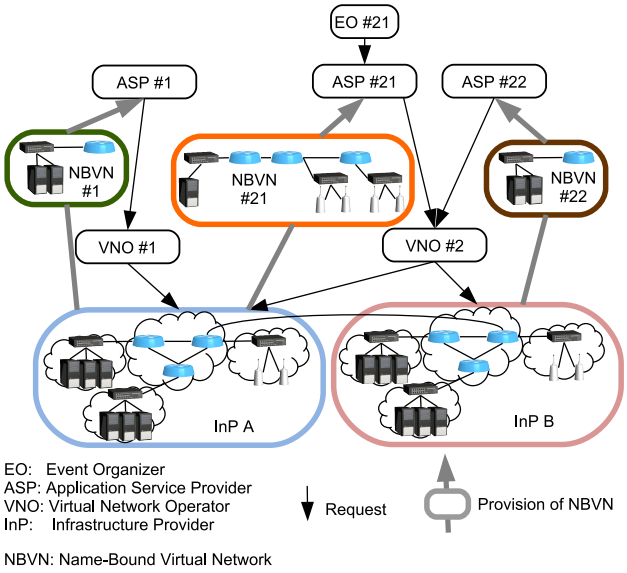

Fig. 1 NBVN construction model.

3.2 ASP/VNO/InP Interactions

We describe interactions between ASP and VNO, and between VNO and InPs. However, we do not specify here any method for interaction between EO and ASP, since we assume that the EO sends its requests to the ASP through Web interfaces, phone, or any other human interaction methods.

In our NBVN construction model, there are network managers who operate NBVN servers, NBVN nodes, and physical servers/nodes in ASP, VNO and InP, respectively. We define dedicated management servers VNO server and InP server for VNO and InP, respectively. Those servers are started in advance before constructing NBVNs. **Figure 2** shows the interactions between the network managers and the servers, and information and requests exchanged between them.

The network manager of ASP sends a request to the VNO server, and operates the allocated NBVN servers. The request includes access point locations, specifications of NBVN servers, network resources, and desired lease duration. The NBVN servers have the functions of naming, addressing and name resolution, or computational and/or storage servers such as Web servers. They are started when an NBVN is being constructed.

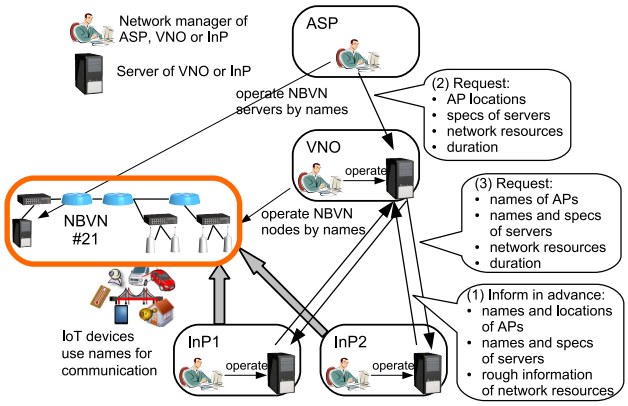

Fig. 2 Interactions between ASP, VNO and InP.

The duration describes when the NBVN service starts and ends.

The network manager of VNO operates a VNO server, and allocates resources for NBVNs through it. The VNO server is informed in advance by the InP about the names and locations of access points and names and specifications of NBVN servers, and rough information of network resources. Here, "rough information" means that the information is a subset of the complete information about available network resources. The VNO server receives the request from the network manager of ASP. Then, it plans to construct the NBVN using resources of multiple InPs, and requests the InPs by including names of access points, NBVN server names and specifications, network resources, and service duration. Subsequently, it sends each of the requests to the correspondent InP.

The network manager of InP maintains the physical network, which consists of network switches and routers as well as access points and servers that are to be used as NBVN servers. He/she also starts an InP server. The InP server informs the VNO server of the rough network information as mentioned above, each time the information is updated (e.g., some network resources are consumed by another VNO server). The request from the VNO includes physical access point names and physical server names, however does not include switch and router names. Therefore, the InP server must search switches and routers in order to construct the NBVN assuring complete connectivity among all the requested access points and servers. It also starts an NBVN node program on each physical node, in order to make the node to be one of the component nodes of the NBVN. The NBVN node program configures interface addresses and forwarding information base (FIB) on each physical node for constructing the NBVN. It also starts a name resolution system on a specified node.

Names configured by the name resolution system are used by VNO and ASP as well as by IoT devices. IoT devices use the names for end-to-end communication. VNO (the network manager and the VNO server) uses the names for NBVN node operation. ASP (the network manager) uses the names for NBVN computational server operation.

3.3 Protocol Stack and Packet Format

Figure 3 shows protocol stacks and packet formats of conventional VLAN-bound VN, our implemented NBVN and essentially-simplified NBVN, respectively.

As for VLAN-bound VNs, VNO constructs a VN with settings of VLAN. The data packets are assigned to the VN by inserting the corresponding VLAN ID into their headers. The network manager of ASP must assign IP addresses to network nodes and configure a name resolution system.

Essentially, in order to implement an NBVN for IoT, the protocol stack and the packet format shown on the right most side in Fig. 3 are sufficient. The data packet includes a field of *domain*, which is used to assign the packet to the corresponding NBVN.

In our implementation, we use the overlay approach. Our proposed system automatically allocates different IP address spaces to different NBVNs, as shown by the middle in Fig. 3. The data packet is categorized into the correspondent NBVN by its address field. For example, two types of packets that include IPv6 addresses starting *2001:db8:1* and *2001:db8:2* in their address fields are distinguished belonging to different NBVNs. Thus, the data packets are distinguished by IPv6 addreses, not by VLAN IDs. Our system also automatically sets up a name resolution system for each NBVN. Therefore, the network managers of ASP and VNO do not have to assign IP addresses to network nodes, configure a data forwarding mechanism, or configure a name resolution system.

