# Requirements and Design of Flexible NFV Network Infrastructure Node Leveraging SDN/OpenFlow

Hitoshi Masutani, Yoshihiro Nakajima, Takeshi Kinoshita, Tomoya Hibi, Hirokazu Takahashi, Kazuaki Obana, Katsuhiro Shimano, Masaki Fukui
NTT Network Innovation Laboratories, 1-1, Hikarino-Oka, Yokosuka-Shi, Kanagawa-Ken, Japan
Email: masutani.hitoshi@lab.ntt.co.jp

***Abstract*—Carrier network service infrastructures are becoming increasingly complex since thousands of service-specific hardware-based network nodes are implemented to support a wide variety of network services. This has resulted in critically high increases in the maintenance costs to ensure service quality for network services and the deployment costs for new network services. Addressing these problems requires new ways to simplify and automate operations and infrastructures. Network Function Virtualization (NFV) and software-defined networking (SDN)/OpenFlow are attractive concepts that address these problems.**

**In this study we present the requirements and a basic design of a flexible and elastic network service infrastructure with NFV and SDN/OpenFlow. We also introduce a virtual BRAS (Broadband Remote Access Server) prototype using Intel DPDK as high performance throughputs.**

***Index Terms*—Network Function Virtualization, Software-Defined Networking, OpenFlow, virtual machine monitor, virtual switch, software switch**

## I. Introduction

Since network carrier infrastructures are composed of custom-designed hardware and special-purpose equipment and since network architectures are designed for specific network services, it is becoming difficult to flexibly deploy and manage network resources to quickly meet unpredictable network service demands.

Currently special-purpose equipment, such as custom ASIC-based routers and switches, is popular in the telecommunication industry, however it takes much more time to develop new network functionality, because the standardization to complete a specification of protocol or architecture is a long-term process, furthermore, it takes much more time to design and implement hardware-based network equipment for special-purpose. Therefore, a hardware-based approach is not suitable for rapidly meeting new network service demands.

Additionally, it is important to share resources among workloads in a network equipment to provide many different network services without deployment. However, since resource allocations have been finished before the day of release for network product, this makes the network infrastructure ineffective.

Furthermore, network operators have to continue to maintain a huge network infrastructure for a long time, with constant service quality provided only to comparatively few users. This long service lifetime leads to increases in the costs of network service deployment and operation.

Network Function Virtualization (NFV) [1-3] and Software-Defined Networking (SDN)/OpenFlow [4] are potentially new ways to address these issues.

NFV is a design concept to enable network services to be provided through a combination of software-based network functions and appliances, such as firewalls, network address translation (NAT), and load balancers, on top of commodity PC servers. Instead of using specialized hardware, NFV tries to exploit server virtualization techniques on commodity servers to provide flexible management in network infrastructure as cloud computing does. Virtualization technology manages computing resources to accommodate multi-tenancy and can reallocate resources to network functions after the initial deployment.

SDN/OpenFlow is based on the concept of decoupling the control plane and the data plane from the current vertically integrated network systems, for example, routers and switches. OpenFlow defines south-bound standard interfaces between these planes and switch packet match/processing behavior models so that developers can define their own packet processing rules using flow header matching and flow table actions without resorting to the modes of conventional network systems. It can provide a flexible multi-layer-aware network flow handling functionality so that tenant networks with network traffic isolation can be accommodated in the same network infrastructure. With standard interfaces, SDN/OpenFlow also enables network operators to program control-plane functions to provide workflow automation functionality and management capability for total networks. As a result, it may enable operation expenses to be decreased.

To achieve an NFV-enabled carrier network infrastructure with fine-grained controllability, we examined the idea of leveraging SDN/OpenFlow to support flexible and programmable underlay network infrastructures. For decoupling network functions from network nodes and achieving both elasticity and scalability in controlling and managing network service levels, the SDN/OpenFlow approach is more suitable than the conventional way for controlling network nodes like CLI. This is because

SDN/OpenFlow can provide much more explicit fine-grain flow control functionality and a unified standardized interface to control.

