Network Platform Enabling Flexible Services Composition

Yoshinori KITATSUJI

Development of cloud technology enables service developers to build service systems in a simple and flexible manner. To gain the similar efficacy to the network service systems, technologies of constructing virtualized networks and realizing simple and flexible construction network services are rapidly developed. This paper introduces the research and development of realizing a network platform enabling a flexible network services composition.

1 Information

The Internet has grown into an infrastructure supporting social activities and economic activities. However, it is difficult to introduce new functions, and there is a need to fundamentally redesign its architecture. To do this, work to revise its architecture is proceeding in Japan as the New-Generation Network[1], and in the U.S. as the NSF's Future Internet Design (FIND) initiative[2]. Until now, multiple novel architectures have been proposed: Content Centric Networking (CCN)[3], ID/locator separation[4], etc. To evaluate these architectures, large scale experiments are required, and multiple testbeds are being constructed in Japan, the U.S. and Europe: JGN-X[5], Global Engineering for Network Innovation (GENI)[6], etc.

In a testbed, multiple architecture experiments are executed at the same time, which requires computing resources and network resources such as communication links and router CPUs to be independently allocated to each experiment, with guarantees that the experiments won't interfere with each other.

Network virtualization technology enables independent resource allocation, so it is used in many testbeds. In such testbeds, "slices" (virtual networks comprised of virtual routers and communication links) are allocated to each experiment, and an experimenter (hereinafter called a "user") can create his/her own programs on a virtual router, so various architectures can be implemented on the slices.

Although testbeds based on conventional virtualization technologies provide network management functions such as allocation and release of resources, sufficient programming environments are not provided. Consequently, this R&D implemented a flexible programming environment (hereinafter "platform") that enables programming in which even users without much network knowledge can reuse previously developed program modules, like combining toy blocks. This paper gives an overview of network virtualization, then describes this platform's design and implementation.

2 Network virtualization overview

There are hopes that network virtualization technology can enable large-scale experiments. Network virtualization provides virtual networks called slices, comprised of virtual routers and virtual links. The slices share with other slices their physical nodes (called "virtualization nodes") and links; the resources allocated to each slice are guaranteed by the virtualization platform (a platform network comprised of virtualization nodes and physical links, which provides the slices).

To achieve network virtualization, it is important that the network satisfies requirements, such as resource separation, scalability, and performance, and is extendable, programmable and secure, with management functions[7]. Resource separation is the most important requirement, guaranteeing the exclusive use of virtual links and CPUs of virtual routers allocated to slices. If users can create programs executed on virtual routers, this creates vulnerabilities in the virtual network, so enhanced security is required. Scalability that enables the execution of multiple experiments at the same time is important, and OpenFlow is sometimes used to enable the multiplexing of multiple slices in physical links[8].

Programs executed in virtual routers can enable packet flow, and can also process packets, but this requires high performance, as mentioned in the previous paragraph.

Moreover, user programs must be allocated and executed in multiple virtual routers, so how to allocate them in virtual routers is also important.

Among the requirements mentioned above, the platform in this paper focuses on scalability, programmability, and management of functions that comprise network services. It also has characteristics that support the development and allocation of programs executed on the slices, and which enables the configuration of slices on multiple virtualization platforms.

3 Services design environment

3.1 Outline

The platform created by this research and development aims to support the development of programs that execute on virtual routers provided in the virtualization platform, and experiments and demonstrations of the programs[9]. It is not efficient for the users to develop new architectures from scratch, so the platform enables the users to create modules of protocol processes like x-kernel[10] and Click Modulator[11], and boosts the productivity of program development through the reuse of smaller modules. Specifically, the smallest modules that can execute on virtual routers are called blocks, and the platform provides a program development environment in which the blocks can be combined like toy blocks.

As shown in Fig. 1, this platform is comprised of a services design environment, a services allocation environment, and a slice exchange point. A conventional Click Modulator, etc. has a block group executed with a single host and router, however this platform has a block group on multiple virtual routers. Below, the group of these routers that execute one architecture is called a "service." To enable experiments on a global scale, the slice exchange point provides mutual connectivity with virtualization platforms developed outside Japan, and slices can be constructed over multiple virtualization platforms.

3.2 Blocks

To handle packet processes such as calculating checksums and resending packets, Click Modulator modules are provided as blocks, as described below.

(1) User level Click blocks are blocks developed in accordance with Click, and are executed in the user space.

(2) Kernel level Click blocks are blocks developed in accordance with Click, and can only be executed on a kernel level Click driver.

(3) User process blocks can be provided as Click modules, and user developed programs can also be provided as blocks.

