A Node Plug-in Architecture for Evolving Network Virtualization Nodes

Yasusi Kanada
Central Research Laboratory, Hitachi, Ltd.
Totsuka-ku Yoshida-cho 292, Yokohama 244-0817, Japan
Yasusi.Kanada.yq@hitachi.com

Abstract – **Virtualization nodes, i.e., physical nodes with network virtualization functions, contain computational and networking components. Virtualization nodes called "VNodes" enabled mutually independent evolution of computational component called programmer and networking component called redirector. However, no methodology for this evolution has been available. Accordingly, a method for evolving programmer and redirector and developing new types of virtualized networking and/or computational functions in two steps is proposed. The first step is to develop a new function without updating the original VNode, which continues services to existing slices, using a proposed plug-in architecture. This architecture defines predefined interfaces called open VNode plug-in interfaces (OVPIs), which connect a data and a control plug-ins to a VNode. The second step is to merge the completed plug-ins into the original programmer or redirector. A prototype implementation of the above plug-in architecture was developed, tested, and evaluated. The prototype extends the redirector by adding new types of virtual links and new types of network accommodation. Estimated throughputs of a VLAN-based network accommodation and a VLAN-based virtual link using network processors are close to a wire rate of 10 Gbps.**

Keywords – **Network-node plug-in architecture, Data plug-in, Control plug-in, Network virtualization, Virtualization node, VNode, Virtual link, Network processors.**

I. Introduction

In Japan, several projects targeting new-generation networks (NwGNs) 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 IPs but work well on NwGNs. In the Virtualization Node Project (VNP), a virtualization-platform architecture consisting of virtualization nodes (VNodes), namely, a "VNode architecture," was developed by Nakao et al. [Nak 10]. They have also developed a high-performance, fully functional, virtualization testbed in JGN-X, which is a testbed widely used by network researchers. The goal of this VNP is to develop an environment in which multiple slices (virtual networks) with independently and arbitrarily designed and programmed NwGN functions run concurrently, but are logically isolated, on a physical network.

The VNode architecture enabled mutually independent development and evolution of *programmers*, i.e., programmable computational node-components in VNodes, and *redirectors*, i.e., networking node-components in VNodes [Nak 12] [Kan 12a]. In future, a VNode may contain various types of programmers and redirectors. If they are modular, and the interface between them is clearly defined and works efficiently, each vendor can develop software and/or hardware components independently from other components. No method for this VNode evolution, however, has been available.

In the present study, such a method for evolving VNodes, especially for developing advanced redirectors and new types of virtual links, is proposed. By means of this method, a VNode is evolved in two steps. The first step is to develop a new redirector or programmer component as software and/or hardware plug-ins and to install and to connect them to the redirector or the programmer of an existing VNode, without updating the original VNode, through predefined open interfaces. The VNode can continue services to existing slices because they are isolated from slices that use the plug-ins. Combinations of data and control plug-ins are used. The second step is to merge the plug-ins into the redirector or the programmer and, thereby, to create an evolved VNode. Plug-in interfaces and prototype plug-ins were implemented, and the evolved VNode can be used to create new types of virtual links and new methods of network accommodation.

The rest of this paper is organized as follows. Section II describes the method for virtualizing a network on the evolvable architecture platform and the independently evolvable VNode architecture. Section III describes the two proposed evolution steps, and Section IV describes the plug-in architecture for the first step of the evolution including the open VNode plug-in interface (OVPI) and control and data plug-ins. Section V first describes a prototype plug-ins that implements this architecture and then presents the results of an evaluation of this architecture. Sections VI describes related work, and Section VII gives some concluding remarks.

II. Method for Network-Virtualization

Network virtualization, the structure of a virtualization platform (i.e., a physical network), and the structure of the virtual network are described as follows.