In this paper, we discuss the requirements for integrating NFV and SDN/OpenFlow, focusing on the architecture design of NFV-enabled network nodes for next generation telecommunication architectures.

The rest of this paper is organized as follows. Section II summarizes the requirements for enabling NFV-enabled network nodes to achieve performance similar to that of hardware-based network equipment. Section III presents the network function deployment patterns in NFV-enabled network nodes. Section IV explains the key software technologies for achieving high-performance packet processing. Section V describes the architecture design of an NFV-enabled network node. Section VI reports a virtual BRAS prototype we are currently developing. Finally, Section VII concludes the paper with a summary and a mention of future work.

## II. Requirements of Network Nodes For NFV

NFV infrastructures, especially NFV-enabled network nodes, must perform as well with commodity servers as special-purpose hardware-based network nodes do. In addition, the NFV-enabled network nodes should provide both management flexibility, such as dynamic service deployment, reconfiguration, and updating, and scalability control, such as dynamic scale-out and scale-in for traffic demands. The requirements for NFV-enabled network nodes and middleware to develop network functionality are summarized in the following subsections.

### *A. High-performance and low-latency packet processing*

Most commodity servers are designed for computation, not for packet processing. Therefore, special techniques are required to build a high-performance and low-latency packet processing mechanism on commodity servers with a commodity network interface card (NIC) for a major operating system. The network nodes should employ a low-overhead mechanism for packet processing. One reason for this is that, since the packet processing is performed through accessing the main memory of a server, the processing performance depends on how much memory copying can be omitted. Additionally, a defined time for packet processing is desirable for defining the service quality in designing network services.

### *B. Easy deployment*

A management system must dynamically install network function software to a dedicated commodity server without an operator's manual labor in a similar way to cloud computing. To achieve easy deployment, the software itself should decrease the dependency of OS kernel code, middleware, and hardware, or the system middleware of the commodity server should provide a hardware abstraction layer or high library compatibility. For example, a virtual machine environment can provide backward compatibility of hardware. Since network services are provided via network function software developed by many venders, the nodes should support a multi-vender software configuration. The software is installed to both a bare-metal commodity server to achieve high performance and a hypervisor-enabled commodity server to run many software programs for the network functions on the same commodity server.

### *C. Multi-tenancy and isolation*

In carrier networks, there are many types of network services. Multi-tenancy and isolation should simultaneously be provided with NFV-enabled network nodes. Especially, performance isolation, control isolation, and failure isolation should be guaranteed. However, since commodity servers do not strongly support these isolations, software techniques or other ways are required to meet the network service SLA. The latest Linux kernel supports the "isolcpus" option that allocates cpu resources exclusively a process. With this option, hypervisor-enabled virtualization is an appropriate platform technology to ensure performance from point of view in isolated tenants, full virtualization and OS version independency. Security isolation may also be required in providing multi-vendor network functions on a commodity server to avoid malicious interference.

### *D. Fine-grained control of network functions*

Since special-purpose hardware network equipment has too many rich network functionalities dedicated to hardware resource, the granularity of deployment is often large in the cases of capacity enhancement or failure maintenance. This will increase the capital expenditure (CAPEX) for network nodes when adding new network nodes to the service platform. Fine-grained deployment, as well as control and management of network functions, are required in the context of NFV-enabled network nodes. OpenFlow can isolate in-coming traffic into individual service-specific traffic packets. Incorporating OpenFlow concepts into underlying network service infrastructures, especially in internal virtual networks, is quite suitable for achieving flexible service-level control and management.

### *E. Control interface for external systems*

To achieve co-operative control among hardware resources, middleware, and software from the network management systems, NFV-enabled network nodes should provide a multi-type interface (e.g., REST API) to ensure dynamic scalability control of network services.

