Federation-less-federation of Network-virtualization Platforms

Yasusi Kanada and Toshiaki Tarui
Hitachi, Ltd., Central Research Laboratory
Yokohama, Japan
{*Yasusi.Kanada.yq, Toshiaki.Tarui.my*}*@hitachi.com*

Kei Shiraishi
Hitachi, Ltd., Telecommunications & Network Systems Division
Kawasaki, Japan
shiraishi_kei@itg.hitachi.co.jp

***Abstract* – A method for federating multiple network-virtualization platforms by creating and managing slices (virtual networks) is proposed. A cross-domain slice can be created, deleted, or modified by sending a slice specification to the domain controller (network manager) of one domain. The specification is then propagated to other domains. Two challenges were addressed while this method was developed. The first challenge is to enable federation among multiple domains that do not support federation functions by only adding a few components without modification of the existing network-virtualization-platform architecture. A domain-dependent specification of a slice, containing a pseudo virtual node that encloses a part of the slice specification in the other domains, is used, and this part is handled by a proxy node that represents another domain and a control component that implements a federation API to create a cross-domain slice. The second challenge is to enable manageable non-IP (arbitrary-format) data communication on a cross-domain slice. For an inter-domain communication, underlay VLAN parameters including MAC addresses are negotiated in advance and data packets on a slice are tunneled between gateways in these domains. The proposed federation method was implemented on two network-virtualization platforms, federation between two homogeneous domains was successfully demonstrated, federation performance was measured, and several issues on functional restrictions and implementation difficulty were found.**

I. INTRODUCTION

In Japan, several projects targeting "new-generation networks" (NwGN) have been conducted [Aoy 09] [AKA 10]. These projects aim to develop new network protocols and architectures (i.e., the "clean slate" approach [Fel 07]) as well as various applications that are difficult to run on internet protocols (IPs) but work well on NwGNs. The Virtualization Node Project (VNP) [Nak 10] and its successive projects aim to develop network-virtualization technology and virtualization nodes (VNodes). The goal of these projects are to develop an environment where multiple slices (or virtual networks) with independently designed NwGN architectures and functions using arbitrary-format (non-IP and non-Ethernet) packets run concurrently, but are logically isolated, on a physical network. The virtualization platform developed by VNP [Nak 12b][Nak 12a] have been deployed in the testbed network JGN-X [Pan 11] for designing, deploying, and testing new network services in Japan.

In this study, a method for federating virtualization platforms is proposed. This method enables both homogeneous and heterogeneous federation; that is, creating a slice across two or more network-virtualization platforms of the same or different type. Slices contain virtual nodes and virtual links, so both virtual-node and -link resources should be managed. Especially, the virtual links between the platforms, which might not be managed by the platforms, should be managed.

Two challenges were addressed while this federation method was developed. The first challenge is to enable federation among multiple domains that do *not* support federation functions without modification of their existing network-virtualization platform architecture. Instead, a proxy node, which communicates with a controller component as a normal virtualization network node, is added to mediate between the domains. The slice structure of one domain is hidden from the slice specification of another domain. This "federation-less federation" reduces the hurdle of introducing a federation function into existing virtualization platforms; that is, it reduces the development cost and time and amount of resources required for developing federation functions as well as the number of bugs, degree of performance degradation, and number of service interruptions caused by deploying federation functions.

The second challenge is to enable non-IP (arbitrary-format) data communication on a cross-domain slice. For an inter-domain communication, underlay VLAN parameters including VLAN ID and MAC addresses are negotiated in advance, and data packets on a slice are tunneled between gateways in these domains. Not only VLAN ID but also MAC addresses must be negotiated because no address resolution or correlation mechanism, such as the Address Resolution Protocol (ARP), is assumed when a non-IP (clean-slate) protocol is used for data communication.

This federation method has been implemented on the VNode platform in a post-VNP project, and successful communication between two homogeneous domains was demonstrated.

The rest of this paper is organized as follows. Section II summarizes the virtualization platform, the slice model, and the slice-creation and management method, which were previously developed. Section III describes the basic federation method, Sections IV outlines federating federation-less platforms, and Section V describes the federation-less-federation architecture and the method of transforming slice specifications. Section VI describes the method of creating inter-domain links, which allows communication using non-IP protocols, and the data format used for inter-domain data communication. Sections VII to VIII describe the implementation and evaluation of the federation method as well as related work, and Section IX concludes this paper.

II. VIRTUALIZATION PLATFORM AND SLICE MODEL

This section explains network virtualization, the virtualization platform, the structure of slices, and the basic slice-creation and management method developed in VNP.

A. Network Virtualization

When many users and systems share a limited amount of resources on computers or networks, virtualization technol-

ogy creates the illusion that each user or system possesses their own resources. Concerning networks, wide-area networks (WANs) are virtualized by using virtual private networks (VPNs). When VPNs are used, a physical network can be shared by multiple organizations, and these organizations can securely and conveniently use VPNs in the same way as virtual leased lines. Nowadays, networks in data centers are virtualized by using VLANs, while servers are virtualized by using VMs.

Many research projects on programmable virtualization networks have been carried out, and many models, including PlanetLab [Pet 02][Tur 07], Virtual Network Infrastructure (VINI) [Bav 06], Global Environment for Network Infrastructure (GENI) [Due 12], and Genesis [Kou 01], have been proposed. In VNP, Nakao et al. [Nak 12b][Nak 10] developed a VNode architecture and platform that make it possible to build programmable virtual-network environments in which slices are isolated logically, securely, and in terms of performance (QoS) from one another [Kan 12a]. In these environments, new-generation network protocols can be developed without disrupting the other slices.

B. Structure of VNode Platform

Each VNode platform domain (managed by a platform operator) is managed by a domain controller (DC), and each domain has two types of nodes: VNode and gateway (see **Figure 1**). VNode forwards packets on the platform. A domain may contain conventional routers or switches that do not have virtualization functions. VNodes are connected by tunnels using a protocol such as Generic Routing Encapsulation (GRE) [Far 00]. Therefore, a slice is neither constrained by the topology of the physical network nor by the specific functions of these nodes. A VNode can operate as a router or a switch for platform packets, so it can be deployed in conventional networks.