Our system enables the different NBVNs to share the same datalink layer without requiring to use distinguishable tags such as VLAN IDs. Therefore, VLAN is not mandatory for the construction of VNs, thus in our implementation, we do not assign VLAN IDs on the datalink layer. However, our system can also run over VLANs by treating VLAN links as logical links for connecting nodes. Moreover, we will describe that configurations of addresses and name resolution systems are not mandatory tasks for the network managers of ASP or VNO.

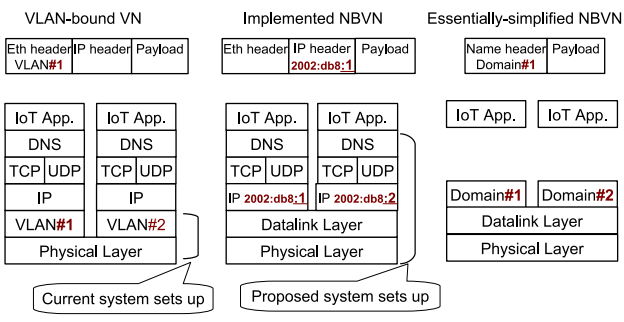

Fig. 3 Packet formats and protocol stacks.

4. Design and Implementation of NBVN Automatic Construction System

In this section, we design and implement the proposed mechanisms in a proof-of-concept (PoC) network. We describe the developed program modules and the procedures for the automatic construction of an area/time-bound NBVN on the PoC network.

4.1 Proof-of-Concept Network

In order to validate the proposed concept of automatic construction of an NBVN, we setup a network as shown in **Fig. 4**. Two InPs, InP1 and InP2, have 11 and 10 network nodes, respectively. Routers are core Layer 3 (L3) routers, and switches are edge L3 switches. All of them have capability of forwarding L3

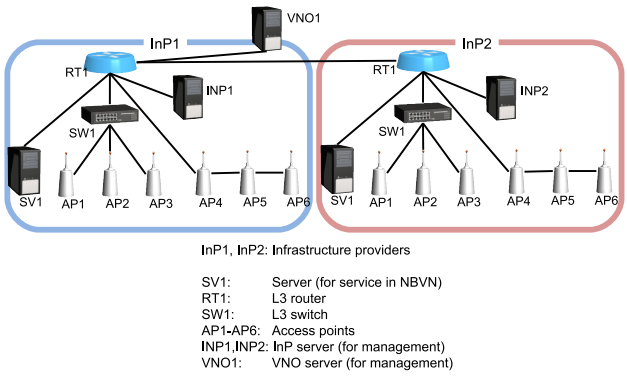

Fig. 4 PoC network.

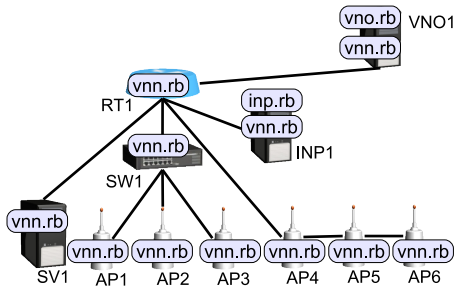

Fig. 5 Developed programs.

packets, i.e., IPv6 packets. Each node has one or more point-to-point links. For example, in InP1, router RT1 is connected to L3 switch SW1, physical server SV1, access point AP4, InP server InP1, and VNO server VNO1 by point-to-point links. Router RT1 in InP1 is also connected to router RT1 in InP2.

4.2 Developed Program Modules

We developed server programs *vno.rb* and *inp.rb* as VNO server and InP server, respectively, in Ruby programming language (**Fig. 5**). They communicate with each other by exchanging JavaScript Object Notation (JSON) [16] format or YAML Ain't Markup Language (YAML) [17] format, according to Representational State Transfer (REST) protocol. JSON is widely-used and human-readable text to transmit data objects in Web applications. YAML has compatibility with JSON, and has better human readability. *Vno.rb* and *inp.rb* also load setting files written in JSON or YAML format. We describe configuration files and exchanged data using YAML format for readability, hereafter.

In addition, we developed an NBVN node program *vnn.rb*. It runs on each physical node for automatically assigning IPv6 addresses to interfaces and building up FIB. For automatic IPv6 address allocation and assignment to their interfaces, we use Hierarchical/Automatic Number Allocation (HANA) protocol [18]. For automatic FIB set up, we use a link-state routing protocol Hierarchical QoS Link Routing Protocol (HQLIP) [19]. Each *vnn.rb* starts HANA/HQLIP protocol software on each physical node. The nodes exchange HANA/HQLIP protocol messages with each other, and assign addresses to their interfaces and set up FIBs.

4.3 NBVN Construction

We assume that ASP receives a request of NBVN for a time/area-bound event such as bike racing from EO. The network manager of ASP compiles the EO's request and generates data that includes the event information in YAML format. The data is contained in a request to a VNO server (i.e., *vno.rb*), which includes access point locations and NBVN server specifications to be used and service duration. Then, it sends the request to the VNO server *vno.rb* according to REST. Here, we assume that the communication channel between the network manager of ASP to *vno.rb* is prepared in advance. Access points, which are supposed to provide Wi-Fi access, are indicated by their locations.

Figure 6 shows an example of ASP's request description along with ASP/VNO interaction described in Section 3.2. NBVN servers, which are supposed to provide Web and video streaming, are also indicated by their computational resources. The network manager of ASP grasps the required locations for the access points, without information which access points are placed in those locations. Similarly, the network manager of ASP grasps the required NBVN server specifications without information which NBVN servers satisfy the required specifications. Duration when the event starts and ends is also specified in the request.

```
asp# cat race1.conf
event: race1
accesspoint:
  - { latitude: 3, longitude: 1 }
  - { latitude: 3, longitude: 2 }
  - { latitude: 2, longitude: 3 }
  - { latitude: 2, longitude: 4 }
  - { latitude: 1, longitude: 5 }
  - { latitude: 1, longitude: 6 }
  - { latitude: 2, longitude: 7 }
  - { latitude: 2, longitude: 8 }
server:
  - { memory: 4G, hdd: 10G }
  - { memory: 4G, hdd: 10G }
duration:
  from: 2016-05-31 06:00:00 +09:00
  to: 2016-05-31 12:00:00 +09:00
asp# curl -X POST --data-binary @race1.conf \
          -H "Content-type: text/x-yaml" \
          http://vno1.somewhere:4000/
asp#
```

Fig. 6 ASP's request.