Note that low layer packet processes would be developed as user level and/or kernel level Click blocks. On the other hand, upper layer protocol processes can be programs developed using various programming languages, executed as blocks in the user space.

An input/output interface between the blocks is defined as a port. The port is defined as a set of attribute values such as name and role. Between two blocks, only the ports which have a name and all attribute values matching can

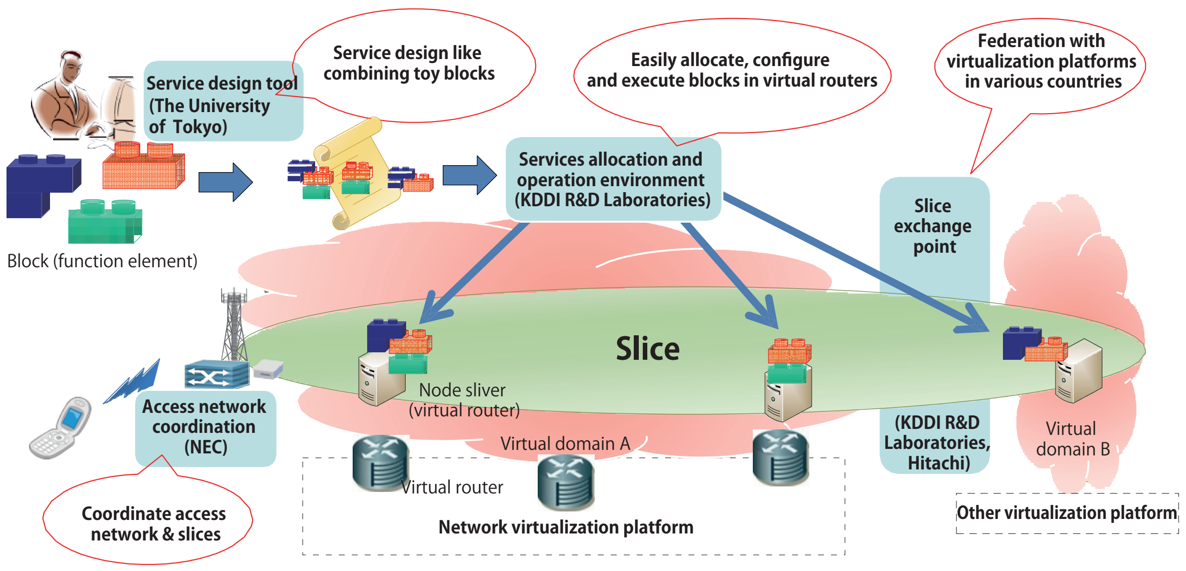
Service design like combining toy blocks
Easily allocate, configure and execute blocks in virtual routers
Federation with virtualization platforms in various countries
Service design tool (The University of Tokyo)
Services allocation and operation environment (KDDI R&D Laboratories)
Block (function element)
Slice exchange point
Slice
Node sliver (virtual router)
Access network coordination (NEC)
Virtual domain A
Virtual router
(KDDI R&D Laboratories, Hitachi)
Virtual domain B
Coordinate access network & slices
Network virtualization platform
Other virtualization platform

Fig. 1 Platform overview

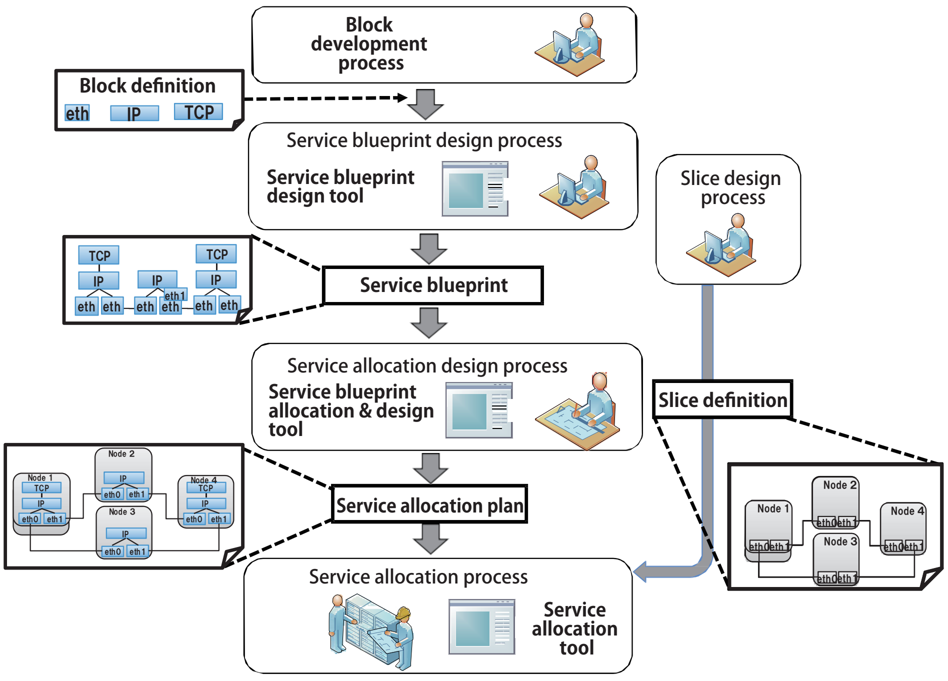
Block development process
Block definition
eth IP TCP
Service blueprint design process
Service blueprint design tool
Slice design process
Service blueprint
Service allocation design process
Service blueprint allocation & design tool
Slice definition
Service allocation plan
Service allocation process
Service allocation tool
Node 1
Node 2
Node 3
Node 4

