Extending Network-virtualization Platforms by using a Specialized Packet Header and Node Plug-ins

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

***Abstract* – A previously developed plug-in architecture for network-virtualization nodes allows network operators to introduce new types of virtual nodes and links and slice developers to use them in *slices* (i.e., virtual networks). In this paper, a method for extending network-virtualization infrastructures by introducing plug-ins to nodes in the infrastructure and a freely-designed plug-in-specific packet header, which enable sharing part of packet contents among the same type of plug-ins distributed in the infrastructure, is proposed. The header is inserted into every data packet handled by the nodes, but it is hidden from slices in a "clean virtualization" infrastructure. This method was applied to creation of a new type of virtual links with network-delay measurement function using a hidden timestamp in each packet. The timestamps do not affect slices; that is, conventional programs can be used in the slice for the measurement without modification. The method was evaluated by edge-to-edge delay measurements and the evaluation results show that it is suitable for developing new functions, including functions requiring wire-rate performance, in shared/public networks.**

***Keywords* – Network-node evolution, Network-node plug-in architecture, Network-delay measurement, Timestamp, Network virtualization, Virtualization node, VNode, Virtual-link-type creation, Deep programmability.**

I. INTRODUCTION

Development of new communication services will help global human society to evolve. To enable simple, flexible, and cost-reduced service creation, network virtualization will play an important role by introducing two concepts, i.e., "slice" and "slice developer." As for the first one, network virtualization enables creation of various types of virtual networks, which are called *slices*, on a single physical network. Slices not only reduce development and maintenance costs of customized networks but also realize simpler and flexibly customizable networks. As for the second concept, network virtualization creates a new role, called *slice developer* [Yam 12], in addition to the two conventional roles of a network: *operator* and *user*. Slice developers develop slices and operate the slices for end users. In this three-role model, an operator may be called an *infrastructure provider*, and a slice developer may be called a *service provider* [Cho 09]. Slice developers create a slice by selecting types of virtual nodes and links that are supported by the network-virtualization infrastructure, and they can program virtual nodes (and virtual links) if they are programmable.

However, slice developers cannot introduce new types of virtual nodes or links as well as new hardware and software into the network infrastructure by themselves, so a method for evolving a node was developed [Kan 13c]. Although the infrastructure may sufficiently support various types of nodes and links at the time of platform development, new types of hardware and software, which can be used for building new types of virtual nodes or links, will become available through technical innovations. A method that allows the operator (or the vendor) to introduce new types of virtual nodes and links by evolving the infrastructure was therefore developed [Kan 13c][Kan 14]. For this method, first, new types of virtual nodes and links are introduced by developing data and control plug-ins. The control plug-ins are then integrated into original control and management components [Kan 14]. The plug-in architecture developed in these previous studies was implemented in the virtualization node (VNode) and VNode infrastructure [Nak 12][Kan 12], which were developed by the Virtualization Node Project (VNode Project) [Nak 10].

In the present study, a method for introducing node plug-ins and a freely-designed plug-in-specific packet header is proposed because this type of packet header greatly extends the potential of infrastructure evolution. A data plug-in may be used for processing predefined packet contents, but it may also be used for processing content of a newly introduced header. Such a header, which is inserted into every data packet handled by the plug-ins, can be hidden from slices if the virtualization infrastructure is "cleanly virtualized" [Kan 12]. In addition, the header is not referenced by the original infrastructure. The format of the header can therefore be freely designed according to the requirements for the data plug-in. The data plug-ins can be distributed to nodes in the virtualization infrastructure and can share the content of the header; that is, some plug-ins write content and others read and update it. This method is applied to measurements of network edge-to-edge delay by using a hidden timestamp in each packet. The timestamps do not affect slices; that is, conventional protocol processing and application programs can be used on the slice for the measurement without modifying them. In addition, they do not affect the original infrastructure either.

The rest of this paper is organized as follows. Section II describes related work. Section III reviews a network-virtualization infrastructure architecture that can be extended (evolved) by adding plug-ins. Section IV describes a method for handling platform-specific or plug-in-specific information (packet headers) in virtual networks. Section V describes a method for packet identification in programmable networks. Section VI proposes a method for plug-in-specific header handling (including timestamp handling) with the packet identification method. Section VII evaluates the proposed packet-handling method, and Section VIII concludes the paper.

II. RELATED WORK

Related work on programmability of network nodes and networks and network-delay measurements is summarized in this section.