Each VNode consists of the following three components. *Programmer* processes packets like a router or a switch on each slice. Slice developers (or slice operators) can inject programs into programmers. *Redirector* forwards (redirect) packets from another VNode to a programmer and forward packets from a programmer to another VNode. *VNode manager* manages the VNode according to instructions from the DC.

C. Structure of Slices

In the virtual-network model developed by VNP, a virtual network (or a collection of resources in a virtual network) is called a *slice*, which consists of the following two types of components (see **Figure 2**(a)) [Nak 12b][Nak 10].

- *Virtual node* (or node sliver) represents a computational resource in a VNode. It is used for node control or processing packets with an arbitrary format. It is generated by slicing and abstracting physical computational resources.
- *Virtual link* (or link sliver) is a link resource between two virtual nodes both IP and non-IP protocols can be used on this link. It is mapped on a physical link between two VNodes. It is generated by slicing and abstracting physical network resources such as bandwidth.

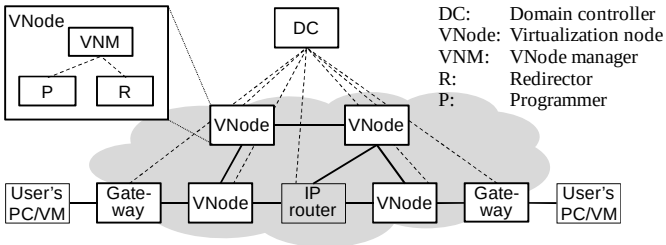


Figure 1. Physical structure of virtualization platform

A slice developer describes a *slice specification* in XML. However, each slice specification is expressed as a diagram hereafter. A specification of a slice, named S, with three virtual nodes (VN1, VN2, and VN3) and three virtual links (VL12, VL13, and VL23) is shown in Figure 2(a). The virtual nodes are mapped to VNodes (N1, N2, and N3). If the description of the mapping between virtual and physical nodes is omitted, the DC determines the mapping instead. This mapping problem, which is called the *virtual-network embedding problem*, has been widely studied (e.g., [Zhu 06] [Cho 10][Zah 10]). The virtual-node specifications in a slice specification contains URLs of the VM images, but the slice developers can load programs into the VMs and run them by using secure shell (`ssh`) commands after loading and starting the VMs.

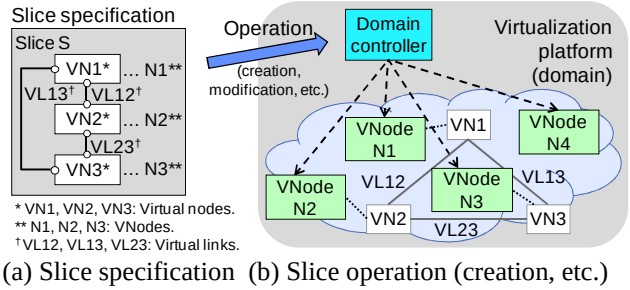
Virtual links are implemented by using a tunneling protocol such as GRE. The tunnels may bypass physical nodes. Slice structures, therefore, do not necessarily depend on the physical structure of the network. The slice developer can program virtual nodes by specifying the URL of a VM image or a fast-path program to be loaded.

D. Basic Slice-Operation-And-Management Method

The DC of a domain receives a slice-operation message with whole or part of a slice specification from the slice developer (see Figure 2). The DC distributes the message to each VNode in the domain. In a VNode, the VNode manager receives the specification and sends a part of the slice specification to the programmer and another part to the redirector: the programmer receives information required for virtual-node configuration, and the redirector receives the information required for virtual-link configuration. For example, in the case shown in Figure 2, the specification of VN1 is deployed to the programmer in N1, and the specification of VL12 is deployed to the redirectors in N1 and N2. The detailed information in virtual nodes and links are managed by VNode managers, programmers, and redirectors, but not by the DC (which is not responsible for the details).

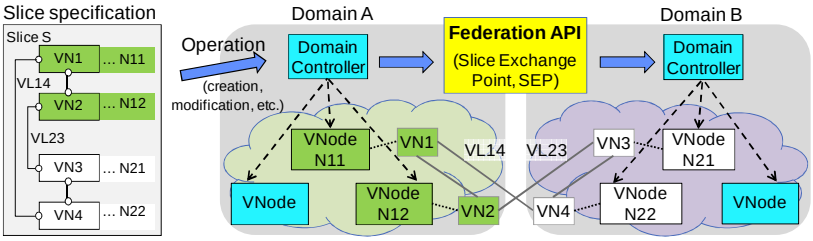
III. BASIC SLICE-FEDERATION METHOD

The federation functions provided by the slice-federation method connect two or more domains of the same or



(a) Slice specification (b) Slice operation (creation, etc.)

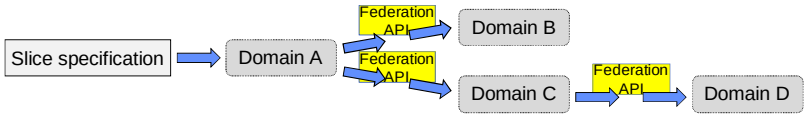
Figure 2. Single-domain slice specification and operation



(a) Basic federation method

1. Resource discovery function: Cross-domain discovery of computational resources available in virtual nodes and link resources available between virtual nodes. The API finds resources from known domains, i.e., known gatekeepers. No function for discovering DCs, DPNs, gateways, and gatekeepers is included.
2. Slice handling function: a) Creation of a slice among multiple domains, b) Slice modification, i.e., addition/removal of virtual nodes or links in a federated domain or cross-domain virtual links, c) Deletion of a slice among domains.
3. Query on statistics and manifests: a) Query on slice (and platform) statistics such as number of packets counted in a virtual link, b) Query on manifests, i.e., bottom-up parameters such as virtual-node host-names or addresses.

(b) Functions of the federation API



(c) More complicated messaging pattern

Figure 3. Federation API and cross-domain slice creation

different type of virtualization platform, including VNode platforms, GENI-based platforms [Pat 10], and so on. The domains are federated by using a set of APIs called *federation APIs* (see **Figure 3**(a) and (b)). The federation APIs should be standardized interfaces supported by various types of virtualization platforms. Each API basically consists of simple pair of a request and a reply. In the post-VNP project, the place where the federation APIs work is called the *slice exchange point* (*SEP*).