Fig. 2 Service development and allocation process

connect. The ports are used to connect two blocks that are in the same virtual router or are running in different virtual routers. For example, the blocks that execute an HTTP protocol client and server would be defined as HTTP (client, …), HTTP (server, …). Ports are described in XML, but the details are omitted here.

The virtual routers use Linux OS, so currently the blocks described above must be in programs that can run on Linux.

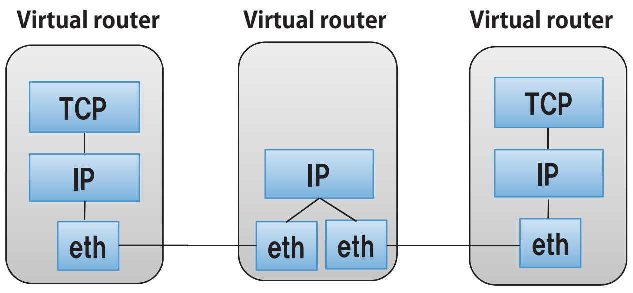

Fig. 3 Service blueprint example

3.3 Service blueprint design tool

First, before developing a service according to the process in Fig. 2, the user develops each block, or searches for a program developed by another person and prepares block definitions so this platform can handle those blocks. A block definition is comprised of the block name, and the port definitions.

Next, the user designs the service blueprint with the prepared block group. The service blueprint is a service reference, so instead of planning specific virtual routers, this combines the minimum block group to comprise the service (service development on this platform is equivalent to creating a service blueprint).

Figure 3 shows an example of a service blueprint that defines a TCP/IP service. It connects 3 virtual routers that are an IP router and send/receive hosts. The router has IP and Ethernet blocks; the hosts have TCP, IP and Ethernet blocks. The service blueprint can be written in an XML file, and can also be created by using a service blueprint design tool with a GUI.

3.4 Service allocation design tool

A service blueprint does not specify details on individual implementations such as which virtual router the block executes on, which virtual routers to connect, etc. Therefore, the user first defines a slice comprised of virtual routers and virtual links, in accordance with the virtualization platform's agreement. Next, the user creates a service allocation plan for virtual routers in the service blueprint by designating in the slice definition, with the service allocation design tool, the virtual routers on which to execute. The service allocation plan allocates a block group to a virtual router group and, according to this plan, installs the block group on the virtual routers to enable execution of the service.

4 Service design environment

4.1 Physical configuration

In accordance with the service allocation plan, the “service allocation environment” is the name for functions that allocate a block group to virtual routers, and start and stop the service. As shown in Fig. 4, this environment is comprised of a virtual router controller executed on a virtual router, and a service control node that sends commands to allocate, execute and stop the block group. The service control node is implemented as a Linux based machine different from virtual routers; virtual routers are connected to the virtualization platform by an independent physical network (control plane).

There is only one service control node for a virtualization platform. It sends commands to allocate, execute and stop all services executed on the slice created on the virtualization platform (that is, block groups for services).

4.2 Physical configuration

Programs called the service controller and virtual router controller are executed on the service control node and virtual router, respectively. They are connected by TCP/IP communications on the control plane, and provide functions such as to allocate, execute and stop the service.

A) Slice creation

The user sends a slice definition via the service control node to the virtualization platform, and creates the slice. As a result, a secure shell server process is launched in the virtual router created (the service controller is not directly involved in this function; it is executed by the virtualization platform).