A. Network virtualization

When many users and systems share a limited amount of resources on computers or networks, virtualization technology creates the illusion that each user or system owns resources of their own. 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,

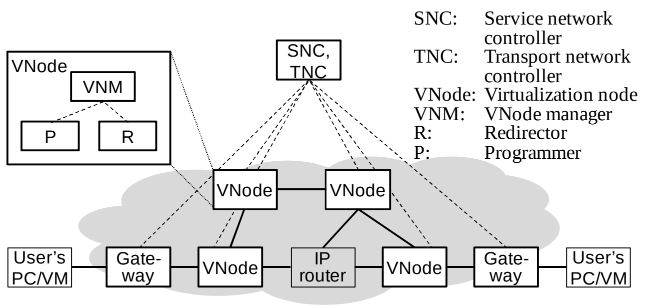

Figure 1. Physical structure of virtualization platform

networks in data centers are virtualized as VLANs, while servers are virtualized as virtual machines (VMs).

Many programmable virtualization-network research projects have been carried out, and many models, including PlanetLab [Tur 07], VINI [Bav 06], GENI [Due 12], and Genesis [Kou 01], have been proposed. Slices are created by network virtualization using a *virtualization platform* (substrate) that operates the slices.

In the VNP, network-virtualization technology was developed by Nakao et al. [Nak 10][Nak 12]. This technology makes it possible to build programmable virtual-network environments in which slices are isolated logically, securely, and in terms of performance (QoS) [Kan 13a] In these environments, new-generation network protocols can be developed on a slice without disrupting other slices.

B. Structure of virtualization platform

In the VNP, a physical network is assumed to consist of one or more domains, which are managed by a service network controller (SNC) and a transport network controller (TNC). SNC was formerly called a domain controller (DC) [Nak 12][Kan 12a]. Each domain has two types of nodes: VNode and gateway (**Figure 1**).

An overlay technology is used in the current version of VNode platform; that is, a VNode forwards packets on the platform, and each packet on the platform contains the contents of a virtual packet in a slice as the payload. VNodes are connected by tunnels using a protocol such as Generic Routing Encapsulation (GRE) [Far 00], and the Internet Protocol (IP) is used in the current version of the virtualization platform. A domain may contain conventional routers or switches that do not have virtualization functions. A slice is therefore 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. VNodes can thus be distributed to any place connected by the IP. An arbitrary packet format and protocol can be used in a slice, so they can be used in a VNode anywhere.

A VNode consists of three components: a programmer, a redirector, and a VNode manager. A programmer processes packets on slices. Slice developers can inject programs into programmers. A redirector forwards (redirects) packets from another VNode to a programmer or from a programmer to another VNode. A *VNode manager* (*VNM*) (a software component) manages the VNode according to instructions from the SNC.

C. Structure of slices

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

- *Node sliver* (virtual-node resource) represents computational resources that exist in a VNode (in a programmer). It is used for node control or protocol processing of arbitrary-format packets. A node sliver is generated by slicing physical computational resources.
- *Link sliver* (virtual-link resource) represents networking resources such as a virtual link that connects two node slivers and that any IP and non-IP protocols can be used on. A link sliver is mapped on a physical link between two VNodes or a VNode and a gateway. A link sliver is generated by slicing physical-network resources such as bandwidth.

Both node slivers and both link slivers are isolated and work concurrently, so two slices that consist of these slivers are also isolated and work concurrently.

The SNC of a domain receives an abstract slice design by using an XML-based *slice definition*. The SNC distributes the slice definition to each VNM, which sends the necessary definitions to the programmer and the redirector: the programmer receives information required for configuring a node-sliver, and the redirector receives the information required for configuring link slivers. For example, a slice definition may contain an abstract link-sliver specification such as the following link sliver with two virtual ports, i.e., end points: vport0 and vport1.

```
<linkSliver type="link" name="virtual-link-1">
 <vports>
  <vport name="vport0" />
  <vport name="vport1" />
 </vports>
</linkSliver>
```

D. Independent evolution of programmers and redirectors

An aim of the VNP is to enable mutually independent development and evolution of programmers and redirectors. Programmers consist of programmable hardware and software and implements node slivers. Redirectors consist of flexible (and maybe programmable) hardware and software and implement link slivers. It is necessary to establish modularity of these components to enable their independence; in other words, the interfaces between the components (both data-plane and control-plane interfaces) must be clearly defined.