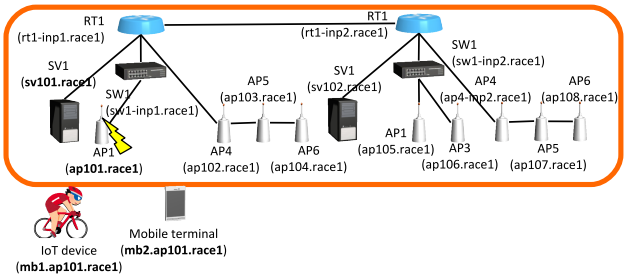

Fig. 7 NBVN for bike race event.

```
sv1.inp1# vnn.rb --vnno 3 --offer eth1 \
                 --nodename sv101 --domain race1
ap1.inp1# vnn.rb --vnno 3 --req eth1 --dhcps eth3 \
                 --nodename ap101
ap4.inp1# vnn.rb --vnno 3 --req eth1,eth2 --dhcps eth3 \
                 --nodename ap102
ap5.inp1# vnn.rb --vnno 3 --req eth1,eth2 --dhcps eth3 \
                 --nodename ap103
ap6.inp1# vnn.rb --vnno 3 --req eth1 --dhcps eth3 \
                 --nodename ap104
                       (a)

rt1.inp1# vnn.rb --vnno 3 --req eth1,eth2,eth3,eth6 \
                 --nodename rt1-inp1
sw1.inp1# vnn.rb --vnno 3 --req eth1,eth2 \
                 --nodename sw1-inp1
                       (b)
```

Fig. 8 *Vnn.rb* command options.

Figure 7 shows an example of NBVN for the bike race event. We design and implement interactions between VNO server *vno.rb* and InP servers *inp.rb*s according to Section 3.2. We refer the reader to Ref. [7] for the detailed information about the implementation.

Inp.rb on each of InPs starts *vnn.rb* on each of the selected nodes with appropriate options, as shown in **Fig. 8** (a), after it receives a request from *vno.rb*. *Inp.rb* defines a domain name as *race1*, which is written in the request. Each node belonging to

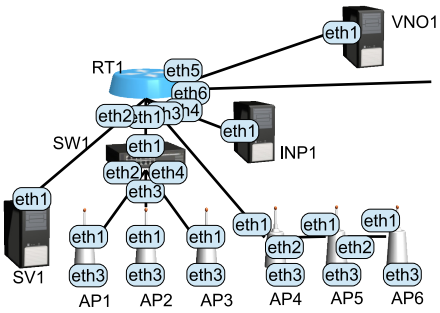

Fig. 9 Interface names in InP1.

InP1 has a name that ends with *.inp1*, e.g., *sv1.inp1* and *rt1.inp1*. The way to name them in InP1 is described in Section 5. *Sv1.inp1*, *ap1.inp1*, *ap4.inp1*, *ap5.inp1* and *ap6.inp1* are indicated in the request coming from *vno.rb*. In the request, *vno.rb* defines the names of the specified nodes as *sv101*, *ap101*, *ap102*, *ap103*, and *ap104*, respectively.

Furthermore, in order to accomplish the complete connectivity in the NBVN, *inp.rb* finds additional nodes, *rt1.inp1* and *sw1.inp1* in this case, according to the Floyd-Warshall algorithm [20], [21]. *Inp.rb* assigns the selected nodes with names like *rt1-inp1.race1* and *sw1-inp1.race1* by just combining the name in InP1 and domain name *race1*, by changing "." into "-", e.g., *rt1.inp1* into *rt1-inp1*, and starts *vnn.rb*s on them, as shown in Fig. 8 (b). This naming convention makes the human network manager of VNO to easily distinguish the event name and the node name in the NBVN. *Vno.rb* preserves the uniqueness of the names by not assigning names ending with an InP name such as *inp1* or *inp2*.

Each *vnn.rb* takes a unique index of the virtual network number (VNNO) by the option *—vnno*. VNNO automatically determines IPv6 address space to be used in the NBVN. For example, *2001:db8:<VNNO>::/48* is determined for the NBVN.

A name resolution server is separately placed in each NBVN. SV1 becomes a name resolution server as well as an NBVN computational server, by the option *–offer*, which is followed by interface names. The option *—domain* specifies domain name of *race1*. Currently, we employ BIND, which is a major DNS software suite. SV1 also becomes a HANA server and allocates IPv6 addresses to the other nodes via the specified interfaces. As for the specified interface names, refer to **Fig. 9**, hereafter. The other nodes become HANA clients and request address spaces for their interfaces by option *—req*, which is followed by the interface names. SV1 registers DNS entries of *rt1.inp1.race1*, *sw1.inp1.race1*, *sv101.race1*, *ap101.race1*, *ap102.race1*, *ap103.race1* and *ap104.race1*. The name of each node in the NBVN is defined by combining the domain name and either of the name defined in InP1 or the name specified by VNO. The latter five node names are specified by VNO, and are passed to *vnn.rb* with the option *–name*.

Table 1 shows the addresses assigned to the interfaces of each node. The address is constructed as *2001:db8:<VNNO>:<link ID>::<node ID>*. Here, the link ID and the node ID are automatically determined and assigned by HANA. The link ID is a unique number in each NBVN, which is dynamically assigned to each point-to-point link. Thus, two end interfaces of adjacent nodes share the same value of the link ID. The node ID is also a unique ID in each NBVN, which is dynamically assigned to each node.

Table 1 Names and addresses of bike race NBVN.

	eth1	eth2	eth3	eth6
rt1.inp1.race1	*X*:3:1::1	*X*:3:2::1	*X*:3:3::1	*X*:3:4::1
sw1.inp1.race1	*X*:3:1::2	*X*:3:5::2	–	–
sv101.race1	*X*:3:2::3	–	–	–
ap101.race1	*X*:3:5::4	–	*X*:3:6::4	–
ap102.race1	*X*:3:3::5	*X*:3:7::5	*X*:3:8::5	–
ap103.race1	*X*:3:7::6	*X*:3:8::6	*X*:3:9::6	–
ap104.race1	*X*:3:8::7	–	*X*:3:a::7	–

(IPv6 prefix of 2001:db8 is replaced by *X* for readability.
For instance, *X*:3:1::1 means 2001:db8:3:1::1.)