### *F. Monitoring*

The management system must have a function for monitoring NFV-enabled network nodes as well as conventional network nodes. Since commodity servers are more fragile than special-purpose network nodes, a function for proactive node management to handle system failures should be supported. Achieving such management requires many means for handling monitoring information, including the hardware, OS, middleware, and software for network functions.

### *G. Fault tolerance*

Commodity servers are more fragile than special-purpose network nodes; this increases the importance of having a fault tolerance mechanism. In cloud computing, redundant

technique, such as load balancing, is often employed to ensure availability and scalability in the event of failures. Cloud computing methods should be incorporated into NFV-enabled network services to ensure fault tolerance is achieved.

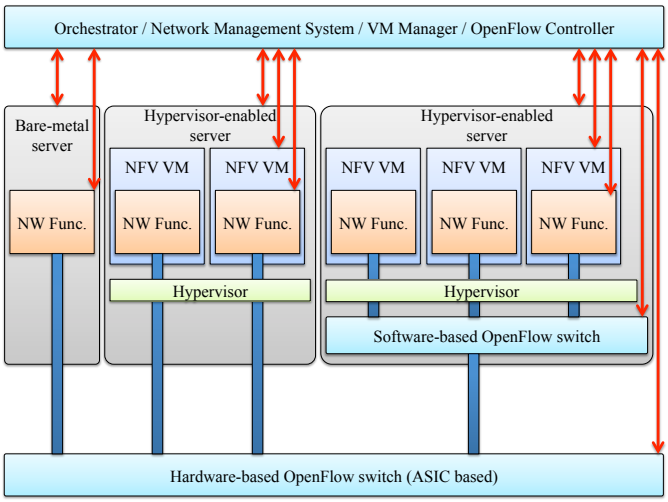


**Fig. 1** Three options of NFV deployment from the node point of view

## III. Deployment Patterns for Network Functions on NFV-Enabled Network Nodes

There are two options, i.e., bare metal and virtual machine (VM), available for deploying network functions on a commodity server to achieve performance or resource usage like that of cloud computing. To achieve service-level controllability, there are another two options (i.e., inside and outside) for deploying OpenFlow switches to apply service isolation to NFV-enabled network nodes. Mixing these options for deploying network functions and OpenFlow switches produces three deployment patterns shown in Fig.1.

### *A. Network functions on bare-metal environment*

The left side of Fig.1 shows the first deployment pattern of network functions: on bare metal. Performance requirements may request a network function as a stand-alone application on a commodity server. There may be other reasons for such requests, e.g., to address security considerations or achieve easy management. In this pattern, network functions can be controlled and managed at the service level if an OpenFlow switch is applied into the underlying network infrastructure.

### *B. Network functions on virtualized environment with directly attached NIC*

The middle of Fig.1 shows the second deployment pattern of network functions: on VM. The management policy in the cases where a network function or network service is mapped to physical network ports may be suitable. In this pattern, the physical port bandwidth can be entirely allocated to a dedicated network function. On the other hand, performance isolation is required to avoid interference among network functions on the same commodity server. An OpenFlow switch should be applied into the underlying network infrastructure to achieve service-level management.

### *C. Network functions on virtualized environment with virtual OpenFlow switch*

The right side of Fig.1 shows the third deployment pattern of network functions: on VM with a virtual OpenFlow switch. This pattern may be suitable for increasing the efficiency of server resource usage as well as for dynamically deploying different network functions on the same commodity server. In this pattern, both performance isolation and traffic isolation at the service level are very important functions to avoid service quality degradation.

In all three patterns, it is also possible to apply traditional network switches (L2 switches, L3 switches) to the underlay infrastructure in place of OpenFlow switches. Deciding which pattern is the most appropriate for providing network services depends for the most part on service operation requirements and management policy requirements.