In future, virtualization platforms will probably consist of computational and networking hardware and software developed by various vendors. A VNode may contain various types of computational components, such as Linux VMs, Microsoft Windows VMs, network processors, and GPGPUs. A network composed of VNodes may consist of various types of networking components, such as VLAN, WDM, and light paths.

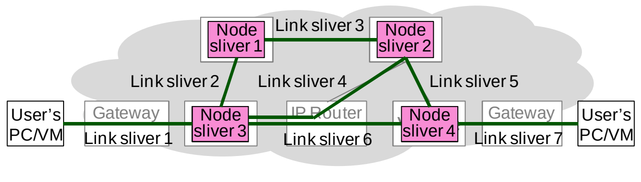

Figure 2. Example of slice design

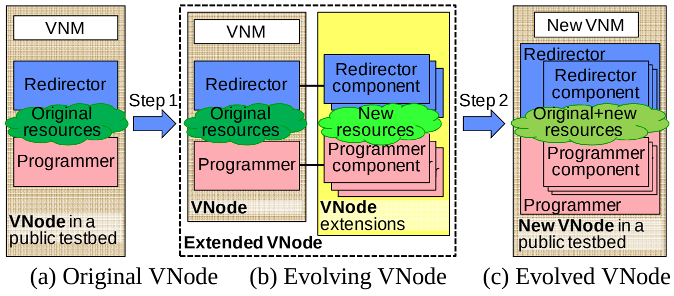

Figure 3. Proposed VNode evolution steps

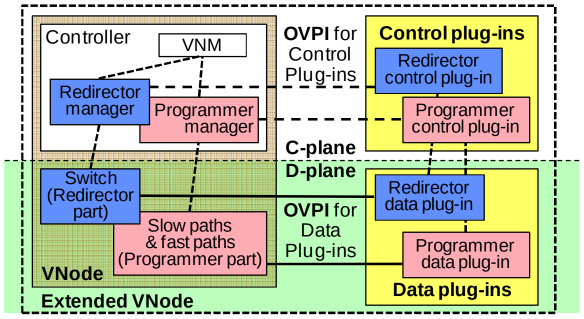

Figure 4. Open VNode plug-in (OVPI) architecture

If the interface between the software/hardware components and subcomponents is clearly defined and works efficiently, a component can be evolved independently from other components, and the components will be modular. That means they can be freely chosen and used in combination and can be freely enhanced or replaced by other components in accordance with emergence of new technology. No method for this evolution, however, has been available.

III. Proposed Evolution Steps

The proposed method for evolving VNodes is explained as follows. This method is applicable to not only VNodes but also other types of nodes for software-defined networks (SDNs). However, the architecture described in the previous section is assumed for simplicity.

As for the proposed method, redirector/programmer plug-ins are used for developing new functions, such as creating or deleting new types of virtual node or link, in two steps (see **Figure 3**). The first step is to develop new subcomponents of redirector or programmer as plug-ins and to install and to connect them to the redirector or the programmer of an existing VNode. The second step is to merge the plug-ins into the redirector or the programmer and to create an evolved VNode.

An "evolvable" VNode is created in the first step; that is, in this step, the plug-ins can be updated at any time without affecting the operation of the original VNode. Not only the redirector and the programmer in the original VNode but also the SNC and TNC (i.e., network managers) and the VNM (i.e., the management part of the VNode) remain unchanged. They manage the resources and the configuration of the original virtualization platform, but they do not manage the resources and the configuration of plug-ins.

The plug-ins can be tested by using newly created slices that specify the VNode and the plug-in. The VNode can continue services to existing slices while the plug-ins are developed because the existing slices do not use the plug-ins and are isolated from the testing slices. If the VNode has isolation function that separates packets generated by the plug-ins from packets for existing slices, the plug-ins can be tested without significant interferences with existing slices.

