Federation-less Federation of ProtoGENI and VNode Platforms

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

***Abstract* – Our previous work enabled "federation-less federation", which means a federation of multiple network-virtualization platforms that do not support federation functions, and applied this method to a homogeneous federation of platforms called the "VNode" infrastructures. In this study, this method was applied to a heterogeneous federation of the ProtoGENI and the "VNode". We intended to federate these platforms through a single management interface. However, the federation architecture of GENI, which is called the slice-based federation architecture (SFA), cannot be used for single-interface federation but we could not modify the ProtoGENI platform to enable it. Therefore, a method for applying federation-less-federation to ProtoGENI was developed. It enabled federation of these platforms by adding several nodes but without modifying preexisting platforms. This method was applied to federation of the ProtoGENI platform at the University of Utah and two VNode infrastructures in Japan, the slice creation and deletion time was measured and evaluated to be acceptable. Although this federation-less-federation implementation still has several minor problems, it was proved to be useful for experiments and demonstrations.**

***Index terms* – Network virtualization, VNode, Federation, GENI, ProtoGENI, Slice-based federation architecture (SFA)**

I. Introduction

In a recent project on new-generation networks (NwGNs), called the VNode Project [Nak 10], a deeply programmable network-virtualization architecture and platform, called VNode [Nak 12][Kan 12], was developed. Two virtualization platforms based on VNode have been deployed. One is in a testbed network, called JGN-X (JGN Extreme) [Pan 11], for designing, deploying, and testing new network services in Japan, and the other is deployed in the Hakusan Laboratory of the National Institute of Information and Communications Technology (NICT) in Tokyo.

In a succeeding project, a method for federating two or more virtualization platforms was developed. Each platform probably has limited types and limited amounts of resources, so their federation enables a combination of various resources for network applications. The previously developed federation method enabled both federation of homogeneous domains [Kan 13], which was applied to the two above-mentioned VNode Infrastructures, and federation of heterogeneous domains [Oka 13] such as a VNode Infrastructure and other virtualization platforms.

The federation method has two features. First, it supports a *federation-less federation* [Kan 13]; that is, it enables federation of multiple domains that do *not* support federation functions without having to modify their existing network-virtualization platform. Several components added to the original platform enable the federation. Second, it supports a *familiar single-interface federation* or *multi-way federation* [Tar 15], which provides a single management interface to slice developers. A federated slice is created by a negotiation between the managers of the domains. The homogeneous federation method for VNode Infrastructures [Kan 13] also supports a single-interface federation.

Other federation methods, however, do not necessarily support a single-interface method. In the case of GENI (Global Environment for Network Innovations) [Due 12], domains, which are called aggregates in GENI, are federated by using the slice-based federation architecture (SFA) [Pet 10][Ric 13] [Ric 12]. Utilizing the SFA, a slice developer can federate aggregates by sending requests to all the managers of the aggregates (AMs). The communication sequence required for the federation is therefore strongly dependent on the domains to be federated and the number of domains. The sequence for creating a federated slice is quite different from that for creating a single domain.

Because we intended to federate platforms by a familiar single-interface method, instead of using the SFA, the method of federation-less federation was applied to not only VNode Infrastructures but also a GENI domain. A federation function between VNode and GENI was implemented in the VNode infrastructures on JGN-X and Hakusan and the ProtoGENI [Ric 13][Ric 12] platform at the University of Utah, which is an implementation of GENI.

The rest of this paper is organized as follows. Section II describes related work. Section III summarizes the previously developed federation-less federation method. Section IV outlines federation-less federation between the ProtoGENI and the VNode platforms. Section V describes the implementation and evaluation of this method, and Section VI concludes this paper.

II. Related Work

ProtoGENI has a federation architecture called the slice-based federation architecture (SFA). Although each virtualization platform in federated platforms has its own method of allocating and managing networking and computational resources, the SFA enables a unified method for allocating resources on these platforms to slices and managing them. Each platform can support an SFA wrapper to federate it with other platforms [Aug 14][Wah 10][Ban 11].