A. Network-node programmability

OpenFlow [McK 08] and other software-defined networking (SDN) technologies enable separation of control and data, control-plane programming, and centralized network control. Network infrastructures (including network nodes) can be virtualized, and the control plane can be evolved by using these technologies. However, conventional SDN does not support data-plane programmability and programmability of decentralized control. These functions are also required in

future networks, so they are supported by the plug-in architecture proposed in this paper. They can be used for a combination of new control-plane and data-plane functions, which are separated but can be flexibly integrated. Moreover, the plug-in architecture supports new functions realized by combinations of software and hardware. Although new software is focused on in regard to future networks, new hardware will also be required.

Although OpenFlow is designed to handle IP/Ethernet packets, it can also handle non-standard protocols. However, OpenFlow cannot be used for *stateful* processing of packets with a plug-in-specific packet header, which is focused on in this paper.

JUNOS® SDK [Kel 09] of Juniper Networks supports service components as plug-ins. Each plug-in consists of control and data components. However, in this architecture, only one instance of a service component is created. This architecture differs from the plug-in architecture proposed in this paper, which supports creation of a type of virtual node or link by network operators (or vendors) and enables creation of multiple instances of components by slice developers.

B. Delay measurements

It is important to measure network communication delay between end hosts or a server and a client because delay is an important measure of QoS. Delay can be measured either directly, i.e., by using timestamps in packets, or indirectly, i.e., by using network tomography or a similar method. In the case of direct methods, probing packets, such as "ping", i.e., a type of packet of ICMP, are used. Although probe packets are known to normally behave in a similar way to application packets, sometimes they may behave differently [Che 03]. Especially, even if the behaviors are similar, the delay distributions may be different.

Several types of application packets may contain timestamps. For example, in regard to the real-time transport protocol (RTP), every packet can contain a timestamp. The timestamp is inserted by the sender application and tested by the receiver application; it is thus used only for end-to-end measurements. A more generic (but non-virtualized) method using timestamps by using OpenFlow is proposed [Adr 14].

Direct methods, however, usually use probe packets unless a specific application uses a packet format that contains a timestamp. Application packets are usually not allowed to contain timestamps because both TCP/IP or UDP/IP protocols and applications do not handle timestamps and, if timestamps are inserted, these protocols and applications cannot process packets correctly. In addition, standard headers, such as Ethernet MAC headers, IP headers, and GRE headers, can be added easily by modifying operating-system (OS) configurations; however, non-standard headers, such as those containing timestamps, cannot easily be added by using OS functions. A method for using such non-standard headers in a network is thus proposed in the following.

III. EVOLVABLE NETWORK-VIRTUALIZATION INFRASTRUCTURE

Network virtualization and the basic VNode architecture, which enable mutually independent development of computational and networking components of VNodes [Nak 12][Kan 12], and an evolving VNode architecture by using plug-ins [Kan 13c][Kan 14] are reviewed.

A. Network virtualization and VNode

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. Many programmable virtualization-network research projects have been carried out, and many models, including PlanetLab [Tur 07], VINI [Bav 06], GENI [Ber 14], and Genesis [Kou 01], have been proposed. Slices are created by network virtualization using a *virtualization infrastructure* (substrate) that operates the slices.

In the VNode Project, 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. Method for node evolution

The method for evolving VNodes [Kan 13c][Kan 14] is reviewed here. Using this method, operators (or vendors) can use plug-ins for developing new functions, such as creating, operating, or deleting new types of virtual node or link, for slice developers. The operator first develops new subcomponents of the original components of the VNode as *plug-ins* and connects them to the original components. Plug-ins consists of hardware and software. The operator then merges the plug-in functions into the VNode (original components) to create an evolved VNode.

This node-evolution method makes it possible to update the plug-ins (i.e., the VNode can be evolved) at any time without affecting the operation of the original VNode. As well as the data-plane components in the VNode, the network managers and control-plane components of the VNode remain unchanged. The latter manage the resources and the configuration of the original virtualization infrastructure; however, they do not manage the resources and configurations of plug-ins.

The resources and configurations of a plug-in with data-plane functions, which is called a *data plug-in*, must be managed by another plug-in, which is called a *control plug-in*. A data plug-in may contain specialized hardware required for assuring high performance, isolation, and QoS of slices. Plug-ins are connected to a VNode by using a predefined interface called an open VNode plug-in interface (OVPI) [Kan 13c], which should be built into the VNode (see **Figure 1**).