The resources and the configuration of the plug-ins must be managed by the plug-ins themselves. Because the resource managers are separated and the original managers do not know the new resources, the original resources and the new resources must be completely separated. The new resources may be new types of virtual node with a new type of network processor, new types of virtual link, or new types of physical subnode or link. If information on the plug-ins must be exchanged between two or more VNodes through the management components, the information should be passed through the components without interpreting or testing it. This tunneling mechanism can be called *control information tunneling* (*CIT*).

If redirectors, programmers, and the management components, i.e., SNC, TNC, and VNM, are designed to exclude interference between them and newly introduced plug-ins, a publicly available platform can be used for the development of new functions. It was intended to apply this method to JGN-X, i.e., a testbed that contains VNodes. The original VNode is probably placed in a place, such as a carrier's building, that is not easily accessible for temporary experimental purposes. However, the plug-ins can be placed in private environments for experiments, such as university laboratories or offices of vendors, and are connected by a layer-2 network such as a VLAN or a layer-2 tunnel over IP networks.

In the second step, an evolved VNode is created; that is, the plug-in functions developed in the first step are introduced into the core part of the platform. The programmer data-plug-in functions are merged into the programmer, the redirector data-plug-in functions are merged into the redirector, and the functions of control plug-ins developed in the fist step are introduced into the management components including programmer manager, redirector manager, SNC, TNC, and VNM. Because the resource managers are merged into the core part, the original and new resources are also merged. As a result, they can select the best method and resource from various methods and resources that were originally implemented in the VNode and added to it for fulfilling slice developers' requests.

IV. Open VNode Plug-in Architecture

A. Outline

The plug-in architecture described in this section is used in the first step of the VNode evolution. Plug-ins are installed and connected to a VNode using a predefined interface called an *open VNode plug-in interface* (*OVPI*), which should be built into both the programmer and the redirector of the VNode (see **Figure 4**).

There are two types of OVPI: a data-plane (D-plane) interface and a control-plane (C-plane) interface. The D-plane interface connects *data plug-ins* that handle data packets to slow-path or fast-path components (software and/or hardware components) in the case of programmer extension or to a switch (which is a part of the redirector) in the case of redirector extension. The C-plane interface connects *control plug-ins* that manage the data plug-ins and the programmer manager or the redirector manager. The data and control plug-ins are therefore used in combination. The management interface between a control plug-in and a data plug-in is a private interface, which has no predefined specification.

Plug-ins may be placed at a distant place from the VNode. A VNode may exist in a publicly available network, and the plug-ins may exist in a private environment such as a university laboratory.

Many implementation methods can be used for the OVPI. For control plug-ins, command-line interfaces (CLIs) and APIs (such as remote procedure calls or XMLs) can be used. Because an OVPI is an interface connected through networks, the host name or the (IP) address is required to identify the host node of the plug-in in the case of a control plug-in. In the case of data plug-ins, packet headers such as VLAN header or GRE can be used. Interface parameters can be passed through procedure arguments, XML tags, VLAN identifiers, GRE keys, and so on.

B. C-plane plug-in interface

The following identifiers and parameters must be specified in a control message of an OVPI for a control plug-in.

1. *Host name or address* specifies the host that contains the plug-in. In usual cases, a domain name or an IP address is used, but a non-IP address or another type of name may also be used.
2. *Plug-in identifier* specifies a plug-in in the host. This identifier may be structured; namely, plug-ins may be hierarchical.
3. *Parameters* specify control information including information that identifies the slice that the information represents. Plug-in parameters may be named or positional; that is, each parameter may have an identifier and a value or parameter values may be specified in a specific order without identifiers.

Two examples of C-plane interfaces are described here. A CLI is used in the first example. In this example, the host name or address is specified as the **ssh**/**telnet** server's domain name or address, a command name (or a file name) can be used for the plug-in identifier, and command arguments can be used for specifying parameters. For example, the following command with named parameters may be used for creating a virtual-link plug-in (see section V-C):

add_link vlan=*id* esmac=*p1* edmac=*p2* ismac=*p3*.