### *D. Management API*

From a management point of view, there are four APIs network management systems can use to control, manage, and monitor different types of instances. These are for physical servers, VM, network functions, or OpenFlow switches (Fig.1). An OpenFlow switch is controlled via an OpenFlow controller that can support the OpenFlow protocol and/or the OF-Config protocol [5]. A VM is deployed, controlled, and managed using a VM manager such as libvirt [6]. Since network functions need service-oriented APIs to be controlled directly or indirectly, each network service has a specific operation policy and SLA.

### *E. Network service scaling capability*

An OpenFlow switch basically recognizes fifteen packet header tuples in OpenFlow version 1.3 to forward incoming packets to the desirable direction by searching the matched flow entry in flow tables. Consequently, the pre-configured flow entries are very important in suppressing the number of incoming packet actions between the OpenFlow switch and OpenFlow controller, a high number of which degrades packet forwarding throughput.

In carrier networks, very complicated session management is normally used to provide network services, which means that OpenFlow is not suitable for carrier network services because a huge number of incoming packet actions along with session state changes can degrade performance. If OpenFlow is used, it is necessary to execute very complicated pre-configuring of flow rules in advance every time a session state changes or network functions are deployed. On the other hand, OpenFlow is a good solution for stateless applications.

For handling unexpected resource demands, dynamic scale-out and scale-in are important for NFV capabilities. If scaling is to be executed for an already active network service, deep packet inspection and a policy-based forwarding is required for separating incoming converged traffic into specific traffic groups. In this case, OpenFlow is not an appropriate solution because of its limited packet analysis capability. For elastic network services, each network function category should have the following features:

- Network-service-oriented deep packet inspection
- Policy-based distribution of workloads

This kind of network function, which we call Network Service-Oriented Distributor (NSOD), is applied into the three deployment patterns shown in Fig.1. More scalable deployment pattern are depicted in Fig.2.

Figure 2 shows that there are two types of deployment patterns, direct attached model in which NSOD is directly attached to a physical NIC, and internal OpenFlow model in which all instances connect to each other via an OpenFlow switch. The direct attached model is suitable for an operation policy where the port bandwidth is wholly allocated to a network service, whereas only the internal flow management is needed to interconnect NFV VMs (or instances). On the other hand, the internal OpenFlow model provides OpenFlow-based controllability with all NFV VMs. This enables multiple different types of network services to share the same physical port, thus achieving effective resource usage. In this two models, Openflow is used to basic flow forwarding at a coarsegranularity with suppressing Packet_in actions.

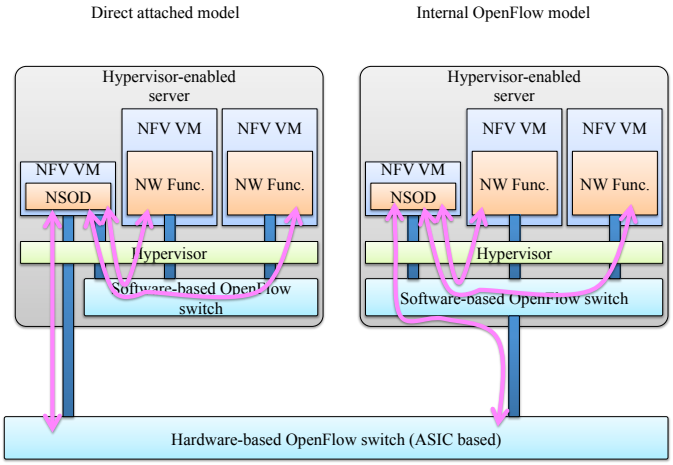


**Fig. 2** Two options of NFV deployment using NSOD

## IV. MOTIVATION AND KEY TECHNOLOGIES

For the cases shown in Fig.1 and Fig.2, the software approach we take for both network functions and OpenFlow switches is suitable for designing and implementing NFV-enabled network nodes. However, to implement network functions as software, there is performance issue because commodity servers are not designed for network packet processing. Well-known performance issues [7-9] arise that are not a factor in hardware-based network nodes such as switches and routers, even with the use of fine-tuned OS.