B) Service allocation function

According to the service allocation plan and slice definition received from the user, the service controller decides the virtual router that allocates the block, and obtains from the virtualization platform the virtual router’s IP access and TCP port number on the control plane. After that, it sends an executable file (specifically, a Debian Linux package) that combines the block group into one, via sshd to the virtual router controller on the virtual router. After that, the virtual router controller installs the received executable file into the virtual router.

C) Service launch and stop

After installing the required executable file in the service, the service controller can instruct the virtual router controller to launch/stop the executable file installed in the virtual router according to the user’s designations. The virtual router controller launches/stops the block group (executable file) that corresponds to one virtual router. For example, if the block group that is executed on a virtual router is all user level Click blocks, then a user level Click driver is launched/stopped. Or, if the user process blocks and a user level Click driver are mixed together, it launches/stops both.

5 Service example

Aiming to verify the service allocation environment we developed, in order to have an upper layer protocol using user process blocks we developed a video transmission service. This video transmission service was separately implemented by modularizing a program implemented on Linux[12] as a user process block, as shown in Fig. 5. The service uses TCP blocks, IP blocks and Ethernet blocks to comprise the load balance block (BL), content repository block (BC) and transcode block (BT). The video playing

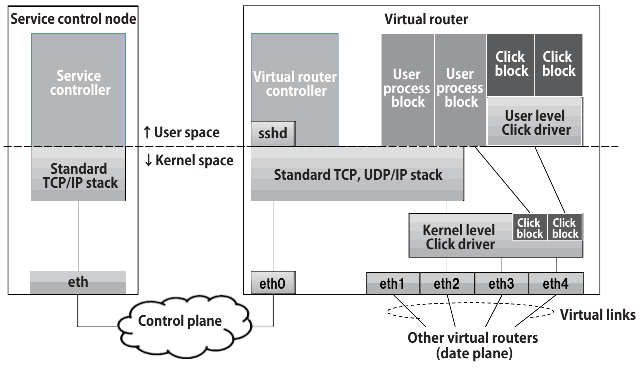

Fig. 4 Implementation of service allocation environment

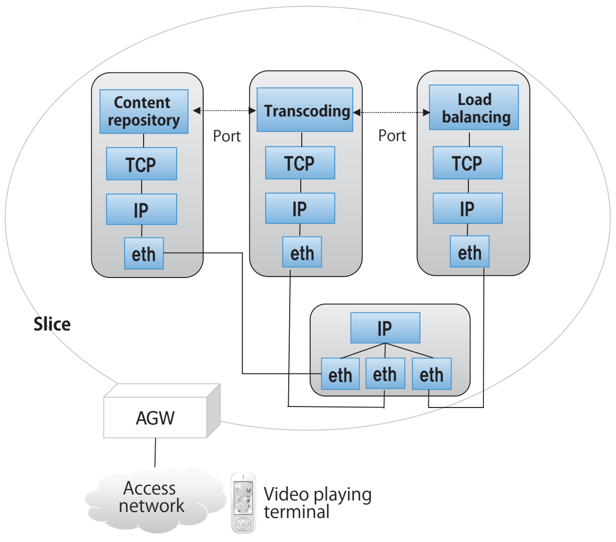

Fig. 5 Blocks configuration of video transmission service

Table 1 Port definition of each block

Blocks	Port definitions	Blocks to be connected
B_L	socket_stream (c)	TCP/IP blocks
	HTTP_sig_content_notify (c)	B_T
B_C	socket_stream (c)	TCP/IP blocks
	HTTP_sig_request_media (s)	B_T
B_T	socket_stream (c)	TCP/IP blocks
	HTTP_sig_content_notify (s)	B_L
	HTTP_sig_request_media (c)	B_C

terminal is outside the slice, so the slice is delivered via the access gateway (AGW) provided by the virtualization platform. Blocks at or below TCP use the standard TCP/IP stack in the virtual router's kernel (Linux). Table 1 shows the port definitions of these blocks.

In this service, first, the content repository that received the transmission request sends to the video playing terminal the content with a resolution that suits the bandwidth of its access network. The transmission request and content transmission are done via the socket_stream port that connects with the standard TCP/IP stack (here, letters c and s show the client and server roles). When there is congestion in the access network, the video playing terminal sends to the load balance block a request to reduce (transcode) the resolution of the content, and that block sends a request via the HTTP_sig_content_notify port to the transcoding block, for it to start transcoding. On the other hand, the transcoding block sends via HTTP_sig_request_media to the content repository block a request to change the content's destination from the video playing terminal to that same transcoding block. As a result, content sent from the content repository block is transcoded at the transcoding block, and then sent to the video playing terminal.