To finalize the VNode evolution, an evolved VNode is created (this final stage was called the "second step" in the previous papers [Kan 13c][Kan 14]); that is, the plug-in functions are introduced into the core part of the infrastructure. The data-plug-in functions of the programmer are merged into the data-plane components and the functions of control plug-ins are introduced into the control-plane components. Slice developers can use the new function in a similar way as they use other functions. A method for finalizing the evolution at reduced cost was proposed in a previous paper [Kan 14]. By using this method, a slice developer can use new types of virtual nodes and links, which are implemented by plug-ins, using the same syntax as built-in types in a slice definition. In addition, the plug-ins are registered to the network manager that authenticates the plug-ins, and the operator authorizes them.

The proposed plug-in architecture supports data-plane programmability and programmability of decentralized control in addition to programmability of centralized control, which is also supported by conventional software-defined networking (SDN) technologies. The data-plane programmability of a data plug-in and the control-plane programmability of a control plug-in are combined to implement a new type of virtual node or link. The data-plane and control-plane functions are

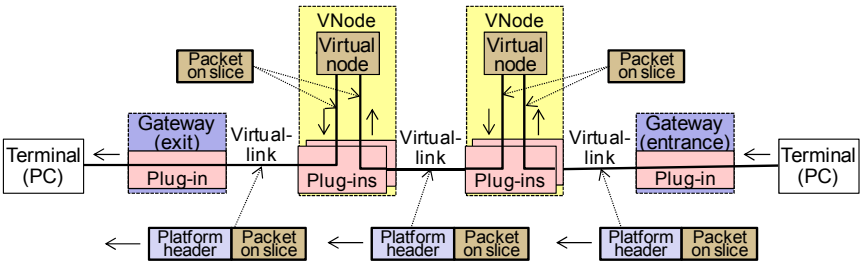


Figure 1. A virtualization network with a type of plug-ins

separated, but they can be flexibly integrated; that is, the management interface between a control plug-in and a data plug-in can be a private interface, which has no predefined specification. Moreover, the plug-in architecture supports new functions created by combinations of software and hardware.

IV. Handling Platform/Plug-in Specific Information in Virtual Networks

In virtualization networks based on an overlay technology, packets contain a platform header, which contains platform-specific information such as a slice and virtual-link identifiers. Platform-specific data can be classified into three categories: standard-header data, standard addresses, and free-form data.

The standard-header data are contained in a standard tunneling header. When a tunneling method is used for virtualization, a tunneling header, such as generic routing encapsulation [Far 00] over IP (GRE/IP), can be used. For example, in the network virtualization using Generic Routing Encapsulation (NVGRE) [Sri 14], a GRE key can be used for identifying a virtual network. Instead, if a VLAN is used for virtualization, a VLAN header can be used for identifying it.

The standard addresses are contained in address fields in a standard header. In addition to a GRE key or a VLAN ID, additional information may be conveyed by the IP addresses in an IP header or by the MAC addresses in an Ethernet header. In a VNode infrastructure, which features clean virtualization, the IP addresses in an IP header are used for identifying virtual links in combination with the GRE keys.

The free-form data may be contained in a header with non-standard form. The platform header can be in any non-standard format and contain more information than can be contained by a standard header format such as GRE or VLAN. For example, a packet may contain a timestamp of any length or other measurement data. A non-standard format can be used in a clean virtualization infrastructure because the platform headers are hidden from slices if they are cleanly virtualized.

A free-form header may be divided into multiple fields, and if one or more data plug-ins exists, the platform header may contain plug-in-specific headers, as shown in **Figure 2**. The plug-in-specific header may be set and tested by a plug-in at any location in the virtualization infrastructure. The numbers of plug-ins and headers may vary, and the orders of headers

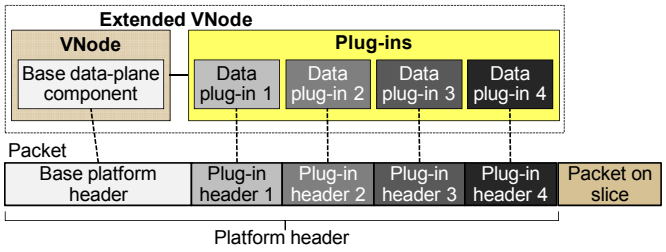


Figure 2. Data plug-ins and plug-in-specific headers

and processing may vary, but this variation is out of scope of this paper.