Many types of federation functions, such as listed in Figure 3(b), are provided. However, the focus in this paper is not the federation functions themselves, but it is the transformation of slice specifications, which is common to all the federation functions. Only creation function is, thus, used for explanation in this paper.

It is assumed that a slice-operation message with a slice specification is sent to the DC of a domain (domain A in Figure 3(a)) at first, and a copy of the specification is forwarded to the DC of the other domain (domain B) through the federation API. No clearing house, or hub, is used for this communication. The slice specification shown in Figure 3 consists of four virtual nodes and four virtual links. Two virtual nodes (VN1 and VN2) belong to domain A, and the other virtual nodes belong to domain B. Two of the four virtual links (VL14 and VL23) are inter-domain links. Because inter-domain links exist, these domains cannot be managed independently; in other words, the inter-domain links must be managed in relation to both domains.

In a slice creation, VN1 and VN2 are created and managed by the DC in domain A, and VN3 and VN4 are created and managed by the DC in domain B. Although the virtual links within a domain are managed solely by the DC in the domain, the inter-domain links are cooperatively created and managed by two DCs in both domains. The information required for this cooperation is exchanged using the federation API.

The federation of two domains is shown in Figure 3(a). If three or more domains are federated, the messaging pattern is more complicated. A four-domain example is shown in Figure 3(c). In this figure, domain A sends federation messages through the API (APIs) between A and B and between A and C. Domain C forwards the message to domain D.

IV. Conceptual Outline of Federation-less Federation

The federation between two domains without federation functions is conceptually outlined as follows (see **Figure 4**). A virtualization platform without federation functions does not have a concept of "other domain" because the "own domain" is the only domain. Therefore, the other domain part of the cross-domain slice specification must belong to the own domain. This means that the other domain is a sub-domain of the own domain. Because a part of the slice in the other domain is to be managed only by the DC in the other domain and duplicated management of the part must be avoided, this part must be hidden, i.e., encapsulated, from the DC in the own domain.

In a slice specification, the only way to express the set of virtual nodes and virtual links in a sub-domain is to use a virtual node. This node belongs to a pseudo VNode, which does not have network-node functions, such as routing or switching, because it only represents and delegates the other domain. The pseudo VNode is, thus, called a *domain proxy node* (*DPN*). Although a DPN is not a real network node, a DPN mimics a VNode; that is, it provides the same API as a VNode and the DC can manage it by the same method. The DC maps a *pseudo virtual node* (*PVN*) on a DPN by using the same way as for a normal virtual node, and the PVN must contain a part of the slice specification for the other domain (B) as a substructure. This means that the DPN conceptually contains an image of the other domain.

Similarly, if the other domain does not have federation functions either, it must have a DPN that contains the image of the own domain (A) and the PVN in a part of the slice specification of the other domain is mapped to a DPN that represents the own domain. Therefore, conceptually, as shown in Figure 4, the image contained in the DPN recursively contains the domain images. The slice specifications exchanged through the federation API reflect the above conceptual structure. However, this static recursion may not cause a message loop in federation.

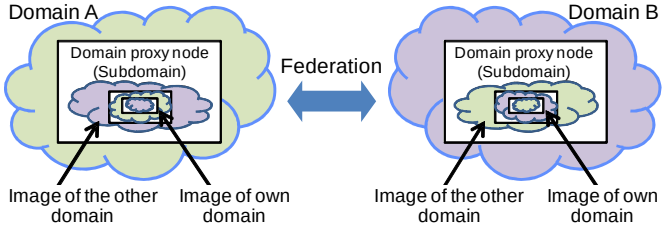


Figure 4. Conceptual outline of federation-less federation

It is assumed that the DC is not responsible for detailed information enclosed in a virtual node in the own domain; in a similar manner, it is not responsible for (does not manage) detailed information in PVNs (i.e., in the other domain).

V. Federation Architecture And Transformation of Slice Specifications

The proposed federation architecture, the transformation process of a slice specification, and a method of message loop avoidance are described in this section.

A. Architecture

The proposed federation architecture is shown in **Figure 5**. In this figure, both domains, D1 and D2, are VNode platforms. However, this architecture can be applied to heterogeneous federation; that is, even when domain D2 is a different type of platform, there is no need to modify the left half of this figure, although the detailed message contents and federation sequence may have to be changed.

The following three physical components are added to the domain as federation interfaces (i.e., SEPs).

- **Domain proxy node (DPN)** is a pseudo VNode that receives a whole slice specification (or the specification of the PVN) from the DC of the own domain. The DPN receives a slice specification containing the PVN, which is mapped to the DPN. This virtual-node contains part of the slice in the other domain. Because the slice structure is symmetric about the domain border, the slice specification described by the slice developer should also be syntactically symmetric. However, the slice specification acceptable by DPN and DC is syntactically asymmetric (that is, the virtual nodes in the other domain are enclosed but those in the own domain are not enclosed) because, in the PVN, the virtual nodes in the own domain must be bare (i.e., managed by DC), and those in the other domain must be enclosed (i.e., not managed by DC).
- **Gatekeeper** is a server for federation management and control. A gatekeeper (gatekeeper 1 in Figure 5) receives the slice specification, transforms it to an API-based format, and forwards it to the gatekeeper of the other domain (gatekeeper 2). The API between the gatekeepers is the federation API. The slice specification sent from one gatekeeper to the other must be syntactically symmetric about the domain border because the two domains are symmetric. A gatekeeper also configures the gateway for inter-domain links through the gateway-control interface (i.e., the control interface of a gateway). The word "gatekeeper" comes from the terminology of ITU-T H.323, although there are differences in its role.
- **Gateway** (federation gateway) is a network node that has a function for data conversion from the internal format in the own domain to an intermediate format between the federated domains. A gateway (gateway 1) sends packets to the other gateway (gateway 2) toward the other domain, and it receives packets for the own domain from the other gateway.