Previously developed programs for content reporting, etc. can be processed on this platform merely by writing the block definitions, and this service can be executed on the virtualization platform in a short time.

6 Mechanism for integrated management between virtualization platforms

6.1 Outline

To execute services over multiple virtualization platform architectures to extended areas, we implemented coordination between slices on different virtualization platforms. Such coordination is called "federation." The federation implements functions between virtualization platforms with different architectures to construct and monitor multiple slices on different virtualization platforms from one of the platforms. To do this, we developed a Slice Exchange Point (SEP)[13] architecture that coordinates control planes and data planes between the virtualization platforms, and a common application programmable interface (API) to provide service construction and monitoring functions between virtualization platforms.

6.2 SEP architecture

Figure 6 shows an outline of the SEP architecture. The SEP architecture is comprised of centrally placed control and management functions (SEP core), and functions that mediate between the virtualization platform ("domain" in the figure) and the SEP core. SEP has the following two broad goals. First, SEP makes it possible for the user to use his typically used control functions provided by one of multiple virtualization platforms, to control and manage the entire slice deployed on multiple virtualization platforms. SEP also enables unrestricted use of unique functions that other virtualization platforms have.

To achieve these two goals, it is desirable that SEP has the following four features.

I. Independence from virtualization platforms
Architecture in which the control and management mechanism does not rely on a specific virtualization platform.

II. Single interface
Can use a single interface to coordinate each virtualization platform and SEP.

III. Abstractness
Extract the common capabilities and features of the resources and functions of multiple virtualization platforms, and handle them by control in SEP.

IV. Extendibility of functions
Corresponding to the extension of individual functions of the virtualization platform, the same virtual infrastructure and SEP interwork can be extended.

The gatekeeper and gateway have the following functions.

● Gatekeeper (GK)
Converts the control plane. Specifically, mutually converts each virtualization platform's unique command/release definitions and the SEP core's command/release definitions.

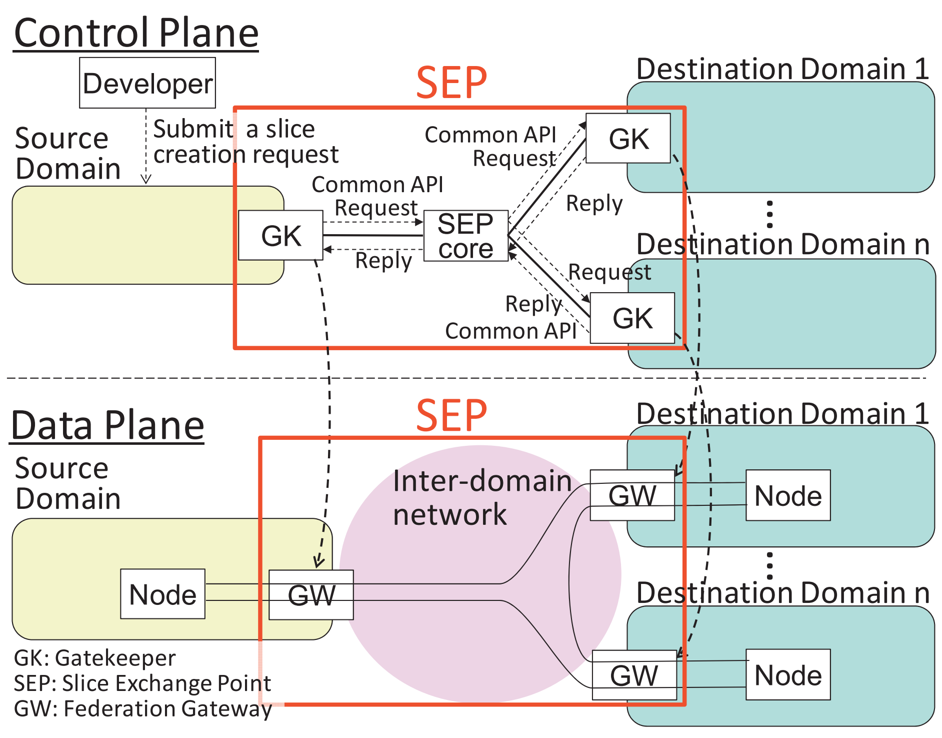

Fig. 6 SEP architecture outline

●Gateway (GW)

Relays the data plane. Converts the format (protocol, network parameters, etc.) of packets between the virtualization platform and SEP network. GW is controlled by the GK.

In the figure, "Source Domain" shows the virtualization platform that originates the slice control (create, update, delete). "Destination Domain" shows the virtualization platform that receives control from the Source Domain.