Server virtualization may also substantially degrade performance because emulating the guest OS makes the I/O workload a possible bottleneck point despite the use of hardware-assisted virtualization technologies, e.g., Intel VT-d.

Furthermore, if network operators try to create new network services as soon as possible, or if they adapt existing network functions to other OS platforms to attain new capabilities or for other reasons, the dependency on other software codes can be a main factor disturbing the quick development of network functions.

### *A. Bypassing OS networking*

There are two kinds of performance issues related to software and hardware for achieving high-performance packet processing.

Packet handling in the OS kernel between outer and internal networks had until recently been based on an interrupt-driven model. However, we have already been able to use a new OS kernel that is based on a hybrid-packet processing model, such as the Linux NAPI (New API) network driver framework [10]. This model adopts the polling concept to the interrupt-driven model for mitigating the number of interrupts, which is important because interrupts from network interface devices degrade packet throughput [11]. Therefore, to maximize network throughput on a physical server, the number of interrupts occurring when sending and receiving packets should be suppressed as much as possible.

The second issue, a hardware-related one that makes the packet processing performance inefficient, is the amount of copying done among memory areas. For the latest CPUs (e.g., Intel Xeon processor), the cycle time is several hundred times shorter than the time needed to access a memory like DRAM. From the CPU point of view, there is a huge interval difference between CPU instruction and memory access, which means that the CPU has to wait for a long time to execute a next instruction if has to refer to the memory area. The latest major OS kernel copies received packets to several memory areas for passing their packets to applications. Eventually, this causes network I/O performance degradation. Packet processing can be done more efficiently by reducing the amount of memory copying and using a faster memory (such as a cache memory) in place of a DRAM memory.

The combination of a polling model and bypassing OS networking is one of the breakthroughs that have been made to address the aforementioned network performance issues. One of such model is the Intel Data Plane Development Kit (DPDK) [12], a packet handling framework that allows application developers to bypass OS networking without developing an OS kernel code. Briefly, Intel DPDK enables direct packet copying to userspace network applications to minimize the number of packets copied in the OS kernel without hardware interrupts, while it allocates CPU resources exclusively to packet handling workloads. Hence, this packet handling framework is one of the key technologies for achieving high-performance packet forwarding applications, such as OpenFlow switch and network function applications.

### *B. Paravirtualized I/O acceleration*

Server virtualization programs such as the Linux kvm hypervisor [13] make it possible to establish a multi-tenant infrastructure by isolating code dependency from the host OS version. However, to obtain good performance as well as flexibility with them, it is necessary to pay attention to the hardware emulation overheads on VM.

Virtio [14] is a paravirtualized I/O framework not only for networking, but also for consoles, block devices, and file systems. This framework consists of two main components, which are emulated virtual PCI devices defined by the virtio

specification and the virtio driver supported in the guest OS. This I/O common framework replaces the lazy I/O device emulations between host OS and guest OS with a simple ring-based queue operation to reduce the amount of packet copying. Virtio has already been used as a main I/O platform in the Linux kvm hypervisor for high speed networking.

Intel DPDK has already been used to support the virtio poll mode driver (PMD) [12] on a VM based on a Linux kvm hypervisor. This has made it possible to apply the features of bypassing OS networking and those of the common virtio framework to network applications on VM.

For deploying NFV-enabled network nodes, especially on VM, Intel DPDK is also one of the fastest packet handling frameworks for building multiple high performance network applications on the same physical server because of its support of the virtio PMD.

Another benefit of Intel DPDK is that it enhances flexibility by decreasing dependency on a kernel-based networking framework, which enables network operators to maintain extended support of network services while reducing software maintenance costs.