As shown in **Figure 3**, plug-ins in entrance gateways insert a platform header that contains a plug-in-specific header (which may contain a timestamp), VNodes in the infrastructure handle the header by the above-described method, and plug-ins in exit gateways handle (may measure the delays) and remove the header. In the figure, two terminals communicate with each other via a slice. The physical network consists of two measurement gateways and two VNodes. Each VNode contains a virtual node on the slice. The virtual nodes are connected by a special type of VLAN-based virtual-link that supports a specialized function such as delay-measurement. The virtual links between VNodes and gateways are of the same type. In this infrastructure, each packet has a non-standard platform header.

Although plug-ins can use any packet-header format in the proposed method, it may be better to use a standard format such as the network service header [Qui 13] when it is available. The network service header, which is being standardized in the IETF, can be added to encapsulated packets to specify service paths in a network and to carry metadata used by network devices or services. However, at least in the case of the delay-measurement applications described in this paper, service paths are not stored in platform headers.

To virtualize the network "cleanly", the timestamp must be removed (or hidden) just before the packet is received by a virtual-node program on a slice, and it must be added again just after the packet is sent by the program because the timestamp may not affect in-slice processing. A method for cooperatively handling plug-in-specific information in plug-ins, especially delay measurement, is described in Section VI, and needs concerning packet identification and a packet-identification method, which is required for header handling in programmable networks, are described in the next section.

V. Packet Identification in Programmable Networks

To identify input and output packets, either the program or the developer of the program (i.e., the slice developer) must specify the method of identifying them. If a virtual node works only as a router or a switch, which do not absorb, generate, or duplicate packets, input packets to the node and the output packets from the node may be easily identified by the virtualization infrastructure.

These packets, however, cannot be identified if the virtual node is programmable because the infrastructure does not know the correspondence between input and output packets (see **Figure 4)**. A programmable node can absorb input packets (Figure 4(a)) and generate new packets that do not correspond to input packets (Figure 4(b)). It can also generate multiple copies of a packet (Figure 4(c)). In addition, the formats of input and output packets may be completely different. The identity of input and output packets depends on the program on the virtual node, and it cannot be defined without certain assumptions concerning the program.

To identify the input and output packets, either the program or the developer of the program must therefore specify the identity. For example, they can specify a field $p.f$ of packet p, which contains a value that can identify the packet, or they can specify a function $f(p)$ that inputs a content of packet p and outputs its identifier.

If packets must be processed quickly by the infrastructure, e.g., they must be forwarded at wire rate, the method for

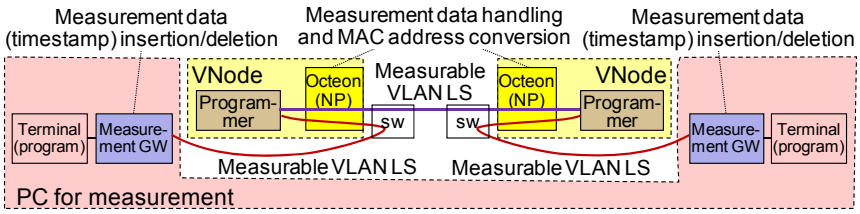


Figure 5. Physical network-structure used for experimentally measuring delay

packet identification must be simple. The most promising method is to include a packet identifier in each packet. Although this method may be restrictive in some situations, packets can usually be identified by their contents. When this method is used, the program in a node or the developer must specify the location and the length of the identifier field in the packet. For example, if TCP is used, the sequence number can be used as the identifier.

VI. Plug-in-specific Header Handling

The proposed method for processing a plug-in-specific header, especially a timestamp in the header, by the plug-ins in the Vnodes is described as follows.

A. Handling a plug-in-specific header in a VNode

Because a VNode must satisfy the clean virtualization criteria, it must hide plug-in headers by using an appropriate method; i.e., a method for storing and retrieving plug-in headers must be developed. It is required that a VNode must hide its platform header from virtual nodes implemented in the VNode. For example, if a plug-in-specific header contains a timestamp and the header is hidden from the slice, e.g., they are removed by the infrastructure before processing by programs on the slice, conventional protocol processors and applications that do not handle timestamps work normally on the slice. The VNode must, therefore, remove the platform header when it sends a packet to the virtual nodes, and it must add a platform header when it receives a packet from the virtual nodes. To restore the packet header, the VNode must identify the packet. As described in the previous section, the most promising method for this identification is to use a packet identifier in the packet (on the slice). The performance of the plug-in to provide these header-handling functions should be high. A network processor may thus be used for implementing these functions.

The method for handling a plug-in-specific header differs from that for handling the base platform header that contains virtual-network and -link information in the following way. If the only contents of the packets are virtual-link-related information, such as virtual-network or -link identifiers, the input and output platform headers are independent. However, if they are related, in such a case as delay measurement, the packet header must be restored when outputting the packet.