6.3 Common slice definition

Figures 7 and 8 show that in SEP, one controls and manages his/her own slice definitions (entire slice definitions) that do not rely on the virtualization platform to which the SEP connects. We assume that the status transitions of the entire slice of each virtualization platform that connects to SEP, and of individual slices (virtual routers and links), are as shown in Figs. 7 and 8, and the status transitions of a common slice flow are shown as in Fig. 7. Implementations in each virtualization platform could use status transitions that differ from Figs. 7 and 8, and in that case, each virtualization platform's GK would absorb the differences in the status transition.

6.4 Data plane

A virtual link that links virtual routers belonging to

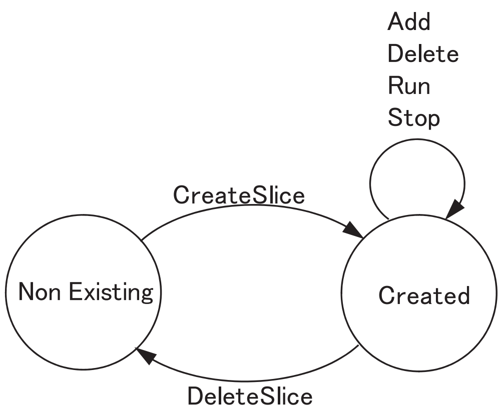

Fig. 7 State transitions of slice

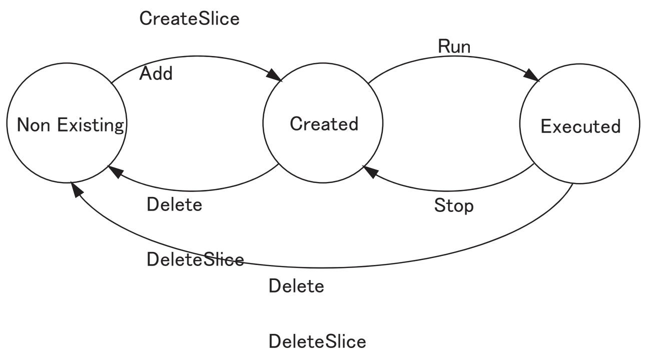

Fig. 8 State transitions of virtual resources (virtual routers and links)

different virtualization platforms (virtual link between virtualization platforms) is comprised of a virtual link in a virtualization platform and a virtual link that links the virtualization platforms (Fig. 9). These are connected by a GW. A virtual link between virtualization platforms is logically one link.

In SEP, one can freely decide the parameters (link type, MAC address, VLAN number, etc.) used for a virtual link between virtualization platforms, not dependent on implementation method of the network in each virtualization platform. In this platform, parameters used in a virtual link between virtualization platforms are decided by the two methods described below.

(a) Negotiation between GK

The GK manages the links of the virtualization platform. The GK of each virtualization platform negotiates the parameters through the signal process that is the common slice's construction request and its response. For example, the GK decide a usable VLAN number, etc.

(b) Decision by SEP core

The SEP core manages the link of the network (SEP network) that constitutes the virtual link between virtualization platforms, and sets the parameters required in the common slice. It also plays the role of notifying to the virtualization platform that requested the slice composition, sending a notice with the parameters of the virtualization platform that receives the slice composition request.

7 Demonstration experiment

7.1 Outline

For service allocation, execution and federation, a demonstration was done at three locations: in Japan, the U.S. and Europe. Its outline is described here.

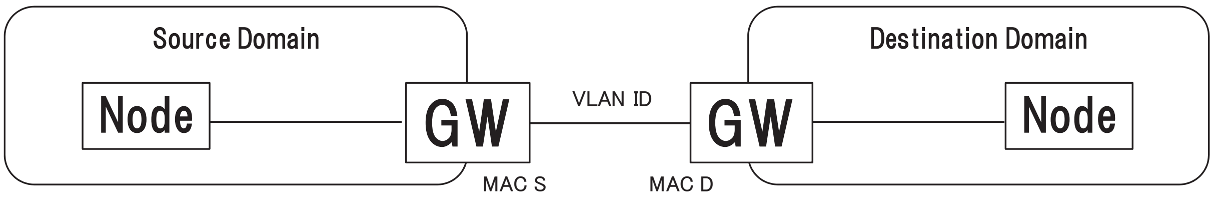

Fig. 9 State transitions of virtual resources (virtual routers and links)