### C. *Investigating dependency on software codes*

Intel DPDK provides network function developers with both a high-performance packet handling framework and programmability in userspace. As long as Intel DPDK continues to be updated and supported for major OSs, network function developers can focus on implementing userspace network applications using DPDK API. This will free them from the need to depend on OS kernel code updates. However, this kind of framework forces them to be dedicated to a specific design model that is almost different from that of other similar frameworks [15]. Such frameworks have already been developed or proposed, and support different NICs or platforms. Therefore, it would be better for network functions to be divided into network service logic and packet handling with abstraction.

## V. Architecture Design of NFV-Enabled Network Node

Figure 3 compares our NFV design approach with a conventional design approach with respect to building network functions on commodity servers.

The conventional design approach cited is based on a socket interface or a kernel interface to implement new network functions in major OSs on commodity servers. A socket interface is an abstraction layer that makes it possible to easily leverage the basic network stack provided by the OS kernel, enabling effective development without the need to depend on a hardware-specific code. On the other hand, OS kernel modification is suitable for adding new features to the low layer of a kernel network stack to achieve improved performance. As described before, this approach is particularly subject to performance or dependency issues.

To obtain good performance as well as multi-tenancy while decreasing dependency on other software codes, we designed the NFV-enabled network node shown in Fig.3. The packet handling layer is particularly important for achieving high performance packet processing and reducing dependency on the OS kernel code to reduce maintenance costs. Our NFV design approach provides abstraction between the packet handling framework and network functions because it should force developers to be dedicated to a specific design model. Network functions are divided into two parts, i.e., network application logic and standardized network protocol stacks, such as IP, UDP, and TCP. A basic network protocol stack should be provided as a common framework because the most network applications always leverage it to communicate with other network applications. In our approach, the software-based OpenFlow switch is also based on userspace implementation to enable flexible extension. Since both the network functions on VM (NFV VM) and the software-based OpenFlow switch work in the same memory area (e.g., userspace), acceleration of the virtual link between NFV VM and OpenFlow switch is also important.

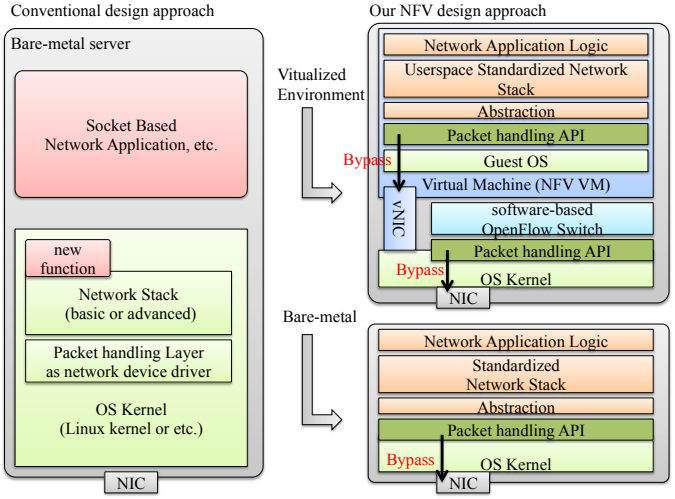


**Fig. 3** Our NFV design approach in comparison with that of conventional network applications.

## VI. Prototype of Virtual BRAS

Figure 4 shows the virtual BRAS prototype we are currently developing as a first target. We integrated Intel DPDK into our NFV-enabled network node as a high-performance packet handling framework on the Linux kvm hypervisor. To achieve the flexible on-demand deployment needed to compensate for a lack of processing resources or the occurrence of failovers, we adopted the virtual OpenFlow switch model shown in Fig.1. The internal underlying network is currently Open vSwitch (OVS) [16] as a temporary software solution. However, OVS is subject to a performance bottleneck because of traditional OS kernel or socket-based implementation. As depicted in Fig.2, we set NSOD as a mediator between service client and network service to eliminate client configuration dependency from NFV VM. In our prototyping, NSOD provides simple port-based load balancing function for forwarding all packets from clients accommodated in a NIC port to a NFV VM using vlan-id matching. When OVS receives a client packet from a NIC port, it is attached a vlan-id as vlan tag by OVS and is forwarded to NSOD. NSOD

looks up the vlan id using the internal vlan-id table and does matching the vlan-id to a NFV VM. If NSOD can find the NFV VM, NSOD switches the vlan id to a new vlan-id, and forward the packet to the NFV VM via OVS.