However, two issues, namely, *interface enforcement* and *federation-resource management*, remain to be addressed. The first issue is that to create a slice that is spread among multiple domains by using the SFA, the client (e.g., a GUI) or the user must understand the architecture, which may be very different from the virtual-network architecture that the user or the designer of the PC client is familiar with. If the client designer or the user is familiar with a certain virtualization platform, it will be easier and better to use the method for this platform to

create and manage a slice than to use the method used for the SFA.

The second issue, namely, federation-resource management, is that there is no established method for handling resources between federated platforms. The GENI architecture has a brokering service called network stitching [GENb], which replies to questions on available VLAN resources among the platforms. However, because the stitching service actually does not manage the resources, a slice may fail to allocate the required resources even when it replies with a positive answer. In addition, no unified and consistent method for obtaining information on other types of resources is available in GENI. The SEP manages all the resources among the platforms, including both the networking and computational resources on the platforms and networking resources between them. Moreover, the familiar single-interface federation provides a unified method for allocating and de-allocating the resources.

III. FEDERATION-LESS FEDERATION

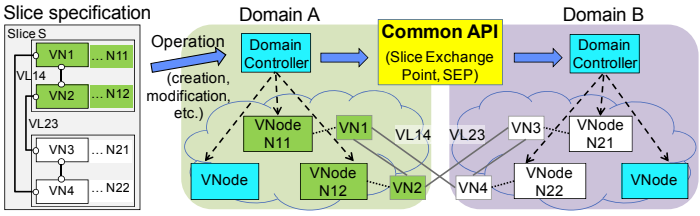
This section briefly overviews the federation-less federation method proposed in a previous paper [Kan 13] and explains the reasoning behind this method.

A. Basic federation method

The federation functions provided by the slice-federation method connect two or more domains of the same or different types of virtualization platform, including VNode Infrastructures and GENI-based platforms such as ProtoGENI. The domains are federated by using a set of XML-RPC-based APIs [XML] called *Common APIs* [VNP 14] (see **Figure 1**(a)). The set of federation APIs should be a standardized interface supported by various types of virtualization platforms. Each API basically consists of simple pair of a request and a reply. The federation through the common APIs is managed by the *slice exchange point* (*SEP*) [Oka 13][Tar 15], which will be explained in detail later. Many types of federation functions, such as those listed in Figure 1(b), are provided.

A feature of this federation method is the *familiar single-interface federation*. It provides a single management interface (and a single control framework) to slice developers who are familiar with (i.e., who usually uses) this interface. That is, a slice developer (or a client such as a GUI) sends a slice creation and other requests to one management interface, i.e., only one domain. A federated slice is created not by sending the definition to multiple domains by the slice developer but by a negotiation between the managers of the domains. Accordingly, a slice operation can be initiated from any domain in the federated domains. Therefore, this feature is also called *multi-way federation* [Tar 15].

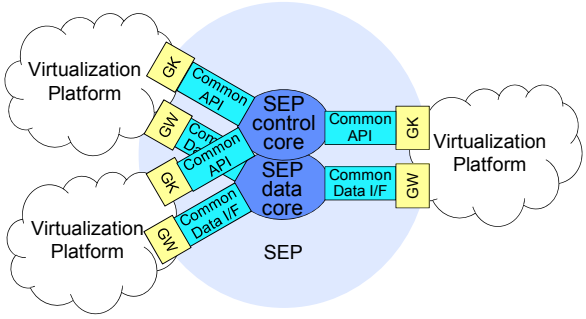
It is assumed that a slice-operation message with a slice specification is first sent to the management server of a domain (domain A in Figure 1(a)), which is called a domain controller (DC). A copy of the specification is then forwarded to the DC of the other domain (domain B) through the federation API. The slice-specification example shown in Figure 1 consists of four virtual nodes and four virtual links. Two virtual nodes (VN1 and VN2) belong to domain A, and the others belong to domain B. For simplicity, in the slice specification, their mapping is assumed to be fixed. Two of the four virtual links (VL14 and VL23) are inter-domain links. Because inter-domain links exist, the domains cannot be managed independently; in other words, the inter-domain links (or



(a) Basic federation function

1. Resource discovery: 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: a) creation and resource allocation of a slice among multiple domains, b) slice modification, i.e., addition/removal of virtual nodes or links (i.e., resource modification) in a federated domain or cross-domain virtual links, and c) resource de-allocation and 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 and b) query on manifests, i.e., bottom-up parameters such as virtual-node host names or addresses.