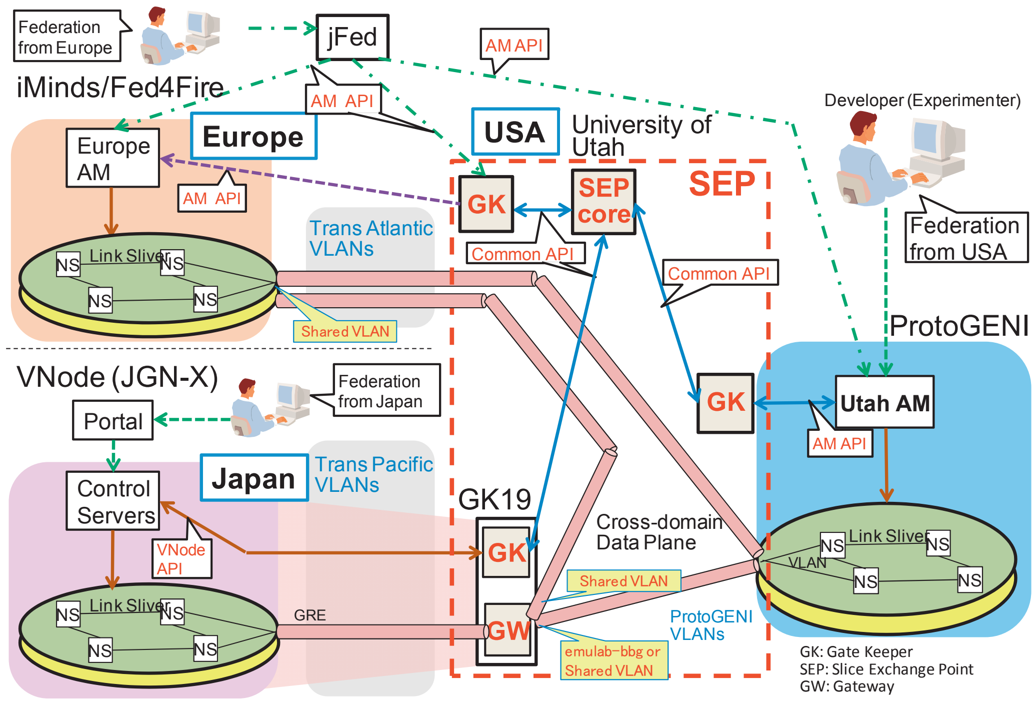

Fig. 10 Overall configuration diagram of an experiment system for federation between three locations: Japan, USA, and Europe

7.2 Federation demonstration between three locations: Japan, USA, and Europe

Figure 10 shows the federation system configuration. These three virtualization platforms have management mechanisms that independently construct slices: VNode virtualization platform (NICT), ProtoGENI virtualization platform (The University of Utah, USA), and Virtual Wall 2 virtualization platform (iMinds, Belgium). Virtual Wall 2 uses ProtoGENI's Aggregate Manager-API (AM-API) as its management mechanism, but in federation from ProtoGENI using SEP in the direction of Virtual Wall 2 and VNode, this operation goes from the slice creation and control commands of ProtoGENI's AM-API, then passes through SEP, and again the GK on the Virtual Wall 2 side reconverts it into an AM-API for Virtual Wall 2.

The control plane is implemented by three GK (one for each virtualization platform) and the SEP core. The SEP core and GKs of VNode and ProtoGENI and Virtual Wall 2 are each implemented by a server installed at The University of Utah.

The SEP core takes a common slice settings (configuration) command requested from one virtualization platform, sends the command to two virtualization platforms, merges the reply returned from those two virtualization platforms, and returns it as one common reply to the originally requesting virtualization platform.

We confirmed that in the connection environment shown in Fig. 10, the slice shown in Fig. 11 is configured, and we can achieve federation between virtualization platforms with different implementation methods. With this SEP architecture, by functions to convert commands and slice definitions, we achieved a single developer interface and independence from the virtualization platform, and enabled the developer to manage (create, control) the entire slice the developer created by federation, via the virtualization platform control functions that the developer is accustomed to using normally. We thereby demonstrated that it is possible to configure a slice that exceeds the limits of resources (virtual routers and links) of one virtualization platform.

8 Conclusion

This paper describes a platform that enables the construction of flexible network service systems, in which even a user with little knowledge of the network can do programming by combining previously developed program modules—like putting toy blocks together.

With the aim of developing a program that is executed on virtual routers provided by the virtualization platform, and to support demonstration, this platform provides programming development in which the protocol processes and various functions of the network service system are made into modules, and the smallest units (blocks) executable on a virtual router are combined like toy blocks to boost the productivity of program development by reusing modules.