In addition, each access point starts a DHCP server with an allocated address space. For example, *ap101.race1* starts a DHCP server with an address space of *2001:db8:3:6::/64*. This is executed by the option of *–dhcps* followed by an interface name, e.g., *eth3*. Here, each access point has *eth3* as a wireless interface (Fig. 9). It provides wireless access with SSID containing domain name *race1*, which is delivered by HANA protocol. Mobile terminals and IoT devices search SSID containing *race1*, and connect to NBVN *race1*. According to DHCP, access point *ap101.race1* assigns IP addresses and DNS names such as *mb1.ap101.race1* and *mb2.ap101.race1* to mobile terminals and IoT devices that are connected to *ap101.race1*. In the current implementation, the HANA sever node (*sv101.race1*) registers 200 of these names for each access point to its DNS entries by default, when it allocates address spaces to access points for DHCP.

Our proposed system automatically sets up FIB in each node owned by InP. It can distinguish packets belonging to different NBVNs, since different NBVNs use different address spaces. We show a setting example in Section 6.

As described so far, our proposed system automatically constructs time/area-bound NBVNs for IoT devices. IoT devices use automatically-assigned names for communication with each other. ASP uses the names for managing Web or streaming servers. VNO uses the names for managing the NBVN nodes.

5. Proposal of Applying NBVNs to Management Networks

Before an InP server *inp.rb* communicates with each of the nodes in the InP and starts a *vnn.rb* on it, a management channel between the InP server and the other nodes must be constructed. In closed areas such as data centers, it is easy to prepare management physical lines that are separated from the data lines. However in wide areas, which are the primary target fields of our research, the management channel and the data channel tend to share the same laid physical lines, since laying them costs much. Therefore, in this paper, we newly apply NBVNs to construct InP management networks on the shared lines. The InP server can communicate with the other nodes by names on the constructed InP management NBVN, which provides addressing, routing and name resolution functions. If VLAN is applied to construct the management network instead, the network manager must manually configure those functions.

Similarly, we apply an NBVN to construct the VNO/InPs management network, which is to be used for the communication be-

tween VNO and InPs.

These management networks are constructed for the automation of the other NBVNs. They are not automatically constructed according to the proposed procedure described in Section 3.2. The network managers must specify network nodes and interfaces, and assign names to the nodes. However, L3 functions such as addressing, forwarding and name resolution, are still automatically configured.

5.1 Requirements and Conditions of NBVNs for ASP and the Management NBVNs

We summarize the requirements and conditions of NBVNs for ASP and the management NBVNs, and clarify the differences between the two types of NBVNs, before we describe the construction procedure of the management NBVNs.

The requirements of NBVNs for ASP is to construct an NBVN that provides addressing, routing and name resolution functions. The NBVN is used for the ASP to make an access to the access points and the servers, and is used for IoT devices to communicate with each other by names.

Conditions are that the network manager of ASP knows locations where access points are to be placed and server specifications, without information about names of access points and servers and their connections. InP management networks and VNO/InPs management network are constructed in advance.

In order to construct the NBVN for ASP, the network managers ASP makes a request to the VNO server *vno.rb* via generic communication means such as the Internet. Then, *vno.rb* makes a request to each of the InP servers *inp.rb*s via the VNO/InPs management NBVN. Subsequently, each *inp.rb* starts a *vnn.rb* on the nodes in the NBVN for ASP via each InP management NBVN (see Section 4.3 in detail).

As for the requirements of InP management NBVNs, the network manager of InP and *inp.rb* can access the network nodes inside his/her InP. As for the requirements of VNO/InPs Management NBVNs, the *vno.rb* can access each of *inp.rb* in InPs.

As for conditions for constructing the management NBVNs, the network managers of InP and/or VNO must assign names to the network nodes, and collect names of the interfaces to be used for the NBVN.

In order to construct the management NBVNs, the network managers of InP and/or VNO separately execute a *vnn.rb* on each network node with the names of the interfaces (see details below in Section 5.2 and Section 5.3).

5.2 Construction of InP Management NBVNs

A *vnn.rb* starts on each node in InP1 with specifying interfaces, VNNO of 1 and domain name *inp1* in order to construct the InP management NBVN, as shown in **Fig. 10**.

In the actual field, we suppose that a *vnn.rb* should be executed at the time each node is powered on, with specifying interfaces that should be shared to data channels. Even in this case, as shown in Fig. 10, before powering on the nodes, the network manager only has to assign a node name and the shared domain name to each node. He/she does not have to assign addresses or configure a name resolution system. Our proposed system considerably reduces the tasks of the network manager.

```
inp1# vnn.rb --vnno 1 --name inp1 --offer eth1 \
                                      --domain inp1
rt1# vnn.rb --vnno 1 --name rt1 --req eth1,eth2,\
                                          eth3,eth4,eth5
sw1# vnn.rb --vnno 1 --name sw1 --req eth1,eth2,\
                                          eth3,eth4
ap1# vnn.rb --vnno 1 --name ap1 --req eth1
ap2# vnn.rb --vnno 1 --name ap2 --req eth1
ap3# vnn.rb --vnno 1 --name ap3 --req eth1
ap4# vnn.rb --vnno 1 --name ap4 --req eth1,eth2
ap5# vnn.rb --vnno 1 --name ap5 --req eth1,eth2
ap6# vnn.rb --vnno 1 --name ap6 --req eth1
sv1# vnn.rb --vnno 1 --name sv1 --req eth1
vno1# vnn.rb --vnno 1 --name vno1 --req eth1
```

Fig. 10 *Vnn.rb* command options in InP1.