B. Handling timestamps in the VNode infrastructure

An application of the proposed method is measurement of gateway-to-gateway communication delay. Timestamps can be inserted to a plug-in-specific header and handled by corresponding plug-ins in all the VNodes and gateways. They do not affect the protocols and applications used on slices because they are hidden from the slice.

An example of the physical network structure used for this measurement is given in Figure 3. The timestamp is inserted at the entrance gateway. Each VNode generates a packet for the virtual node by removing the platform header from an incoming packet, and the VNode restores the timestamp to one or more outgoing packets that comes from the virtual node and are identified with a stored incoming packet. The delay of these removal and restoration functions must be small, and the throughput of these functions should be high. The timestamp is tested and deleted at the exit gateway. This gateway calculates the delay (the average and distribution) between the entrance and exit gateways. Their clocks must be synchronized as exactly as possible to measure sub-millisecond-order delay.

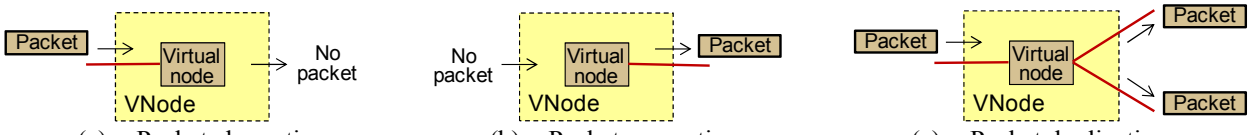
VII. Evaluation

The proposed method was evaluated by developing a measurable VLAN virtual link (MVL), which is a new type of virtual link with delay-measurement functions. The MVL is added to the VNode infrastructure by using a plug-in architecture for VNodes [Kan 13c]. MVL-creation requests are handled by the management components of VNodes and by the control plug-in, which is an MVL-specific control program.

The experimental network used for this evaluation is as follows. In the physical network described in **Figure 5**, the two VNodes, which are placed in the same room, and one PC is used for the two gateways (and terminals) to avoid any problems with clock synchronization. Terminal PCs communicate with each other by using Ethernet (but not IP/Ethernet) packets, which are switched by the MAC addresses on the slice in the virtual nodes. An Ethernet switch program, which is a slow-path program running on Linux (CentOS), works on a virtual node in each VNode.

The computational and networking environments for the data plug-ins in the VNodes and the gateways are described as follows. A network-processor board called WANic-56512 (developed by General Electric) was used for each VNode. This board contains twelve 750-MHz Cavium Octeon® Plus cores, which contains the program, and handles both incoming and outgoing packets. The data plug-in for VNodes was programmed by a hardware-independent language for network processing, which is called "Phonepl" (portable high-level open network-processing language) [Kan 13b]. This program is handled by a +Net development environment [Kan 13b] that consists of a Linux (CentOS), a Phonepl compiler, run-time routines, and a GNU C compiler for Octeon. The "data plug-in" for gateways was programmed in C and compiled by a GNU C compiler for Linux (CentOS).

Although the platform header can be modularized as shown in Figure 2, the base header and the timestamp are handled by a single program in this implementation. The size of the platform header and the displacement and size of the



(a) Packet absorption (b) Packet generation (c) Packet duplication

Figure 4. Non-corresponding packet generation by a virtual-node program

timestamp are embedded in the plug-in.

The Phonepl program also swaps the VNode-external and VNode-internal MAC addresses in the platform header. The reason that swapping MAC addresses is required is explained in a previous paper [Kan 12], but it is out of scope of this paper. To swap addresses, the program contains a conversion table for internal and external MAC addresses and accepts virtual-link-creation and deletion requests. A creation request adds an entry to the conversion table. The entry contains a pair of internal and external MAC addresses.

The gateway-to-gateway delay, the throughput of timestamp handling and conversion, and the program lengths were measured. The evaluation results of the delay is 178 μS (σ = 24 μS). If entrance and exit gateways are separated, it is hard to measure delay of this order. **Table 1** shows the other results. It compares the performances of the Phonepl program for Octeon and the C program for 3-GHz Xeon. Although the overhead of storing and searching for the packet header is not light, the former performance of the Phonepl program is very close to 10-Gbps wire rate. Although the C program is relatively short because the conversion-table configuration code is omitted, the Phonepl program, which contains it, is still much longer.