(b) Functions of common APIs



(c) Federation of three or more domains

Figure 1. Federation method and common APIs

resources required for implementing these links) must be managed in relation to both domains (and possibly a third domain between these 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 the DCs in both domains. The information required for this cooperation is exchanged using the federation API.

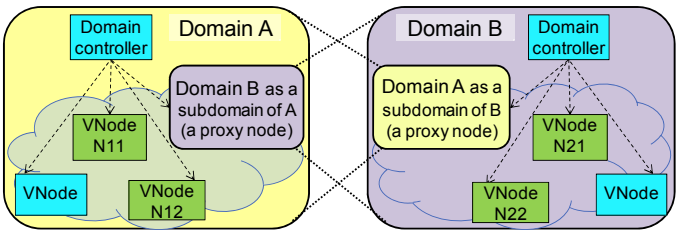
Figure 1(a) shows a federation between two domains only. If three or more domains are federated, the SEP plays the major role in communicating between these domains. A three-domain example is shown in Figure 1(c). The federation is managed by the SEP, consisting of a conceptually centralized federation manager (broker) called a SEP core, which consists of a control-plane component called a SEP control core and a data-plane component called a SEP data core (which may be empty). The SEP contains unified federation APIs (called common APIs) and an interworking function (IWF) between a domain and the SEP core. The IWF consists of control-plane components called gatekeepers (GKs) and data-plane components called federation gateways (GWs). The GKs and GWs are components of the SEP; that is, they are external to the

virtualization platform. However, the GKs convert platform-dependent proprietary messages of the virtualization platform to unified messages of the common APIs.

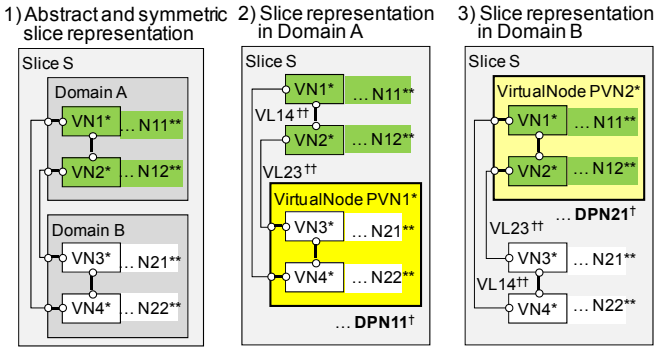
B. Outline of federation-less federation

Even if the management system of a domain (which is called the "own domain") does not have federation functions, a federation can be achieved by regarding other domains as sub-domains of the own domain. That is, each other domain to be federated may be regarded as a proxy node of the own domain [Kan 13]. **Figure 2**(a) shows the domains to be federated and the physical nodes of these domains. The nodes in the other domain are virtually enclosed in (encapsulated by) the *domain proxy node* (*DPN*), so they are not managed by the own domain. (Instead, they are managed by the management system of the other domain.) A DPN communicates with the DC in a similar way to a VNode (that is, it has the same APIs as a VNode), but it does not have network-node functions, such as routing or switching, because it only represents the other domain and delegates requests. The SEP handles messages between these two proxy nodes through GKs.

Figure 2(b) shows examples of three representations of a federated slice, which must be mapped to the physical networks shown in Figure 2(a) by the domain controllers. Figure 2(b1) shows an abstract and symmetric representation. This representation is suitable for allowing the slice developer to specify the slice structure. However, because it explicitly contains the concept of a multiple-domain network, it is not acceptable to a DC without a federation function. Representations close to the physical structure are therefore used. The representation (RSpecs) for domain A is shown in Figure 2(b2), and that for domain B is shown in Figure 2(b3). The virtual nodes of the other domains are enclosed by a *pseudo virtual node* (*PVN*), which is mapped to the DPN and



(a) Physical structure



*VN1, VN2, VN3, VN4, PVN1, PVN2: Virtual nodes.
** N11, N12, N21, N22: VNodes (physical nodes).
† DPN11, DPN21: Domain proxy nodes (physical nodes).
†† VL14, VL23: Cross-domain virtual links.