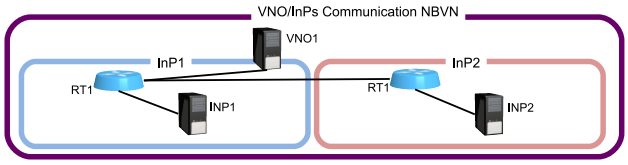

Fig. 11 VNO/InPs management NBVN.

```
vno1.inp1# vnn.rb --vnno 2 --name vno1 --offer eth1 \
                                        --domain vno1
inp1.inp1# vnn.rb --vnno 2 --name inp1 --req eth1
rt1.inp1# vnn.rb --vnno 2 --name rt1-inp1 --req eth4,\
                                            eth5,eth6
inp2.inp2# vnn.rb --vnno 2 --name inp2 --req eth1
rt1.inp2# vnn.rb --vnno 2 --name rt1-inp2 --req eth4,eth6
```

Fig. 12 *Vnn.rb* command options for VNO/InPs management NBVN.

The network manager does not necessarily have to assign VNNO. This is because different InPs are allowed to use the same number of VNNO, since the InP networks are separated from each other. Thus, the VNNO is fixed to 1 in all the InPs. As a result, the same IPv6 address space is used in them. For this, the network interface that is connected to another InP must not be specified in the *vnn.rb* command options. In Fig. 10, *eth6* is not specified, which is connected to InP2.

5.3 Construction of VNO/InPs Management NBVN

The VNO/InPs management NBVN (**Fig. 11**) is constructed for the communication between VNO server *vno.rb* and InP servers *inp.rb*s. VNO receives information and sends a request from/to InPs on the NBVN, as depicted in Fig. 2. For the network construction, both of InP1 and InP2 network managers start *vnn.rb*s on InP1 and InP2 management NBVNs, respectively (**Fig. 12**).

These commands are executed on the nodes of the both InPs, therefore, nodes of *vno1.inp1*, *inp1.inp1* and *rt1.inp1* in InP1, and on nodes of *inp2.inp2* and *rt1.inp2* in InP2. When the network manager of InP1 logs in inp1.inp1, then he/she can log in vno1.inp1 and rt1.inp1. He/she starts *vnn.rb*s on them via the constructed InP1 management NBVN in Section 5.2. Similarly, the network manager of InP2 starts *vnn.rb*s on *inp2.inp2* and *rt1.inp2* nodes in InP2. Each of the network managers must select names that are different from node names used in another InP, e.g., *rt1-inp1* and *rt1-inp2*. *Vnn.rb*s on *rt1-inp1* and *rt1-inp2* are executed with network interfaces that are connected to each other.

As a result, *vno1.vno1*, *inp1.vno1*, *rt1-inp1.vno1*, *inp2.inp2*, and *rt1-inp2.vno1* are assigned to the nodes. The name space ends with vno1, and is completely separated from the name space

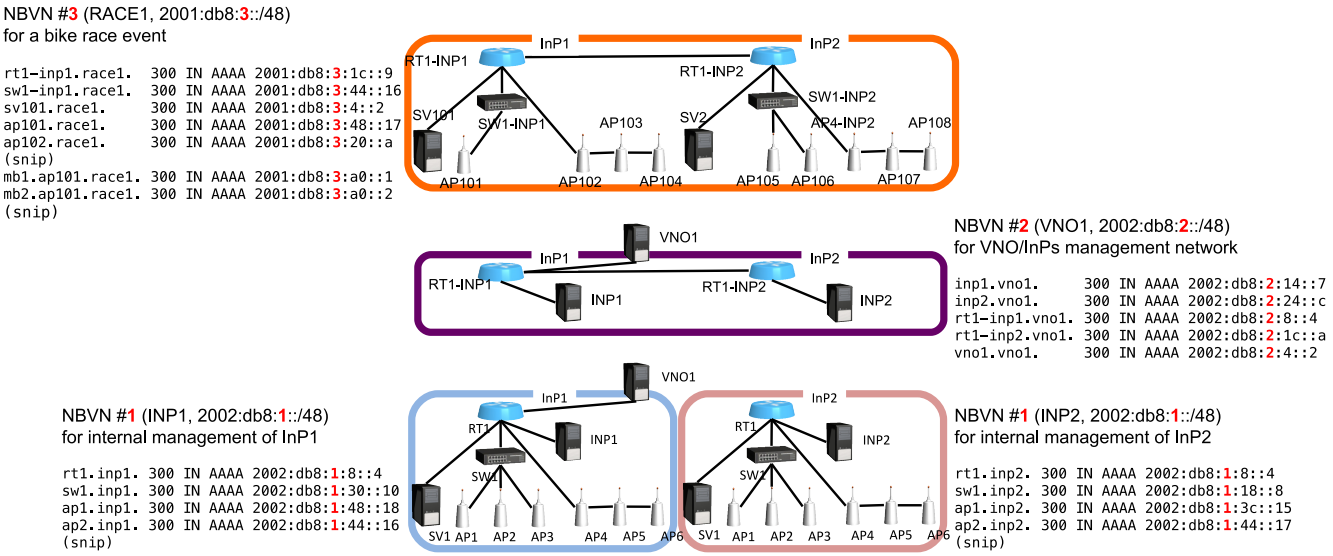

Fig. 13 Names and IPv6 addresses in NBVNs.

of InP management NBVNs, which end with inp1 and inp2, respectively.

The network managers of InPs must know the interfaces in InPs that connect to each other, and specify those interfaces, as shown in Fig. 12. Here, *eth6* of *rt1* in InP1 and *eth6* of *rt1* in InP2 are included in the command options.

6. System Evaluation

We used two physical machines to validate the operation of the proof-of-concept network. Two InPs, InP1 and InP2, are constructed on two physical machines. They have 11 and 10 virtual machines (VMs), respectively. Each of the network nodes shown in Fig. 4 is constructed in a VM. Each physical machine is equipped with dual Intel Xeon X5670 (2.93 GHz/6 core), 64 GBytes memory, and two GbE network interfaces. All the VMs are connected to a single L2 switch using one of the GbE interfaces. All the point-to-point links depicted in Fig. 4 are defined by Generic Routing Encapsulation (GRE) tunnels.