This command specifies a set of control information for a data plug-in; namely, it specifies an addition of a virtual link (i.e., link sliver) between external virtual ports specified as *p1* and *p2* by the specified VLAN identifier (*id*). Virtual port *p3* specifies the internal port of the node. In this example, all the parameters are named and can be specified in an arbitrary order.

The second example is as follows. The same contents are specified by an XML-based interface, such as XML-RPC [XML] or SOAP [Mit 03]. In this case, the plug-in identifier and the parameters are passed to the host as XML elements and attributes.

In general, identifiers and parameters in an OVPI must be supplied by the slice definition or the VNode (i.e., redirector, programmer, or VNM). An example of a link-sliver specification, which is similar to the link sliver shown in section II-D, is shown below. This definition contains the domain names or addresses of the control plug-ins and the physical data ports of data plug-ins. The VLAN identifier and MAC addresses are not included in this definition because they are generated by the VNodes and the control plug-ins.

```
<linkSliver type="link" name="virtual-link-1">
 <vports>
  <vport name="vport0"
   <params>
    <param key="controller" value="plug-in-0-addr" />
     <param key="port" value="data-plug-in-0-port"/>
     <!-- Additional parameters -->
   </params>
  </vport>
  <vport name="vport1"
   <params>
     <param key="controller" value="plug-in-1-addr" />
     <param key="port" value="data-plug-in-1-port"/>
     <!-- Additional parameters -->
   </params>
  </vport>
 </vports>
 <params>
  <param key="ExtensionName" value="vlan_link" />
  <!-- Additional parameters -->
 </params>
</linkSliver>
```

C. D-plane plug-in interface

The following parameters must be specified in a data packet as an OVPI for a data plug-in.

1. *Plug-in channel tag*: In contrast to the C-plane interface, a host and a plug-in are not specified separately. A tag, which may be a protocol parameter such as a VLAN identifier, specifies a channel or a collection of plug-ins. Multiple plug-ins specified by a tag may be in one host or distributed to multiple hosts connected by a network channel (such as a VLAN).
2. *Parameters*: Plug-in parameters are specified as protocol parameters. Some parameters identify the slice of the data path that the plug-in implements. Some parameters may be used for identifying a plug-in among the plug-ins specified by the plug-in channel tag.

Two examples of D-plane interfaces are described here. In the first example, a VLAN is used for the D-plane protocol. In this case, the plug-in channel tag may be specified as a VLAN identifier. The parameters, which represent the end-point addresses of the virtual link, are expressed as source and destination MAC addresses. If only one (or a few) VLAN identifier can be used or if no tagged VLANs can be used, plug-ins may be distinguished by a set of MAC addresses; in other words, they can be used for specifying both a plug-in tag and parameters.

In the second example, GRE/IP is used for the D-plane protocol. In this case, the plug-in tag is represented by a key in the GRE header, and the parameters are represented by addresses in the IP header.

V. Prototyping and Evaluation

A version of the OVPIs was implemented, and two sets of plug-ins were installed and connected by using the OVPIs and partially evaluated. The hardware and software for the OVPIs and the plug-ins are described below first; the design, implementation, and preliminary results of an evaluation of the plug-ins are described after that.

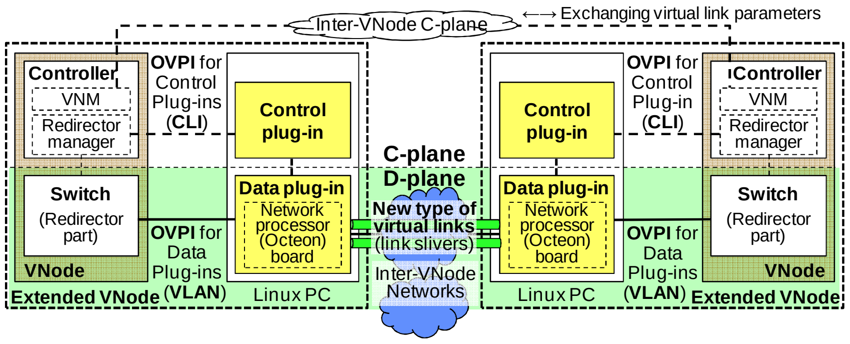
Inter-VNode C-plane
Exchanging virtual link parameters
Controller
VNM
Redirector manager
OVPI for Control Plug-ins (CLI)
Control plug-in
C-plane
D-plane
Switch
(Redirector part)
OVPI for Data Plug-ins (VLAN)
Data plug-in
Network processor (Octeon) board
New type of virtual links (link slivers)
Inter-VNode Networks
VNode
Extended VNode
Linux PC