(b) Virtual structure representations

Figure 2. Structures of federated domains and nodes

represents the other-domain part of the slice. The DC of the own domain does not manage the virtual nodes of the other domain because they are enclosed in (encapsulated by) the pseudo node.

IV. FEDERATION-LESS FEDERATION OF PROTOGENI AND VNODE PLATFORMS

This section describes the requirements and the method of federating ProtoGENI and other domains including VNode Infrastructures by using the federation-less federation method.

A. Requirements

Federations on ProtoGENI are based on the SFA. To federate domains by using the SFA, each domain must have entities and abstractions required by the SFA; that is, it must have components and slices as abstractions, and each group of components, which is called an aggregate, must be managed by an aggregate manager (AM). A slice developer or a client (e.g., a GUI) must send requests, such as slice creation requests, to the AMs of all the federated domains. That is, the user or client must know all the domains to be federated.

However, to apply the single-interface federation of ProtoGENI and VNode platforms, the domains must be federated by only one request to an AM by the slice developer. The SEP handles the request and determines the domains and propagates the request to all the domains through GKs of the domains. To federate ProtoGENI and VNode domains by this method, therefore, a request sent to the VNode domain controller (DC) must be propagated to the AM of the ProtoGENI domain, and a request sent to the AM must be propagated to the DC. The original ProtoGENI architecture including the SFA does not have this function.

To achieve a familiar single-interface federation, a federation method different from that of ProtoGENI must be developed. Because a method for achieving this federation requires an extension of ProtoGENI function but we cannot modify the ProtoGENI platform at the University of Utah, a new method that enables the federation without modifying the ProtoGENI platform must be developed.

B. Outline of method

To implement a familiar single-interface federation of both platforms, the federation-less federation method was applied not only to VNode but also to ProtoGENI. As described in the previous section, when sending a request to the home domain, the nodes in the other domain are enclosed in a pseudo node in the slice specification in the request, which contains the resource specifications (RSpecs) in GENI. If the request is submitted to the DC in a VNode domain, the request is processed by using the method described in the previous section. That is, the DPN receives the specification of a PVN, which contains the specifications of the other domain (see Figure 2(b)). However, if a slice-creation request is submitted to the AM in the ProtoGENI domain, the slice specification and the handling method are different.

When the AM receives a slice specification, which is in the domain-specific form shown in Figure 2(b2), it is processed as follows (see **Figure 3**). When the DPN, which is actually implemented as a PVN as explained later, is invoked (that is, receives a slice creation request), it invokes the GK. The GK sends the specifications of the other domains, which are actually retrieved from the AM, to the SEP. The ProtoGENI-

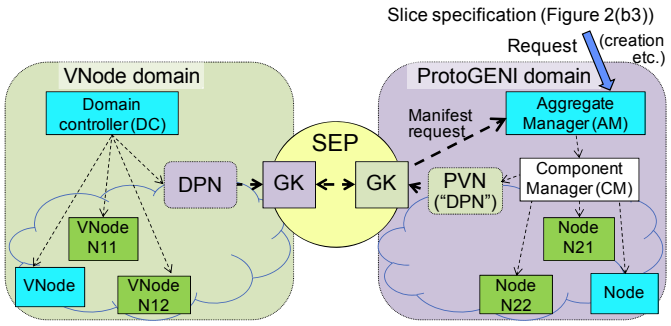


Figure 3. ProtoGENI-to-VNode federation method

side GK, which is connected to the ProtoGENI platform, and the program used in the PVN were implemented.

To implement the above-described federation method, the following four problems must be solved. First, no DPN was available and could not be implemented in the ProtoGENI aggregate at the University of Utah. DPN thus had to be replaced by other components. Second, the DPN replacements do not receive the RSpecs of the other domain, but they need the RSpecs to create the slice. Third, the DPN replacements do not receive the parameters for inter-domain connections, especially VLAN IDs in the ProtoGENI aggregate. They are necessary to establish the connections between the domains. Fourth, the DPN replacements do not receive slice-creation and slice-deletion messages; instead, one is invoked when the created slice is "started", and it is just killed when the slice is "stopped" before it is deleted. This sequence is a problem because the DPN replacements are unable to time slice creation or deletion. These problems and solutions are explained more in the following subsections.