Figure 13 shows the registered names and IPv6 addresses in InP1 management NBVN, VNO/InPs management NBVN and race event NBVN. The name resolution servers (DNS servers) are executed at *inp1*, *vno1*, and *sv1* in InP1, respectively, on which each *vnn.rb* is executed with the option *–offer*. InP1 and InP2 use the same address space *2001:db8:1*, since the nodes are executed with the same VNNO of 1. Note that the DNS entry for one node has only one line in Fig. 13 for simplicity. However, the DNS entry in the actual system can have multiple lines, which correspond to IPv6 addresses assigned to the various interfaces of the node.

Figure 14 shows the interface address and FIB on *rt1*. The interface addresses for the race event NBVN, which start with *2001:db8:3* as shown in Table 1, are assigned to GRE interfaces *greth1*, *greth2*, *greth3* and *greth6*. The domain name and VNNO are *race1* and 3, respectively. Similarly, interface addresses that start with *2001:db8:2* are assigned to *greth4*, *greth5* and *greth6* for the VNO/InPs management NBVN, of which domain name and VNNO are *vno1* and 2, respectively. These interfaces are connected to InP1 server and VNO1 server, respectively. Interface addresses that start with *2001:db8:1* are assigned to GRE interfaces from *greth1* to *greth5* for the InP1 management NBVN, of which domain name and VNNO are *inp* and 1, respectively.

```
rt1.inp1$ ip -6 addr show
(snip)
4: greth1@NONE: <BROADCAST,MULTICAST,UP,LOWER_UP>...
    inet6 2001:db8:3:8::9/64 scope global
    inet6 2001:db8:1:8::4/64 scope global
5: greth2@NONE: <BROADCAST,MULTICAST,UP,LOWER_UP>...
    inet6 2001:db8:3:4::9/64 scope global
    inet6 2001:db8:1:24::4/64 scope global
6: greth3@NONE: <BROADCAST,MULTICAST,UP,LOWER_UP>...
    inet6 2001:db8:3:20::9/64 scope global
    inet6 2001:db8:1:1c::4/64 scope global
7: greth4@NONE: <BROADCAST,MULTICAST,UP,LOWER_UP>...
    inet6 2001:db8:2:8::4/64 scope global
    inet6 2001:db8:1:14::4/64 scope global
8: greth5@NONE: <BROADCAST,MULTICAST,UP,LOWER_UP>...
    inet6 2001:db8:2:10::4/64 scope global
    inet6 2001:db8:1:58::4/64 scope global
9: greth6@NONE: <BROADCAST,MULTICAST,UP,LOWER_UP>...
    inet6 2001:db8:3:1c::9/64 scope global
    inet6 2001:db8:2:c::4/64 scope global
(snip)
rt1.inp1$ ip -6 route show
(snip)
2001:db8:1:3::/64 via ap4.inp1 dev greth3...
2001:db8:1:20::/64 via ap4.inp1 dev greth3...
(snip)
2001:db8:2:3::/64 via inp1.vno1 dev greth4...
2001:db8:2:18::/64 via rt1-inp2.vno1 dev greth6...
(snip)
2001:db8:3:3::/64 via ap102.race1 dev greth3...
2001:db8:3:24::/64 via ap102.race1 dev greth3...
(snip)
```

Fig. 14 Interface addresses and FIB on *rt1*.

As shown in Fig. 14, the routing entries that start with *2001:db8:3* are registered for the race event NBVN. Similarly, the routing entries that start with *2001:db8:2* are registered for the VNO/InPs management NBVN. Routing entries that start with *2001:db8:1* are configured for the InP1 management net-

```
ap108.race1$ traceroute6 ap101.race1
traceroute to ap101.race1 (2001:db8:3:48::17) from...
 1 ap107.race1 (2001:db8:3:70::24) 0.162 ms...
 2 ap4-inp2.race1 (2001:db8:3:54::1c) 0.187 ms...
 3 rt1-inp2.race1 (2001:db8:3:68::21) 0.293 ms...
 4 rt1-inp1.race1 (2001:db8:3:1c::9) 0.337 ms...
 5 sw1-inp1.race1 (2001:db8:3:44::16) 0.316 ms...
 6 ap101.race1 (2001:db8:3:48::17) 0.227 ms...
(snip)
vno1.vno1$ traceroute6 inp2.vno1
traceroute to inp2.vno1 (2001:db8:2:24::c) from...
 1 rt1-inp1.vno1 (2001:db8:2:8::4) 0.255 ms...
 2 rt1-inp2.vno1 (2001:db8:2:1c::a) 0.307 ms...
 3 inp2.vno1 (2001:db8:2:24::c) 0.306 ms...
(snip)
inp1.inp1$ traceroute6 ap6.inp1
traceroute to ap6.inp1 (2001:db8:1:54::1c) from...
 1 rt1.inp1 (2001:db8:1:8::4) 0.208 ms...
 2 ap4.inp1 (2001:db8:1:1c::9) 0.11 ms...
 3 ap5.inp1 (2001:db8:1:3c::13) 0.173 ms...
 4 ap6.inp1 (2001:db8:1:54::1c) 0.134 ms...
```

Fig. 15 Traceroute in each NBVN.

Table 2 Times to construct NBVNs.

	DNS entries	IP addresses	Construction time of NBVN (sec)	Average time of starting one node (sec)
Race event NBVN	15 (+1600)	36 (+1600)	34.15	2.28
VNO/InPs management NBVN	5	8	13.29	2.66
InP1 management NBVN	11	20	26.19	2.38
InP2 management NBVN	10	18	23.27	2.38

work. Each of the rest parts of the addresses consists of link ID and node ID. These values can be different from each other even in the same interface.

Figure 15 shows the results of traceroute6 commands from *ap108.race1* to *ap101.race1* in race event NBVN, from *vno1.vno1* to *inp2.vno1* in InP1 management NBVN, and from *inp1.inp1* to *ap6.inp1* in VNO/InPs management NBVN. All the packets are forwarded by L3 forwarding.

These configurations prove that our proposed system automatically configures IP addresses and FIB on each NBVN node, and that data packets that belong to different NBVNs are distinguished and forwarded accordingly, as shown in Fig. 3.