Figure 6. VNode plug-in and interaction architecture for extension of a virtual link

A. Hardware and software environment for plug-ins

The prototype system and the environment used for prototyping and evaluation are described as follows. A preliminary version of the OVPIs is implemented in the redirectors of the VNodes. A CLI is used for the C-plane interface, and a VLAN-based interface is used for the D-plane. Data plug-ins are implemented in two sets of PCs with CentOS (Linux). Each PC has a PCIe board with a network processor, Cavium Octeon [Cav 10]. This board is called WANic-56512 (developed by General Electric Company). An open and high-level language called CSP (Continuous Stream Programming) and its development environment, “+Net,” for Octeon [Kan 13b] is used for developing the plug-in programs. Control plug-ins are implemented in the PCs.

Two sets of plug-ins were developed. The first set of plug-ins, a control plug-in and a data plug-in, implements a network-accommodation function that connects a slice to an external network through a VLAN, and the second set of plug-ins implements VLAN-based virtual links (link slivers) between VNodes.

B. Re-implementing network-accommodation function

The first set of plug-ins implements a network accommodation function that connects a slice of the VNode platform to an external network through a VLAN (see **Figure 5**). This function is similar to that of “network accommodation equipment” (NACE or NC) [Kan 12b], which is built into the VNode platform. However, this function is re-implemented to test the plug-in architecture and the network-accommodation-function implementation.

The data plug-in converts the packet format; that is, the packet format for the external network is X/Ethernet, where X is usually IP but other protocols can also be used, and the internal format for a VNode is X/Ethernet/Ethernet. The outer MAC header contains the platform parameters, and X/Ethernet (including the inner MAC header) is the packet format for the slice.

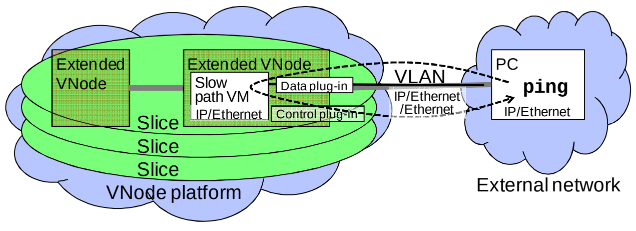

Figure 5. Re-implementation of network-accommodation function

As shown in Figure 5, successful IP communication between a PC in the external network and a VM in the virtual node was confirmed by a **`ping`** command. The performance of the whole prototype system, which contains two VNodes and plug-ins, has not yet been measured. However, the performance of the data plug-in implemented on the Octeon board was measured. When packet size was sufficiently large, i.e., 600 bytes or larger, the throughput was measured to be 8 Gbps or more, namely, close to the wire rate, i.e., 10 Gbps.

C. Implementing a new type of virtual link

The second set of plug-ins implements a new type of virtual link. GRE-based virtual links are the only type available in the current version of VNodes. VLAN-based virtual links are thus implemented by using the plug-ins. The architecture for the VLAN-based virtual link is shown in **Figure 6**, and the packet formats and example contents are shown in **Figure 7**. To separate a programmer from the network and other programmers, internal MAC addresses of the programmer, which are part of the data plug-in interface, must be hidden outside of the programmer [Kan 12a]. The redirector data plug-in therefore swaps the MAC addresses in data packets as shown in Figure 7.