C. Implementation of DPN functions

The first problem to be solved is that no physical DPN is available in a ProtoGENI domain and a DPN cannot be implemented without modifying the ProtoGENI platform. In the original proposal of federation-less federation [Kan 13], a DPN is a physical component of a domain. It receives the slice definition, extracts and converts the information on the other domain, and sends it to the other domain through the common APIs. However, because we are just a user of the ProtoGENI platform (i.e., not a developer or manager), we cannot modify it by adding or modifying a platform component of ProtoGENI.

This problem can be solved by giving the role of a DPN to a PVN and the GK. The PVN is a slice component and is allocated and handled by the same method as other nodes in a slice of ProtoGENI. To create a slice that spreads between ProtoGENI and VNode domains, the PVN is specified in the slice specification and allocated to a single computer. However, the resources that the PVN in the slice specification contains, namely, the resources of the other domains, are not allocated by the AM. When the PVN is invoked, it runs an initiation script that triggers the GK. The GK collects the RSpecs and parameters and sends them to the other domain. The method for collecting them is described in the next subsection. If there are multiple slices between ProtoGENI and VNode domains, each slice contains a PVN, but only one GK exists. A PVN sends a request with the slice identifier to the GK.

D. Obtaining RSpecs of other domains

The second problem is that the DPN replacements, at least the GK, need the RSpecs of the other domain, but the RSpecs are not pushed to them. In the original design described in the previous paper [Kan 13], a DPN receives the whole slice specification that contains a specification of a PVN, which contains the RSpecs of the other domain. However, a PVN in the ProtoGENI domain does not receive the RSpecs because an AM just allocates resources to an allocated node; namely, it does not send the RSpecs to the node.

This problem can be solved by the following method. When the GK is triggered by the PVN, it obtains the "manifest" of the slice by sending a "list resources" request to the AM. A manifest is an XML-based description that contains both the RSpecs and parameters assigned as the results of resource allocation, including VLAN IDs, MAC addresses, and IP addresses; that is, it contains both top-down and bottom-up information concerning the slice. The GK extracts the RSpecs of the other domains and sends them to the domains through the SEP. Because the manifest contains a copy of the specification of the PVN that contains the original RSpecs of the virtual nodes and links, which are described by the slice developer, it contains the RSpecs of the other domains.

E. Obtaining parameters of allocated resources

The third problem is that the DPN replacements do not receive the parameters for specifying inter-domain connections. In the original design of our federation method, domain-internal parameters for connecting inter-domain links are obtained by negotiation. However, no such mechanism is available on ProtoGENI. As for the proposed federation method, each inter-domain link is divided into three parts, i.e., an inter-domain part and two intra-domain parts (**Figure 4**), as described in the previous paper [Kan 13]. A DPN in each domain receives the parameters for the intra-domain part (i.e., GRE keys and IP addresses of GRE tunnels) by negotiating with the end-point VNode. However, in a ProtoGENI domain, no negotiation is used for this purpose, but it is necessary to get the networking parameters, i.e., VLAN ID, for the intra-domain (and for the inter-domain) connection.

This problem can be solved by extracting the required parameters, such as the VLAN IDs, from the manifest. The PVN sends these parameters to the other domain through the SEP.

F. Determining resource allocation/de-allocation timing

The fourth problem is that nodes in a created slice do not receive the creation and deletion messages. They are merely invoked when the created slice is "started", and it is merely killed when the slice is "stopped" before it is deleted. The DPN replacements are thus unable to catch the timing of slice creation or deletion.