We counted the numbers of configured DNS entries and IPv6 addresses on the constructed NBVNs, as shown in **Table 2**. 15 DNS entries for network nodes, and 1600 DNS entries for mobile terminals and/or IoT devices are automatically configured. 36 IPv6 addresses for network nodes, and 1600 IPv6 addresses for mobile terminals and/or IoT devices are automatically configured. An IPv6 address is assigned to each link of each network node, thus, the number of IPv6 addresses is larger than that of DNS entries.

We also measured the construction time of the NBVN. The construction time starts from the time when InPs start *vnn.rb* on the first network node to the time when they start *vnn.rb* on the last network node. In the current system, InPs sequentially start *vnn.rb*s on the nodes. For example, it took 34.15 seconds to start all the 15 nodes for the race event NBVN. Average time of starting one node is 2.28 seconds.

These results show that our proposed system constructs an NBVN that consists of 15 network nodes with a name resolution system within one minute. The names of mobile terminals and IoT devices are pre-set in DNS, thus they can communicate with each other by names. Furthermore, the names of NBVN servers and NBVN nodes for operations of ASP and VNO are automatically configured. These prove that our proposed system practically constructs NBVNs that are used for area/time-bound events.

We discuss the scalability of our proposed system. The number of nodes in the actual infrastructure may reach millions. However, the number of nodes that are required for each NBVN only reaches hundreds, since the NBVN is used for an area-bound event, and the number of attendees may be in the order of tens of thousands. The total starting time of an NBVN is proportional to the number of NBVN nodes. Therefore, it is expected to take tens of minutes to construct the NBVN for the event of tens of thousands of attendees from our PoC network experiments.

We suppose an event lasts for hours or days, thus construction of the NBVN within tens of minutes is practical. Our system processes the requests in the manner of first come first serve. Therefore, the system blocks the process of a newly-arrived request until it finishes the process of another request. Consequently, the process of the request may be postponed for tens of minutes. However, this is not a problem for practical use, since the network manager of ASP has only to send the request tens of minutes or a few hours before the event.

Previous approaches based on VLAN and OpenFlow such as GENI [13] and AutoVFlow [14] did not mention L3 addressing, forwarding and name resolution. Therefore, we cannot compare our proposed NBVN to those approaches in terms of the total VN construction times including the configuration and operation of L3 functions. However, we have proven through experiments that the proposed NBVN system can construct VNs including L3 functions with labor for management equal to or less than that required by the previous approaches, which do not include the L3 functions.

In order to construct a VN by the previous approaches, the network manager has to specify access points and servers, and assign a VLAN ID to them or the L2 edge switches that are directly connected to them [*1]. In NBVN, the network manager of ASP specifies access point locations, server specifications, and an event name. VNNO is automatically assigned by VNO (*vnn.rb*).

7. Conclusion

In this paper, we have proposed an automatic construction mechanism of name-bound virtual networks (NBVNs) to be used in IoT. We have defined ASP, VNO, and InP as the business players behind the whole operation and exploitation of NBVNs. We have also clarified the roles of ASP/VNO/InP and the tasks to be performed by their network managers, also proposing the required interactions among them. We have developed a poof-of-concept system that implements the operations of ASP/VNO/InP, and automatically constructs NBVNs.

[*1] In addition, the network manager should assign the VLAN ID to the trunk L2 switches for management and/or security reasons in general. However, here we compare our approach to the others by ignoring this operation.

In the NBVNs, the required IPv6 addresses are automatically allocated to the network nodes and IoT devices, and the data forwarding and name resolution mechanisms are also automatically configured. Thus, the system is able to provide both area-bound and time-bound event-oriented NBVNs to IoT applications such as outdoor concerts and sporting events. In our experimental deployment, thousands of addresses and name entries are automatically configured on an NBVN within a minute, allowing IoT devices to communicate with each other by their names. ASP/VNO only need names to operate the servers, switches and routers present in their NBVNs. Furthermore, our system constructs the name-bound management networks for wide areas with requiring very few tasks for the network managers.

For future work, we will define, design and implement methods for the InP to inform the VNO of the network resources such as bandwidth, delay and CPU, in addition to access point locations and the server memory/storage resources. We will also design a mechanism of VNO's probing whether the requested NBVN is to be satisfied with provisioned resources by InPs, before the actual allocation request. Moreover, we will apply our system to construct actual area-/time-bound IoT networks.

References

[1] Gubbi, J., Buyya, R., Marusic, S. and Palaniswami, M.: Internet of Things (IoT): A vision, architectural elements, and future directions, *Future Generation Computer Systems*, Vol.29, No.7, pp.1645–1660 (2013).

[2] Mell, P. and Grance, T.: The NIST definition of cloud computing (2011), available from ⟨https://www.nist.gov/programs-projects/cloud-computing⟩.

[3] Bakshi, K.: Considerations for software defined networking (SDN): approaches and use cases, *2013 IEEE Aerospace Conference*, pp.1–9 (2013).

[4] Schaffrath, G., Werle, C., Papadimitriou, P., Feldmann, A., Bless, R., Greenhalgh, A., Wundsam, A., Kind, M., Maennel, O. and Mathy, L.: Network virtualization architecture: Proposal and initial prototype, *Proc. 1st ACM Workshop on Virtualized Infrastructure Systems and Architectures*, pp.63–72 (2009).

[5] Open Connectivity Foundataion, available from ⟨https://openconnectivity.org⟩.

[6] Open Mobile Alliance, available from ⟨http://www.openmobilealliance.org/⟩.

[7] Fujikawa, K., Kafle, V.P., Martinez-Julia, P., Al Muktadir, A.H. and Harai, H.: Automatic Construction of Name-Bound Virtual Networks for IoT, *IEEE Computer Software and Applications Conference* (*COMPSAC 2017*), pp.529–537 (2017).

[8] Bonomi, F., Milito, R., Zhu, J. and Addepalli, S.: Fog computing and its role in the Internet of things, *Proc. 1st edition of the MCC Workshop on Mobile Cloud Computing*, pp.13–16 (2012).