To operate a virtual link correctly, control plug-ins in two VNodes, which are the end-points of the virtual link, must exchange control parameters through the inter-VNode C-plane (see the top of Figure 6). The end-point addresses in the control parameters identify the slice to which the virtual link belongs. This negotiation should be performed by the VNode managers (VNMs) of the VNodes when a virtual link is created or deleted. However, currently they only have negotiation function of

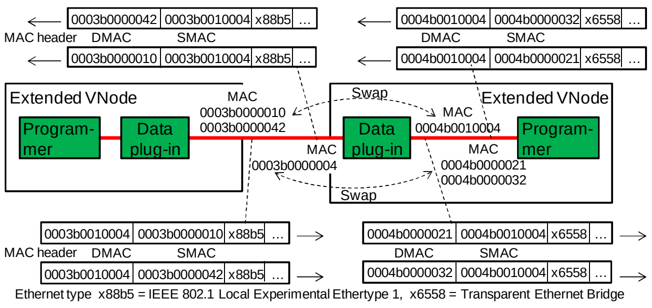
0003b0000042 | 0003b0010004 | x88b5 | …
MAC header DMAC SMAC
0003b0000010 | 0003b0010004 | x88b5 | …
0004b0010004 | 0004b0000032 | x6558 | …
DMAC SMAC
0004b0010004 | 0004b0000021 | x6558 | …
Extended VNode
Program-mer
Data plug-in
MAC 0003b0000010 0003b0000042
MAC 0003b0000004
Swap
Swap
Extended VNode
Data plug-in
MAC 0004b0010004
MAC 0004b0000021 0004b0000032
Program-mer
0003b0010004 | 0003b0000010 | x88b5 | …
MAC header DMAC SMAC
0003b0010004 | 0003b0000042 | x88b5 | …
0004b0000021 | 0004b0010004 | x6558 | …
DMAC SMAC
0004b0000032 | 0004b0010004 | x6558 | …
Ethernet type x88b5 = IEEE 802.1 Local Experimental Ethertype 1, x6558 = Transparent Ethernet Bridge

Figure 7. Packet formats for the VLAN-based virtual link

GRE-based virtual links. Therefore, in this preliminary and temporary implementation, the GRE-based link parameters, i.e., IP addresses (and a GRE key), are passed to the control plug-ins and they are converted to the VLAN-based link parameters, i.e., MAC addresses (and a VLAN ID). In a future version of redirector plug-in architecture, VNMs should implement a tunneling mechanism, i.e., CIT. The VNMs can exchange VLAN-based link parameters or any other type of control information that control plug-ins manage using CIT.

Successful IP communication between the virtual nodes connected by the VLAN virtual-link was confirmed by a **`ping`** command; although virtual links in VNodes can transmit arbitrary format packets such as IPEC packets [Kan 12c], IP was used because it requires only two commands (i.e., **`ifconfig`** and **`ping`**) built into the virtual node. The performance of the whole prototype system was not measured, but the throughput of the data plug-in was measured to be 9 Gbps or more when the packet size was 900 bytes or larger.

VI. Related Work

Click [Koh 00] is a software architecture that uses two-level description for describing routers modularly. The lower-level components, which are described in C, can be regarded as plug-ins. The higher level is described in a domain-specific language, which connects modules in several ways. Both data and control plug-ins may be described by using Click; however, Click is suited to (data) packet processing but not well suited to control processing and hardware plug-ins.

OpenFlow [McK 08] enables easy implementation and extension of network control. It is easy to use OpenFlow to design a plug-in architecture for management and control. It cannot, however, be used to implement data plug-ins.

Active networks enabled ad hoc extension of data paths. Capsules, or active packets [Wet 98], which are packets containing programs, may be regarded as a temporary plug-ins. There are, however, two issues concerning capsules. First, capsules are not suited to repeatedly used functions because of redundancy; that is, multiple packets contain the same program. Second, they cannot be used for hardware plug-ins. Other types of active networks, such as SwitchWare [Ale 98], solve the first issue but not the second one.

In contrast to OpenFlow and active networks described above, the plug-in architecture proposed in this paper can be used for both control and data plug-ins, and for both software and hardware plug-ins.