This problem can be solved by the following method. The solution for slice creation is to create the VNode-domain part

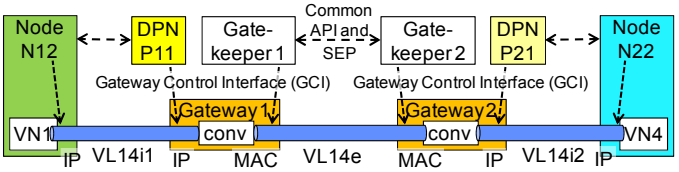
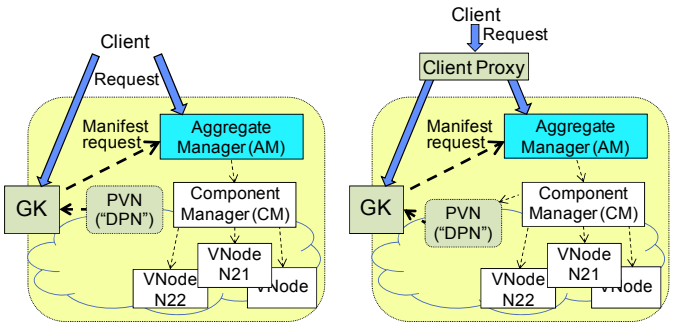


Figure 4. Inter-domain link structure



(a) Message copy by client (b) Message copy by proxy

Figure 5. Alternative solutions to the fourth problem

of the slice when the ProtoGENI-domain part is started. This solution is acceptable when the size of the slice is not large. However, if a more scalable solution is required, an alternative solution (shown in **Figure 5)** can be applied. The solution is that the client sends a message to the GK in addition to the AM (Figure 5(a)) or it sends a message through a client proxy that passes the message to both the AM and the GK (Figure 5(b)).

A solution for slice deletion is to send a deletion request, which is called "deleteslivers" in GENI, to the client proxy or the PVN instead of sending it to the AM. The client proxy or the PVN forwards the message to the AM and the GK. If the PVN is used for this purpose, this messaging must precede

```
<rspec ...>
  <node client_id="GKname" component_id="..." ...>
    <sliver_type name="emulab-bbg" />
    <interface client_id="GKname:if0" />          Gatekeeper I/F (ProtoGENI domain)
  </node>
  <node client_id="NS01" component_manager_id="..." ...>
    ...
  </node>                                         Virtual node in ProtoGENI domain
  ... <!-- Other virtual nodes (node slivers) in ProtoGENI domain -->
  <node client_id="PVN" ...>                      PVN = Other domain
    <sliver_type name="raw-pc" />
    <federation_design>
      <node client_id="GKname" ...>               Gatekeeper I/F (VNode domain)
        <sliver_type name="emulab-bbg" />
        <interface client_id="GKname:if0" />
      </node>
      <node client_id="NodeVNode1" component_id="VN13"
            component_manager_id="VNodeDomainName" ... >
        <sliver_type name="SlowPath_VM" sub_type="KVM">
          <common_API_param key="cpu" value="4" />
          <common_API_param key="arch" value="x86_64" />
          <common_API_param key="memory" value="4096" />
          <common_API_param key="storage" value="30GB" />
          ...
        </sliver_type>
        <interface client_id="NodeV1:vip1" />
        <interface client_id="NodeV1:vip2" />
      </node>                                     Virtual node in VNode domain
      ... <!-- Other virtual nodes (node slivers) in VNode domain -->
      <link client_id="LinkVNode1">
        <component_manager name="remotedomain" />
        <component_manager name="VNodeDomainName" />
        <interface_ref client_id="GKname:if0" />
        <interface_ref client_id="NodeV1:vip1" />
        <property source_id="GKname:if0"
                  dest_id="SP021:vip1" />
        <property source_id="SP021:vip1"
                  dest_id="GKname:if0" />
      </link>                                     Virtual link in VNode domain
      ... <!-- Other virtual links (link slivers) in the VNode domain -->
    </federation_design>
  </node>
  <link client_id="LS01">
    <interface_ref client_id="NS01:vip1" />
    <interface_ref client_id="GKname:if0" />
    <property source_id="NS01:vip1" dest_id="GKname:if0"
              capacity="100000" />
    ...
  </link>                                         InterDomain virtual link
  ... <!-- Other interdomain virtual links -->
</rspec>
```