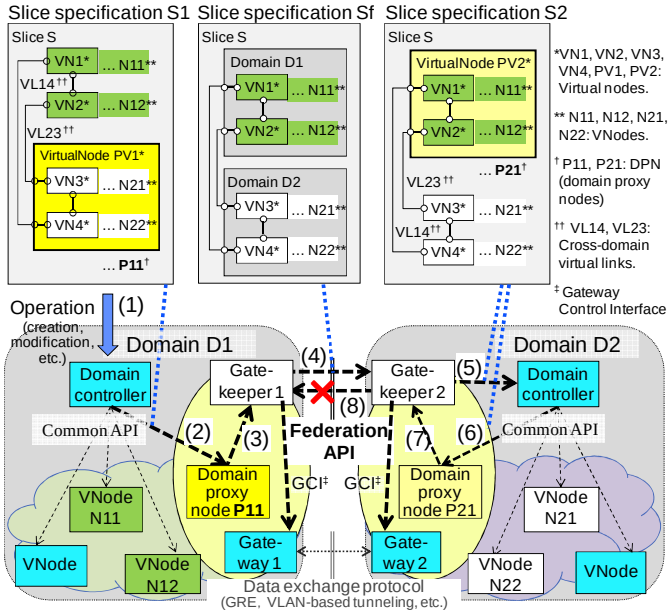


Figure 5. Federation architecture and transformation of slice specification

In the case of federating two domains, each domain has a DPN, a gatekeeper, and one or more gateways. In contrast, in the case of federating *N* domains, each domain may have *N* (or less) DPNs, one to *N* gatekeepers, and one to *N* gateways. These three types of nodes may therefore be separately deployed because of flexibility of design and implementation, and performance.

B. Transformation of Slice Specification

The forms of slice specification and the transformation process are also shown in Figure 5. In a slice specification given to the DC, the PVN encloses the part of the slice for the other domain. In slice specification S1, PV1 encloses this part. This enclosure, i.e., PV1, is the border of responsibility. Because the DC does not have federation functions, this domain-dependent form (S1) must be used for creating a cross-domain slice. However, if an appropriate preprocessor is supplied, the slice developer can use a domain-independent form, i.e., the slice Sf; in other words, the preprocessor can translate the domain-independent form to the domain-dependent form.

The original slice specification (S1) is sent to the DC (step 1 in Figure 5) at first, and the DC distributes it to all the VNodes in the own domain including the DPN, which is manually registered to the DC (step 2). Although the whole slice specification is sent to each VNode in the VNode platform, only the PVN (PV1) may be sent to the DPN (P11) in general. However, the complete specification would probably work when an error or exception occurs. If only partial slice information is available in each domain, the slice developer must collect and combine pieces of operation-and-management information to find the actual problem, such as a bug, by oneself instead of using an automated information collector.

The DPN sends the specification to the gatekeeper, which is manually registered to the DPN (step 3), and the gatekeeper transforms it to the domain-independent form Sf. This specification is syntactically symmetric about the domain border. In this specification, the virtual nodes of each domain are enclosed in an envelope labeled by the domain name. In this figure, the envelopes are labeled as domains "D1" and "D2".

The slice specification is sent to the other domain (step 4). The federation API (i.e., SEP) may have a discovery function of gatekeepers. However, gatekeepers of other domains may also be manually registered to a gatekeeper. Gatekeeper 2 transforms it to a domain-dependent form of Domain D2, and sends it to the DC (step 5). In this specification (S2), a PVN called "PV2" encloses the part of the slice (of D1). This enclosure, i.e., PV2, is the border of responsibility. PV2 is mapped to the DPN labeled "P21" in gatekeeper 2. The DC of domain D2 distributes the specification to all the VNodes in domain D2 including P21 (step 6). P21 sends the specification to the gatekeeper (step 7) in the same way that the P11 sends it to gatekeeper 1.

Gatekeeper 2 behaves differently from gatekeeper 1 in the original domain. If a slice specification is received by the current domain (D2) at first, the gatekeeper forwards it to the other domain (D1). However, if it is received by the other domain, an infinite loop may occur. In spite of this risk, this slice specification must contain information on the other domain because this information is required for interfacing the domains, i.e., for completing inter-domain links. To avoid this loop, the gatekeeper must reject sending it to the original domain (step 8) (or the gatekeeper in the other domain (D1) must reject the message). The available methods to avoid this loop are described in the next subsection.

Note that the proposed federation method may have two problems: one concerning scalability and one concerning security and privacy. The first problem is that it may take too much time or resources to parse the XML of the slice specification because PVN may contain a large and complicated structure to be parsed and analyzed. However, because neither the DC nor the DPN have to parse it, this scalability problem can be avoided. The parsing and analyzing overhead can thus be avoided, for example, by encoding the structure by a scheme such as Base64.

The second problem is that the DC or the DPN may leak domain internal information contained in the other-domain part of a slice specification. This problem can be solved by encrypting the information. For example, the content of PV1 in the slice S1, which belongs to domain D2, can be encrypted before it is given to the DC of domain D1, and this content can be decrypted by gatekeeper 2. In addition, the content of PV2 in the slice S2, which belongs to domain D1, can be encrypted by gatekeeper 1. Alternatively, the problem can be solved by removing the internal information of D1 by gatekeeper 1 before sending it to the other domain. In both cases, PV2 becomes a black box. Both contents of envelopes labeled D1 and D2 in the slice Sf between two gatekeepers are encrypted.

C. Message Loop Avoidance

As described in the previous subsection, inter-domain messaging may cause an infinite loop. Several methods to avoid this loop are described here.

A message loop is caused by the nature of federation-less federation, i.e., conceptual recursion, described in Section IV. The message pattern described in the previous subsection is the simplest one. However, if there are three or more domains, various loop patterns may occur.

The basic method for avoiding a loop is to identify a slice specification and not to forward or to process such a specification. The slice identifier contained in the message may be used for this identification. However, this problem may occur not only in a slice-creation message but also in modification or query messages. Two or more slice modification or query messages that specify the same slice may be designated by a slice developer simultaneously. The message identity must therefore be recognized by using more powerful means such as unique request identifiers.