VII. Conclusion

A method for evolving programmer and redirector, i.e., computational and networking components of a VNode, independently was proposed and tested. This method is composed of two steps. In the first step, plug-in interfaces called "open VNode plug-in interfaces" (OVPIs) for both data and control plug-ins are used. These OVPIs are built in both the programmer and the redirector of VNodes.

A prototype of OVPIs and plug-ins were developed and evaluated. The evolved VNode can implement new types of network accommodation functions and can create new types of virtual links. The throughput of the network accommodation and the VLAN-based virtual links is close to a wire rate of 10 Gbps. This result means that the first step of VNode evolution was succeeded for these new functions.

Future work includes implementing CIT to the VNM and implementing new types of virtual links and network accommodation methods, including non-IP-protocol based ones, using advanced technologies and methods. It also includes applying this method, including the second step, to VNodes in JGN-X.

Acknowledgments

Part of the research results is an outcome of "Advanced Network Virtualization Platform Project A" funded by the National Institute of Information and Communications Technology (NICT). The author thanks Kazuhisa Yamada from NTT, Akihiro Nakao from the University of Tokyo, Toshiaki Tarui from Hitachi, and other members of the above project for their valuable discussions on the VNode evolution process. The author also thanks Yasushi Kasugai, Kei Shiraishi, Takanori Ariyoshi, and Takeshi Ishikura from Hitachi for implementing the plug-in interfaces in the redirector.

References

[AKA 10] AKARI Architecture Design Project, "New Generation Network Architecture — AKARI Conceptual Design (ver 2.0)", May 2010.

[Ale 98] Alexander, D. S., Arbaugh, W. A., Hicks, M. W., Kakkar, P., Keromytis, A. D., Moore, J. T., Gunter, C. A., Nettles, S. M., and Smith, J. M., "The SwitchWare Active Network Architecture", *IEEE Network*, Vol. 12, No. 3, pp. 29–36.

[Aoy 09] Aoyama, T., "A New Generation Network: Beyond the Internet and NGN", *IEEE Communication 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", *SIGCOMM 2006*, pp. 3–14, September 2006.

[Cav 10] "OCTEON Programmer's Guide, The Fundamentals", Cavium Networks, 2010, http://university.caviumnetworks.com/downloads/-Mini_version_of_Prog_Guide_EDU_July_2010.pdf

[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-Virtualization Nodes that Support Mutually Independent Development and Evolution of Components", *IEEE International Conference on Communication Systems* (*ICCS 2012*), November 2012.

[Kan 12b] 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.

[Kan 12c] Kanada, Y. and Nakao, A., "Development of A Scalable Non-IP/Non-Ethernet Protocol With Learning-based Forwarding Method", *World Telecommunication Congress 2012* (*WTC 2012*), March 2012.

[Kan 13a] Kanada, Y., Shiraishi, K., and Nakao, A., "Network-resource Isolation for Virtualization Nodes", *IEICE Trans. Commun.*, Vol. E96-B, No. 1, pp. 20-30, 2013.

[Kan 13b] Kanada, Y., "Open, High-level, and Portable Programming Environment for Network Processors", *IEICE 7th Meeting of Network Virtualization SIG*, July 2013 (in Japanese).

[Koh 00] Kohler, E., Morris, R., Chen, B., Jannotti, J., and Frans Kaashoek, M., "The Click Modular Router", *ACM Transactions on Computer Systems* (*TOCS*), Vol. 18, No. 3, pp. 263–297, 2000.

[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.

[McK 08] McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J., "OpenFlow: Enabling Innovation in Campus Networks", *ACM SIGCOMM Computer Communication Review*, pp. 69–74, Vol. 38, No. 2, April 2008.

[Mit 03] Mitra, N., and Lafon, Y., "SOAP version 1.2 part 0: Primer", W3C Recommendation 24 (2003): 12.

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

[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.

[Wet 98] Wetherall, D., et al. "ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols", *1st IEEE Conference on Open Architectures and Network Programming* (*OPENARCH'98*), pp. 117–129, April 1998.

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