VIII. Conclusion

A method for extending a virtualization network infrastructure by introducing node plug-ins and a freely-designed plug-in-specific packet header, which enable sharing part of packet contents among the same type of VNode plug-ins spread around the infrastructure, is proposed. This method was applied to measurements of network edge-to-edge delay by using a hidden timestamp in each packet. The timestamps for multiple do not affect slices because the VNode infrastructure is "cleanly" virtualized; that is, conventional programs can be used in the slice for the measurements without modifying them. This method was evaluated on the basis of the delay measurements, and the evaluation results show that the throughput of timestamp handling and conversion is 10-Gbps wire rate and that the latency caused by the measurement is less than 100 μS. This method is suitable for developing new functions, including functions requiring wire-rate performance, in shared/public networks. Future work includes implementation of other node/link functions and handling multiple plug-in-specific headers.

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 Akihiro Nakao from the University of Tokyo, Kazuhisa Yamada from NTT, Toshiaki Tarui from Hitachi, and other members of the above project for their valuable discussions on the proposed packet-header handling and delay-measurement methods. The author also thanks Yasushi Kasugai, Kei Shiraishi, Takanori Ariyoshi, and Takeshi Ishikura from Hitachi for implementing the plug-in interfaces.

Table 1. Evaluation results on timestamp (TS) handling and header conversion

Implementation	Throughput (Gbps)*		Program lines
	TS insertion	TS deletion	
Phonepl program	10.0†	9.5†	99‡
C program (Xeon)**	2.3† (4.0††)	2.2† (4.0††)	190‡

*Packet size: 1024 B. **Promiscuous mode is used. †No packet loss ($< 10^{-6}$)
†† Packet loss ratio = 10^{-3} ‡Comment-only lines are not counted.

References

[Adr 14] Adrichem, N. van, Doerr, C., Kuipers, F., "OpenNetMon: Network Monitoring in OpenFlow Software-Defined Networks", *IEEE/IFIP NOMS 2014*, May 2014.

[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.

[Ber 14] Berman, M., Chase, J. S., Landweber, L., Nakao, A., Ott, M., Raychaudhuri, D., Ricci, R., and Seskar, I., "GENI: A Federated Testbed for Innovative Network Experiments", *Computer Networks*, Vol. 58, January 2014.

[Cas 04] Castro, R., Coates, M., Liang, G., Nowak, R., and Yu., B., "Network Tomography Recent Developments", *Statistical Science*, Vol. 19, No. 3, pp. 499–517, 2004.

[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

[Che 03] Chen, Y., Bindel, D., and Katz, R. H., "Tomography-based Overlay Network Monitoring", *ACM SIGCOMM Conference on Internet Measurement* (*IMC'03*), October 2003.

[Cho 09] Chowdhury, N. M. M. K. and Boutaba, R., "Network Virtualization: State of the Art and Research Challenges", *IEEE Communications Magazine*, Vol. 47, No. 7, pp. 20–26, July 2009.

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

[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 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).

[Kan 13c] Kanada, Y., "A Node Plug-in Architecture for Evolving Network Virtualization Nodes", *2013 Software Defined Networks for Future Networks and Services* (*SDN4FNS*), November 2013.

[Kan 14] Kanada, Y., "A Method for Evolving Networks by Introducing New Virtual Node/link Types using Node Plug-ins", *1st IEEE/IFIP International Workshop on SDN Management and Orchestration*, May 2014.

[Kel 09] Kelly, J., Araujo, W., and Banerjee, K., "Rapid Service Creation using the JUNOS SDK", *ACM Workshop on Programmable Routers for Extensible Services of Tomorrow 2009* (*PRESTO'09*), pp. 7–11, 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.

[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.

[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

[Qui 14] Quinn, P., Agarwal, P., Manur, R., Fernando, R., Guichard, J., Kumar, S., Chauhan, A., Smith, M., Yadav, N., and McConnell, B., "Network Service Header", draft-quinn-sfc-nsh, work in progress, IETF, 2013.

[Sri 14] Sridharan, M., et al., "NVGRE: Network Virtualization using Generic Routing Encapsulation", *draft-sridharan-virtualization-nvgre*, work in progress, IETF.

[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.

[Yam 12] Yamamoto, T., Katayama, Y., Yamada, K., and Nakao, A., "A Management Model for the Network Virtualization Platform to Provide Network Programmability", *World Telecommunications Congress 2012 Workshop "SDN and OpenFlow"*, March 2012, http://www.ieice.org/~wtc2012/Slides/Workshops/WS2-2/-WS2_2_4.pdf