In addition, the following methods may be used in combination with the above basic method for avoiding unexpected events caused by errors or bugs. One method is *marking*. A gatekeeper can mark messages that come from other domains. This method is probably useful only in federation of two domains because three or more types of messages, which must be distinguished when three or more domains exist, cannot be distinguished by this method. Another method uses TTL (time-to-live). The management system can add a TTL to a message when it first receives the message from outside of the system. This method is more powerful than marking. However, it still fails when the number of domains that are linearly chained is larger than the initial value of TTL. These two methods are therefore useful only as a backup of the method based on identity.

VI. MANAGEABLE INTER-DOMAIN LINKS FOR NON-IP COMMUNICATION

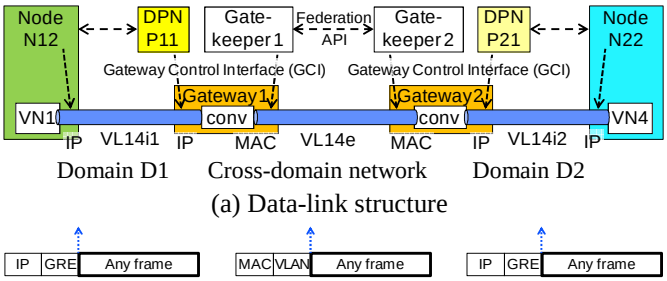
A cross-domain slice specification usually contains inter-domain virtual links. Several issues on these links and their solutions are explained. To enable manageable inter-domain non-IP communication, both issues must be solved.

1. *Protocol and data format used for inter-domain data communication (Data-plane issue)*

   The protocol and the data format used for platform (underlay) communication must be selected in consideration of available hardware and software. The function of a network between two domains may be restricted; only a limited set of protocols may be available. However, to exchange virtualized packets, a type of tunneling protocol is required. If the network is IP-based, GRE is a candidate, and if the network is Ethernet, VLAN-based "tunneling" may be used. We assume the latter is used.

2. *Method of setting up the underlay link that corresponds to the virtual link (Control-plane issue)*

   To set up an underlay inter-domain link, the facing domains must exchange necessary parameters such as IP addresses (and GRE keys) in the case of an IP-based network. These parameters are exchanged between the domains by the gatekeepers. If the network between the domains is managed by another operator, a negotiation with its manager may be required. When IP-over-VLAN is used (i.e., IP is used on the slice and VLAN is used on the platform), VLAN ID must be exchanged but MAC addresses are not necessarily exchanged by the gatekeepers because they can be obtained by using ARP. However, if "X over VLAN" is used (i.e., a non-IP protocol, which might have no "address" concept, is used on the slice and VLAN is used on the platform), MAC addresses must be exchanged in addition to VLAN ID because no address resolution or correlation mechanism such as ARP is assumed to be available.



(a) Data-link structure

(b) Data communication using arbitrary packet-format

Figure 6. Inter-domain data-link structure and data communication

The method of the link creation, which is the most important link operation, is outlined below. In slice S1 in Figure 5, there are two inter-domain virtual links, VL14 and VL23. A link that corresponds to each inter-domain virtual link is created by stitching three sections (see **Figure 6**(a)). Because two or three different protocols may be used in the two domains and the network in between, the link is divided into three sections. In the case of VL14, VL14i1 is the domain-D1-internal section, VL14e is the inter-domain section, and VL14i2 is the domain-D2-internal section. Both end points of each section of the links have their own addresses; that is, they are independent.

In the control plane, node N12 and the gateway in domain D1 (gateway 1) negotiate the parameters for VL14i1 when the domain-D1 part of the slice is created [Kan 12b]. The parameters for VL14i2 are negotiated by using the same method. As described in Figure 5 (step 4 and its response), however, the gatekeepers negotiate the parameters for VL14e, including the VLAN ID, the MAC addresses, and possibly the tunneling method. If the link resource (such as bandwidth) between two domains is managed by a network operator other than the operators of the two domains, the gatekeeper(s) must negotiate the resource with the operator. The link may even cross firewalls between the domains.

In the implementation, GRE tunneling is used for VL14i1 and VL14i2, and VLAN is used for VL14e. Available protocols and formats will be added in future. No management system in the network between the domains is assumed. Both end points of VL14i1 and VL14i2 have IP addresses for the GRE tunnels, and both end points of VL14e have MAC addresses for the inter-domain VLAN communication. VL14i1 is generated by an intra-domain signaling (using GMPLS between VNode Managers), which is a normal signaling for generating an intra-domain virtual link, between the VNode Managers of N12 and P11. The VNode managers negotiate the GRE key and the IP addresses for both end points of VL14i1. VL14i2 is generated by the same method. VL14e is generated by messaging through the federation API. The gatekeepers negotiate the VLAN ID and the MAC addresses for both end points of VL14e. The gatekeepers send commands to the gateways for setting up the link.

In the data plane, the data conversion between VL14i1 and VL14e is performed in gateway 1, and that between VL14e and VL14i2 is performed in gateway 2. If no data conversion is required, that is, the same protocol and data format are used on both sides of a gateway, the gateway can be bypassed. Figure 6(b) shows the data format when an arbitrary (non-IP or IP) packet format is used on the slice. Network processors can be used for high-performance (10-Gbps) data conversions in gateways.

VII. Implementation and Evaluation

The federation functions were partially implemented on the VNode platform and evaluated by connecting two domains as described in the following. The functions of DPN, gatekeeper, and gateway are implemented as a modified version of a VNode called “network accommodation equipment” (NACE) [Kan 12c]. NACE was originally designed for accommodating a non-virtualized network in a slice. This function is here extended to cross-domain federation.

The slice specification described in **Figure 7** and visualized in **Figure 8**(a) is used for the evaluation. (Note that in Figure 7 several parts of the specification are omitted and several identifiers are renamed for understandability.) The specification contains inter- and intra-domain virtual links, a virtual node VN0 that contains a slow-path VM, a virtual access gateway AG00 for terminal users (which is created in the gateway explained in Section II B), and a PVN named PV1 (which contains a virtual node named VN1 that contains another slow-path VM and is assigned to a DPN named P11forD2).