The demonstration experiment (federation between Japan/USA/Europe) described in Section **7** was performed live during the IEICE Technical Committee on Network Virtualization (March 16–17, 2015 in Koganei, Tokyo), and at GEC22 held March 23–26, 2015 in Washington, D.C. This R&D NICT was performed as the NICT contract "Research and Development of Network Virtualization Platform Technology to Support the New-Generation Network, Issue B, Research and Development of a Network

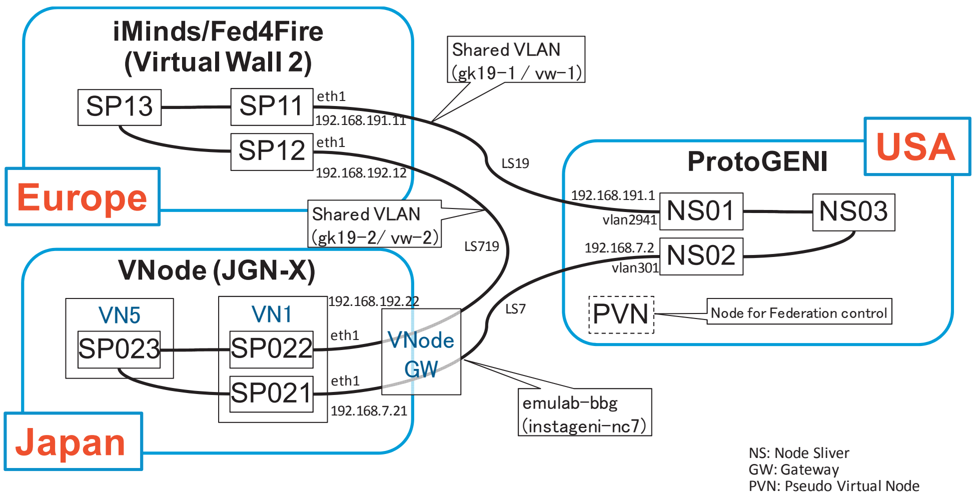

Fig. 11 Slice configured between 3 locations: Japan, USA, and Europe

Platform that Can Create Services," done jointly by the Interfaculty Initiative in Information Studies at The University of Tokyo, NEC Corporation, Hitachi, Ltd., and KDDI R&D Laboratories Inc.

References

1 Generation Network Promotion Forum, http://forum.nwgn.jp/english/

2 FIND Initiative, http://www.nets-find.net/

3 V. Jacobson, D. Smetters, J. Thornton, M. Plass, N. Briggs, and R. Braynard,. "Networking Named Content," Proceeding of ACM CoNEXT'09, pp.1–12, Dec. 2009.

4 Hiroaki Harai, Kenji Fujikawa, Ved P. Kafle, Takaya Miyazawa, Masayuki Murata, Masaaki Ohnishi, Masataka Ohta, and Takeshi Umezawa, "Design Guidelines for New Generation Network Architecture," on Communications, Vol.E93-B, No.3, pp.462–465, March 2010.

5 http://www.jgn.nict.go.jp/english/index.html

6 GENI (Global Environment for Network Innovations), http://www.geni.net/

7 A. Nakao, "Network Virtualization as Foundation for Enabling New Network Architectures and Applications," IEICE Transactions on Communications, Vol. E93B, Issue 3, pp.454–457, March 2010.

8 N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling innovation in campus networks" ACM SIGCOMM Computer Communication Review, Vol.38, Issue 2, pp.69–74, April 2008.

9 Yamada, "Network KASOUKA KIBAN NI OKERU KASOU Network KANRI SEIGYO GIJUTSU," IEICE Technical Committee on Network Virtualization (NV), Document for the 4th domestic conference, July 2012.(in Japanese)

10 http://www.ieice.org/~nv/nv201207-05-yamada.pdf

11 N. Hutchinson and L. Peterson, "The x-kernel: An architecture for implementing network protocols," IEEE Transactions on Software Engineering, Vol.17, Issue 1, pp.64–76, January 1991.

12 Komorita, Ito, Yokota, Makaya, and Falchunk, " Services Composition based on Next-Generation Service Overlay Networks Architecture," MoMuC, Journal of IEICE Mobile Multimedia Communications Committee, 2011-9, pp.87–92, Sept. 2011.

13 OKAMOTO, MATSUMOTO, KUROKI, MIYAMOTO, OGAKI, and HAYASHI "Network resource control and management technologies for the federation and New Generation services," Technical Report IEICE Technical Report, IN 2012-91, pp.87–91, Oct. 2012 (in Japanese).

Yoshinori KITATSUJI, Ph.D.

Leader, Mobile Network Group, KDDI R&D Laboratories
Network Architecture, Mobile Network