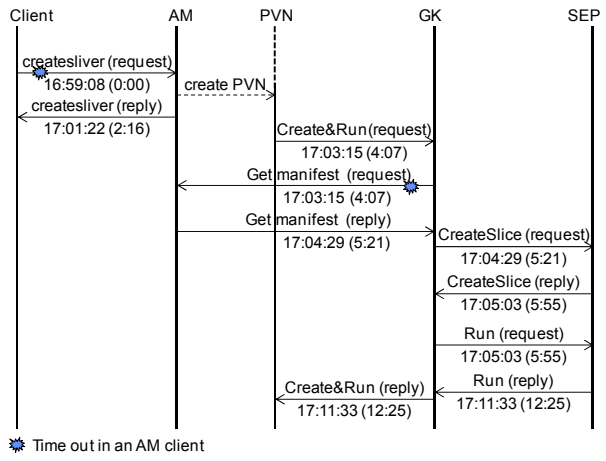
Figure 6. Outline of RSpecs (a slice definition) for a ProtoGENI-VNode federation

destruction of the PVN. If the client sends the message to the GK or the client proxy, this constraint is not required, and this method is symmetric with the methods described in Figure 5. However, the client must know the address and interface of the GK.

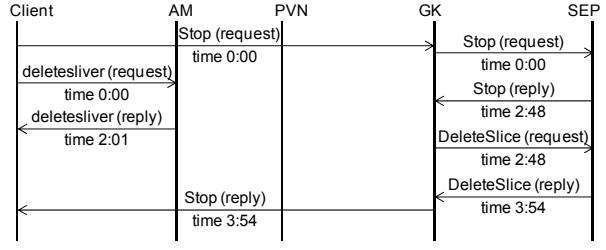
V. Implementation and Evaluation

The federation functions were partially implemented for demonstration and evaluation to show the method is feasible and its performance is acceptable. The slice specification used is distributed between a ProtoGENI domain in Utah and two VNode domains in Japan, which are connected through trans-pacific VLANs. It is expressed as a collection of RSpecs of GENI, which is partially listed in **Figure 6**. This figure contains the definition of the PVN, which contains specifications for two VNode domains as well as specifications for the ProtoGENI domain and the inter-domain virtual links. The interfaces to the GKs are also specified explicitly in this specification. Note that one of the VNode domains is omitted here. Sequences for a slice creation and a slice deletion are described in **Figure 7**. This figure also contains the elapsed time since the client requests a creation or deletion. The creation took 12 min 41 sec, and the deletion took 3 min 54 sec on average. (Each time value is an average of two measured values.)

Figure 7(a) shows a slice-creation sequence. First, the client sends a "createsliver" request to the AM. This request creates a PVN as a part of the slice. When the PVN starts, it triggers the GK. The GK obtains the manifest from the AM, extracts the slice design of the VNode part, and sends it to the SEP. The SEP negotiates with the VNode domains and returns a reply



(a) Slice creation



(b) Slice deletion

Figure 7. Implemented communication sequences and measurement results

after the VNode parts are created.

The elapsed time is acceptable but much longer than the optimum for two reasons. One reason is that when the client and the GK send requests to the AM, timeouts caused by an erroneous usage of an intermediate program (called Omni) consume approximately two minutes, but they do not affect the result. The other reason is that the creation of three parts of the slice, i.e., the ProtoGENI part and the two VNode parts, are mostly sequential because the latter is invoked after the PVN is created. If the sequence is optimized so that the client sends a request for other domains to the GK instead of the AM, the completion time can be earlier. However, this sequence complicates both the request and the sequence.

Figure 7(b) shows a slice-deletion sequence. The elapsed time is reasonable; however, because of limitations of implementation resources, this sequence is a temporary version. That is, in an ideal design, the client sends only one request for a slice deletion. In the above sequence, however, it sends two requests. One request is sent to the AM, and the other is sent to the GK.

The implemented federation method was successfully used for demonstrating federations of ProtoGENI and VNode domains at the 20th GENI Engineering Conference (GEC 20) [GENa]. Although the creation time and the deletion sequence are not ideal, this implementation was proved to be useful for experiments and demonstrations.