The specification of PV1 is sent to P11forD2, and a part of the specification is converted to the specification for domain D2 visualized in Figure 8(b). Several virtual nodes,

```
<?xml version="1.0" encoding="UTF-8"?>
<slice-design>
 <slicespec name="federation-h1">
  <sliverdef>
   <linkSlivers>
    <linkSliver name="InterDomainLink" type="link" subtype="GRE">
     <vports><vport name="e1" /><vport name="e2" /></vports>
    </linkSliver>
    <linkSliver name="IntraDomainLink" type="link" subtype="GRE">
     <vports><vport name="e1" /><vport name="e2" /></vports>
    </linkSliver>
   </linkSlivers>
   <nodeSlivers>
    <nodeSliver name="VN0" type="prog">
     <vports><vport name="vp1"/><vport name="vp2"/></vports>
     <hierarchy>
      <sliverdef>
       <nodeSlivers>
        <nodeSliver name="SP0">
         <vports><vport name="vip1" /><vport name="vip2"/></vports>
         <instance type="SlowPath_VM" subtype="KVM">
          <resources>...</resources>
          <params>
           <param key="bootImage" value="http://192.168.50.50/nict-test/sp/KVM_Ubuntu1010Server32.img" />
          </params>
         </instance>
        </nodeSliver>
       </nodeSlivers>
       <linkSlivers>...</linkSlivers>
      </sliverdef>
      <structure>...</structure>
      <params><param key="authKey" value="ssh-dss ... vnode@ubuntu1" /></params>
     </hierarchy>
    </nodeSliver>
    <nodeSliver name="PV1" type="prog">
     <vports><vport name="vp1"/></vports>
     <hierarchy>
      <sliverdef>
       <nodeSlivers>
        <nodeSliver name="VN1" type="prog">
         <vports><vport name="vip1" /></vports>
         <hierarchy>
          <sliverdef>
           <nodeSlivers>
            <nodeSliver name="SP1">
             <vports><vport name="vip1" /></vports>
             <instance type="SlowPath_VM" subtype="KVM">
              <resources>...</resources>
              <params>
               <param key="bootImage" value="http://192.168.50.58/nict-test/sp/KVM_Ubuntu1010Server32.img" />
              </params>
             </instance>
            </nodeSliver>
           </nodeSlivers>
           <linkSlivers>...</linkSlivers>
          </sliverdef>
          <structure>...</structure>
          <params><param key="authKey" value="ssh-dss ... vnode@ubuntu1" /></params>
         </hierarchy>
        </nodeSliver>
       </nodeSlivers>
       <linkSlivers>
        <linkSliver name="ln01" type="link"><vports><vport name="e1" /><vport name="e2" /></vports></linkSliver>
       </linkSlivers>
      </sliverdef>
      <structure>
       <bind name="b1"><vport sliverID="__TOP__" portname="vp1" /><vport sliverID="ln01" portname="e1" /></bind>
       <bind name="b2"><vport sliverID="VN1" portname="vip1" /><vport sliverID="ln01" portname="e2" /></bind>
      </structure>
      <params><param key="authKey" value="ssh-dss ... vnode@ubuntu1" /></params>
     </hierarchy>
    </nodeSliver>
    <nodeSliver name="AG00" type="agw"><vports><vport name="vp1"/></vports></nodeSliver>
   </nodeSlivers>
  </sliverdef>
  <structure>
   <bind name="b11"><vport sliverID="PV1" portname="vp1" /><vport sliverID="InterDomainLink" portname="e1" /></bind>
   <bind name="b12"><vport sliverID="NS00" portname="vp1" /><vport sliverID="InterDomainLink" portname="e2" /></bind>
   <bind name="b21"><vport sliverID="AG00" portname="vp1" /><vport sliverID="IntraDomainLink" portname="e1" /></bind>
   <bind name="b22"><vport sliverID="NS00" portname="vp2" /><vport sliverID="IntraDomainLink" portname="e2" /></bind>
  </structure>
 </slicespec>
 <mapping slice="federation-h1" vnetwork="Slice">
  <amap node="PV1" vnode="P11forD2" />
  <amap node="VN0" vnode="Node11" />
  <amap node="AG00" vnode="AGW0" />
 </mapping>
</slice-design>
```

InterDomain virtual link

IntraDomain virtual link

Virtual node in domain D1

A slow-path VM

Other domain (logical domain)

Virtual node in the other domain (D2)

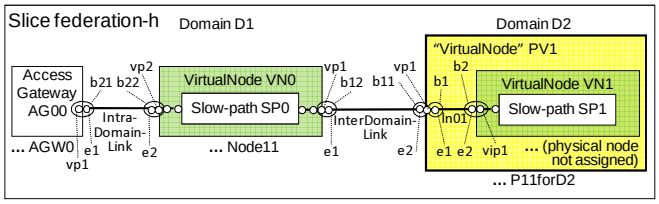
A slow-path VM

Virtual access gateway

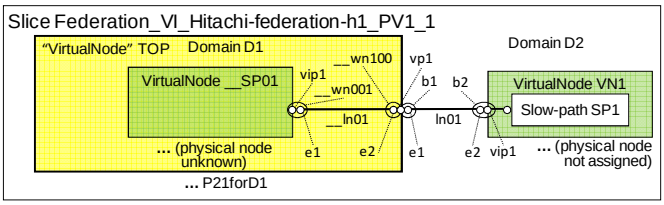
Logical-physical domain mapping

Logical-physical node mapping in own domain

Figure 7. Slice specification given to domain 1



(a) Slice specification given to domain 1



(b) Slice specification generated for domain 2

Figure 8. Visualized slice specifications

links, and port names, e.g., TOP, __SP01, and __ln01, are defined arbitrarily; that is, they are systematically generated. In contrast to the slice S2 in Figure 5, information concerning VN0 and InterDomainLink in Figures 7 and 8(a) is not provided by the gatekeepers in this implementation. This specification therefore does not contain the information.