[9] Garcia Lopez, P., Montresor, A., Epema, D., Datta, A., Higashino, T., Iamnitchi, A., Barcellos, M., Felber, P. and Riviere, E.: Edge-centric computing: Vision and challenges, *ACM SIGCOMM Computer Communication Review*, Vol.45, No.5, pp.37–42 (2015).

[10] Hawilo, H., Shami, A., Mirahmadi, M. and Asal, R.: NFV: State of the art, challenges, and implementation in next generation mobile networks (vEPC), *IEEE Network*, Vol.28, No.6, pp.18–26 (2014).

[11] Dong, M., Kimata, T. and Zettsu, K.: Service-controlled networking: Dynamic in-network data fusion for heterogeneous sensor networks, *2014 IEEE 33rd International Symposium on Reliable Distributed Systems Workshops*, pp.94–99 (2014).

[12] Akyildiz, I.F., Lee, A., Wang, P., Luo, M. and Chou, W.: Research challenges for traffic engineering in software defined networks, *IEEE Network*, Vol.30, No.3, pp.52–58 (2016).

[13] Berman, M., Chase, J.S., Landweber, L., Nakao, A., Ott, M., Raychaudhuri, D., Ricci, R. and Seskar, I.: GENI: A federated testbed for innovative network experiments, *Computer Networks*, Vol.61, pp.5–23 (2014).

[14] Yamanaka, H., Kawai, E. and Shimojo, S.: A technique for full flow virtualization of multi-tenant OpenFlow networks, *Computer Networks*, Vol.102, pp.1–19 (2016).

[15] Miyazawa, T., Kafle, V.P. and Harai, H.: Reinforcement Learning Based Dynamic Resource Migration for Virtual Networks, *IFIP/IEEE International Symposium on Integrated Network Management* (2017).

[16] Introducing JSON, available from ⟨http://www.json.org/⟩.

[17] YAML: YAML Ain't Markup Language, available from ⟨http://yaml.org/⟩.

[18] Fujikawa, K., Tazaki, H. and Harai, H.: Inter-AS Locator Allocation of Hierarchical Automatic Number Allocation in a 10,000-AS Network, *2012 IEEE/IPSJ 12th International Symposium on Applications and the Internet* (*SAINT2012*), pp.68–73 (2012).

[19] Fujikawa, K., Harai, H., Ohmori, M. and Ohta, M.: Quickly Converging Renumbering in Network with Hierarchical Link-State Routing Protocol, *IEICE Trans. Information and Systems*, Vol.99, No.6, pp.1553–1562 (2016).

[20] Floyd, R.W.: Algorithm 97: shortest path, *Comm. ACM*, Vol.5, No.6, p.345 (1962).

[21] Warshall, S.: A theorem on boolean matrices, *J. ACM*, Vol.9, No.1, pp.11–12 (1962).

Kenji Fujikawa received his M.E. and Ph.D. degrees in Informatics, Kyoto University, Japan, in 1995 and 2000, respectively. After completing graduate school, he became Assistant Professor in the Graduate School of Informatics, Kyoto University in 1997, Senior Researcher at ROOT Inc. in 2006, and joined National Institute of Information and Communications Technology in 2008. His research topic is hierarchical routing and autoconfiguration of network. He is a member of IEICE, IPSJ and IEEE.

Ved P. Kafle received his B.E. in Electronics and Electrical Communications from Punjab Engineering College (now PEC University of Technology), India, an M.S. in Computer Science and Engineering from Seoul National University, South Korea, and a Ph.D. in Informatics from the Graduate University for Advanced Studies, Japan. He is currently a senior researcher at National Institute of Information and Communications Technology (NICT), Tokyo, and concurrently holding a visiting associate professor position at the University of Electro-Communications, Tokyo. He has been serving as a Co-rapporteur of ITU-T Study Group 13 since 2014. His research interests include new network architectures, naming and addressing, machine-to-machine communication, Internet of things (IoT), and privacy, security management in networks. He received the ITU Association of Japan Encouragement Award and Accomplishment Award in 2009 and 2017, respectively, and two Best Paper Awards (second prize) at the ITU Kaleidoscope Academic Conferences in 2009 and 2014. He is a senior member of IEEE.

Pedro Martinez-Julia received his B.S. in Computer Science from the Open University of Catalonia, an M.S. in Advanced Information Technology and Telematics and a Ph.D. in Computer Science from the University of Murcia, Spain. He is currently a full-time researcher at National Institute of Information and Communications Technology (NICT), Tokyo. His main expertise is in network architecture, control and management, with particular interest in overlay networks and distributed systems and services. He has been involved in EU-funded research projects since 2009, leading several tasks/activities, and participating in IETF/IRTF for the standardization of new network technologies. He has published more than twenty papers in referred conferences and journals. He is a member of ACM and IEEE.

Abu Hena Al Muktadir received his B.Sc. (Honors) and M.Sc. degrees from the University of Rajshahi, Bangladesh in 2004 and 2005, and a Ph.D. degree from The University of Electro-Communications, Tokyo, Japan in 2014, with all degrees majoring in Information and Communication Engineering. He is currently working as a full-time Researcher at the National Institute of Information and Communications Technology (NICT), Japan. He worked as a full-time Lecturer at Daffodil International University, Bangladesh from 2007 to 2009. His research focuses on network resource management using Game theory and machine learning, Internet of Things (IoT), network design, routing, and network coding. He has published more than twenty-five papers in referred conferences and journals. He is a member of IEEE.

Hiroaki Harai received his M.E. and Ph.D. degrees in Information and Computer Sciences from Osaka University, Japan in 1995 and 1998, respectively. He is currently a Director at National Institute of Information and Communications Technology (NICT), Tokyo, Japan, where he is leading Network Science and Convergence Device Technology Laboratory for the research and development of new network architecture and optical networks. He received the Outstanding Young Researcher Award from IEEE ComSoc Asia-Pacific Region in 2007. He also received the Young Researcher Award from the Ministry of Education, Culture, Sports, Science and Technology (MEXT), Japan in 2009. He is a member of IEEE.