VI. Conclusion

To implement a familiar single-interface federation, the previously proposed federation-less federation method was applied to not only VNode but also ProtoGENI. The federation method used for ProtoGENI was applied to federation of the ProtoGENI platform at the University of Utah and VNode Infrastructures in Japan, and it enabled federation of these platforms by adding several nodes but without modifying preexisting platforms. The federation sequences were logged and the elapsed time was measured. Although this federation-less-federation implementation still has several minor problems, the measurement result shows that the time required for slice creation and deletion is acceptable for demonstration purposes. The proposed method was successfully used for demonstrating federation of the ProtoGENI and two VNode domains and proved to be useful for experiments and demonstrations.

Future work will include improving the implementation of the proposed method for the ProtoGENI platform, including improvement of the slice-deletion sequence, and application of this method to federation of three or more different types of domains (including a VNode domain).

Acknowledgments

The authors thank Michiaki Hayashi and Shuichi Okamoto of KDDI R&D Laboratories for their collaboration concerning federation architecture and for their useful comments on this paper. The authors also thank Nozomu Nishinaga from the National Institute of Information and Communications Technology (NICT) and Rob Ricci and Gary Wong from the University of Utah for setting up ProtoGENI for federating to the VNode Infrastructure, and the authors thank Professor Akihiro Nakao from the University of Tokyo, Yasushi Kasugai from Hitachi, Ltd., Kei Shiraishi, Hidenori Takagi, Takanori Ariyoshi, and Hidenobu Iwatake from Hitachi Systems, 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 NICT.

References

[Aug 14] Augé, J., Parmentelat, T., Turro, N., Avakian, S., Baron, Larabi, M. A., Rahman, M. Y., Friedman, T., and Fdida, S., "Tools to Foster a Global Federation of Testbeds", *Computer Networks*, Special Issue on Future Internet Testbeds, 2014.

[Ban 11] Bannazadeh, H., Leon-Garcia, A., Redmond, K., Tam, G., Khan, A., Ma, M., Dani, S., and Chow, P., "Virtualized Application Networking Infrastructure", *TridentCom 2010*, LNICST 46, pp. 363–382, 2011.

[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.

[GENa] "GEC 20", http://groups.geni.net/geni/wiki/GEC20Agenda

[GENb] "GENI Network Stitching-Overview", https://wiki.-maxgigapop.net/twiki/pub/GENI/NetworkStitching/geni-network-stitching-architecture-overview.pdf

[Kan 12] 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 13] Kanada, Y., Tarui, T., and Shiraishi, K., "Federation-less-federation of Network-virtualization Platforms", *IFIP/IEEE International Symposium on Integrated Network Management* (*IM 2013*), May 2013.

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

[Nak 12] 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

[Oka 13] Okamoto, S., Kuroki, K., Matsumoto, N., and Hayashi, M., "Design of Network Slice Exchange for Bridging Future Internet Testbeds", *18th OptoElectronics and Communications Conference* (*OECC/PS 2013*), pp. 1–2, June-July 2013

[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.

[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.

[Pet 10] Peterson, L., Ricci, R., Falk, A., and Chase, J., "Slice-based Federation Architecture, Version 2", http://groups.geni.net/-geni/wiki/SliceFedArch, July 2010.

[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.

[Ric 13] Ricci, J., Wong, G., Stoller, L., and R., Duerig, "An Architecture for International Federation of Network Testbeds", *IEICE Trans. Commun.*, Vol. E96-B, No. 1, pp. 2–9, 2013.

[Tar 15] Tarui, T., Kanada, Y., Hayashi, M., and Nakao, A., "Federating Heterogeneous Network Virtualization Platforms by the Slice Exchange Point", submitted for IM 2015.

[VNP 14] VNode Project, "Federation Architecture and Common API / Common Slice Definition (Draft V2.0)", https://nvlab.nakao-lab.org/Common_API_V2.0.pdf

[Wah 10] Wahle, S., Magedanz, T., and Gavras, A., "Conceptual Design and Use Cases for a FIRE Resource Federation Framework", in Tselentis, G., et al. ed., "Towards the Future Internet", IOS Press, 2010.

[XML] XML-RPC Home Page, http://www.xmlrpc.com/.