The sequence of the slice creation, which is extracted from an execution log, is shown in **Figure 9**. The slice is given by a "slice developer" to the DC through the portal of this domain (D1) on a VNode platform in JGN-X. A part of the slice is deployed to domain D2 on a VNode platform on a local site. As shown in the figure, the processing time required for the source domain (D1) can be divided into three parts, and that required for the destination domain (D2) can be divided into four. The total time for a cross-domain slice creation is estimated as 18.2 s, which consists of waiting time and estimated downward and upward processing time. In contrast, a single-domain slice creation was measured as approximately 6 s. That is, a cross-domain slice creation takes approximately three times longer than a single-domain slice creation.

The implementation of the proposed federation method revealed the following issues.

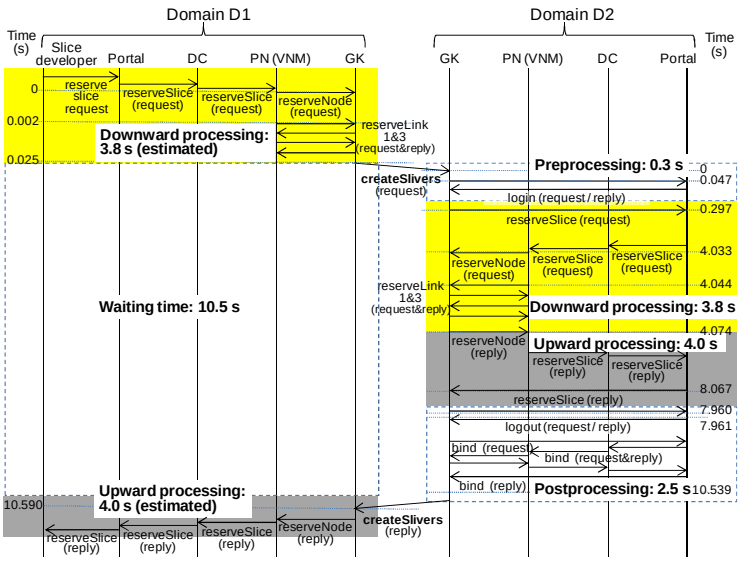


Figure 9. Measurement of slice creation

- *Restriction on modification*: Slice creation and deletion functions are performed well by the proposed method. However, there may be several restrictions on modification functions (i.e., addition and removal of virtual nodes and links). If the domain does not have a command to update a virtual node, there is no way to update the structure of the other domain, because the update on multiple nodes and links in the other domain is mapped to a single-node update in the original domain.
- *Difficulty in collecting information*: Resource discovery, statistical query, and asking manifests (such as virtual-node host-names or addresses) may be difficult to implement because the DC of a domain does not collect information of the other domain. They can be implemented only when the DC requests information from VNodes and the DPN and returns the content of the reply from the VNodes and the DPN as is to the slice developer.

However, these issues will not occur if the virtualization platform has sufficient (intra-domain) slice-modification and on-the-fly discovery and statistical query functions.

VIII. Related Work

The GENI Project defines federation architecture with a set of interfaces and data types for federation [Pat 10]. Based on this architecture, ProtoGENI [Ric 12] has federation functions for federating multiple GENI-based networks. In ProtoGENI, all the parameters required for data communication are specified in the RSpec (resource specifications) because there is no mechanism for negotiating virtual-link parameters. In contrast, in the case of the proposed federation method, the slice designer does not need to specify the parameters because the protocol to be used and the parameters such as VLAN or GRE/IP are negotiated by the gatekeepers. Especially, in ProtoGENI, ARP (and non-virtualized Ethernet and IP) is required between the domains because underlay addresses are not negotiated. Therefore, only IP/Ethernet can be used for communication between the domains. The method can be applied to other protocol combinations, including clean-slate protocols, in which no address resolution mechanism is available.

Several papers describe the relationships between federation and network virtualization [Kim 09] [Ser 11]. However, they do not focus on link-layer functions (i.e., creating and managing virtual links, etc.), namely, the focus of this study. Wahle et al. [Wah 08] proposed an architecture for cross-domain federation. This architecture contains a central federation controller called "Teagle" and a gateway in each domain. However, it does not focus on a link layer either. Zaheer et al. [Zah 10] and Chowdrury et al. [Cho 10] have studied cross-domain virtual-network embedding problems. Their focus was on virtual-to-real node mapping, and they did not mention management and control problems caused by cross-domain federation.

IX. Conclusion

A method for federating virtualization platforms was proposed. To enable federation between domains

without a federation function, first, a cross-domain slice specification that contains a PVN, which encloses other-domain part of the slice and which indicates the border of responsibility, is used. Second, the specification is handled using a DPN, which mimics a VNode that implements the PVN, and it is transformed using gatekeepers with the federation API. To enable non-IP data communication on the slice, underlay parameters (including MAC addresses) are negotiated in advance, and data packets on a slice are tunneled between gateways in these domains. Methods for avoiding infinite message loops (for specification exchange) between federated domains are also proposed.

The federation method was implemented on the VNode platform, federation between two homogeneous domains was successfully demonstrated, and the federation performance was measured. Although several issues were revealed by this implementation, they do not occur in the case of a platform with sufficient intra-domain functions.

Future work includes implementation of all the federation functions and heterogeneous federation. This method is now being applied to heterogeneous federations between a VNode platform and a platform based on ProtoGENI [Ric 12]. Future work also includes scalable federation; that is, reducing degradation in performance of creation and deletion of a slice caused by a large number of nodes in other domains or a large number of federated domains.

ACKNOWLEDGMENTS

We thank Kenichi Ogaki, Shuichi Okamoto, and Michiaki Hayashi from KDDI R&D Laboratories for their collaboration concerning federation architecture and for their useful comments on this paper. We also thank Associate Professor Aki Nakao from University of Tokyo, Yasushi Kasugai, Takeshi Ishikura, and Hidenobu Iwatake from Hitachi, Ltd., and other members of the project for their help and comments on the design, implementation, and evaluation of the federation function. Part of the research results described in this paper is an outcome of the Advanced Network Virtualization Platform Project B funded by National Institute of Information and Communications Technology (NICT) and experiments using JGN-X testbed deployed by NICT.

REFERENCES

[AKA 10] AKARI Architecture Design Project, "New Generation Network Architecture — AKARI Conceptual Design (ver2.0)", http://akari-project.nict.go.jp/eng/concept-design/-AKARI_fulltext_e_preliminary_ver2.pdf, May 2010.

[Aoy 09] Aoyama, T., "A New Generation Network: Beyond the Internet and NGN", *IEEE Communications Magazine*, Vol. 47, Vol. 5, pp. 82–87, May 2009.

[Bav 06] Bavier, A., Feamster, N., Huang, M., Peterson, L., and Rexford, J., "In VINI Veritas: Realistic and Controlled Network Experimentation", *2006 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications* (*SIGCOMM'06*), pp. 3–14, September 2006.

[Cho 10] Chowdhury, M., Samuel, F,. Boutaba, R., "PolyViNE: Policy-based Virtual Network Embedding Across Multiple Domains", *2nd ACM SIGCOMM Workshop on Virtualized Infrastructure Systems and Architecture* (*VISA*), pp. 49–56, September 2010.

[Due 12] Duerig, J., Ricci, R., Stoller, L., Strum, M., Wong, G., Carpenter, C., Fei, Z., Griffioen, J., Nasir, H., Reed, J., and Wu, X., "Getting Started with GENI: A User Tutorial", *ACM SIGCOMM Computer Communication Review*, Vol. 42, No. 1., pp. 72–77, January 2012.

[Far 00] Farinacci, D., Li, T., Hanks, S.,. Meyer, D., and Traina, P., "Generic Routing Encapsulation (GRE)", RFC 2784, IETF, March 2000.

[Fel 07] Feldmann, A., "Internet Clean-Slate Design: What and Why?", *ACM SIGCOMM Computer Communication Review*, Vol. 37, No. 3, pp. 59–74, July 2007.

[Kan 12a] Kanada, Y., Shiraishi, K., and Nakao, A., "Network-resource Isolation for Virtualization Nodes", *17th IEEE Symposium on Computers and Communications* (*ISCC 2012*), July 2012.

[Kan 12b] Kanada, Y., Shiraishi, K., and Nakao, A., "Network-Virtualization Nodes that Support Mutually Independent Development and Evolution of Components", *13th IEEE International Conference on Communication System* (*ICCS 2012*), October 2012.

[Kan 12c] Kanada, Y., Shiraishi, K., and Nakao, A., "High-performance Network Accommodation into Slices and In-slice Switching Using A Type of Virtualization Node", *2nd International Conference on Advanced Communications and Computation* (*Infocomp 2012*), IARIA, October 2012.

[Kim 09] Dae Young Kim, Mathy, L., Campanella, M., Summerhill, R., Williams, J., Shimojo, S., Kitamura, Y., and Otsuki, H, "Future Internet: Challenges in Virtualization and Federation", *5th Advanced International Conference on Telecommunications* (*AICT 2009*), pp. 1–8, May 2009.

[Kou 01] Kounavis, M., Campbell, A., Chou, S., Modoux, F., Vicente, J., and Zhuang, H., "The Genesis Kernel: A Programming System for Spawning Network Architectures", *IEEE J. on Selected Areas in Commun.*, vol. 19, no. 3, pp. 511–526, 2001.

[Nak 10] Nakao, A., "Virtual Node Project — Virtualization Technology for Building New-Generation Networks", *NICT News*, No. 393, pp. 1–6, June 2010.

[Nak 12a] Nakao, A., et al., "Advanced Network Virtualization: Definition, Benefits, Applications, and Technical Challenges, January 2012", NVSG White Paper v.1.0, https://nvlab.nakao-lab.org/nv-study-group-white-paper.v1.0.pdf

[Nak 12b] Nakao, A., "VNode: A Deeply Programmable Network Testbed Through Network Virtualization", *3rd IEICE Technical Committee on Network Virtualization*, March 2012, http://www.ieice.org/~nv/05-nv20120302-nakao.pdf

[Pan 11] Pan, J., Paul, S., and Jain, R., "A Survey of the Research on Future Internet Architectures", *IEEE Communications Magazine*, Vol. 49 , No. 7, pp. 26–36, July 2011.

[Pat 10] Peterson. L., Ricci, R., Falk, A., and Chase, J., "Slice-based Federation Architecture", http://groups.geni.net/geni/-wiki/SliceFedArch, July 2010.

[Pet 02] Peterson, L., Anderson, T., Culler, D., and Roscoe, T., "A Blueprint for Introducing Disruptive Technology into the Internet", *ACM SIGCOMM Computer Communication Review*, Vol. 33, No. 1, pp. 59–64, January 2003.

[Ric 12] Ricci, R., Duerig, J., Stoller, L., Wong, G., Chikkulapelly, S., and Seok, W., "Designing a Federated Testbed as a Distributed System", *TridentCom 2012*, June 2012.

[Ser 11] Serrano, M., Davy, S.,Johnsson, M., Donnelly, W., and Galis, A., "Review and Designs of Federated Management in Future Internet Architectures", in "Future Internet Assembly 2011: Achievements and Technological Promises", *Lecture Notes in Computer Science*, Vol. 6656, Springer, 2011.

[Tur 07] Turner, J., Crowley, P., Dehart, J., Freestone, A., Heller, B., Kuhms, F., Kumar, S., Lockwood, J., Lu, J.,Wilson, M., Wiseman, C., and Zar, D., "Supercharging PlanetLab — High Performance, Multi-Application, Overlay Network Platform", *ACM SIGCOMM Computer Communication Review*, Vol. 37, No. 4, pp. 85–96, October 2007.

[Wah 08] Wahle, S., Gavras, A., Gouveia, F., Hrasnica, H., and Magedanz, T., "Network Domain Federation – Infrastructure for Federated Testbeds", *NEM Summit 2008*, October 2008, http://www.nem-summit.eu/

[Zah 10] Zaheer, F. E., Jin Xiao, and Boutaba, R., "Multi-provider Service Negotiation and Contracting in Network Virtualization", *2010 IEEE Network Operations and Management Symposium* (*NOMS*), pp. 471–478, April 2010.

[Zhu 06] Zhu, Y. and Ammar, M., "Algorithms for Assigning Substrate Network Resources to Virtual Network Components", *25th IEEE Int'l Conference on Computer Communications* (*INFOCOM 2006*). pp. 1–12, April 2006.