Virtual BRAS accommodates two service types: Internet connection services and SIP services (e.g., IP telephone services). Internet connection services provide Internet connectivity to forward PPPoE client packets to LNS (L2TP Network Server) on the tunnel provided by L2TP. SIP services provide SIP registering and resolver of SIP URI. We adopted Asterisk [17] as a sample SIP server for SIP function validation purposes.

Incoming service traffic involves a two-step isolation process. At the first step, the internal OpenFlow switch isolates traffic into two separate route domains at service level using header parameters, including input port, ether-type, and/or port number. At the next step, NSOD distributes service traffic to each NFV VM in accordance with a vlan id matched to a NIC port as mentioned above. This hierarchical isolation can be adapted to any type of network service.

The OpenFlow switch flow table is configured two times. The initial configuration is carried out before VM deployment and the second is carried out every time an individual SIP service session is established. This means that updating the flow table is a possible performance bottleneck.

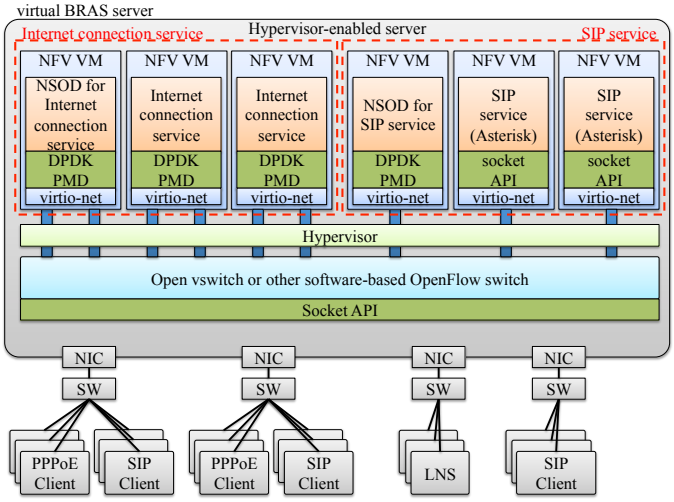


**Fig. 4** Virtual BRAS network prototype. LNS: L2TP Network Server.

## VII. Conclusion

High-level requirements have already been published in an official Network Function Virtualization (NFV) document [3]. However, the requirements for implementing NFV-enabled network nodes are almost completely dependent on the fact that operation and management policies differ among network operators and regions. In this paper, we discussed the requirements for NFV-enabled network nodes from a network operator's perspective. We also proposed three deployment patterns for an NFV VM with an OpenFlow switch and two deployment patterns using NSOD. Finally, we presented a virtual Broadband Remote Access Server (BRAS) prototype we developed under an Open vSwitch (OVS) network using Intel DPDK as state-of-the-art technology.

For the next step, we will evaluate virtual BRAS from several points of view excepting the OVS bottleneck. Then, we will replace OVS with our software-based OpenFlow switch to attempt to achieve overall enhancement of performance and flexibility. Since both network functions and software-based OpenFlow switches are implemented in userspace, we may have to adopt a high-speed virtual link to achieve improved performance. Additionally, we would like to incorporate an abstraction layer around the packet handling framework, since doing so would be extremely advantageous for providing developers (i.e., network operators) with alternatives to packet handling schemes dependent on vendor NICs. However, it will be necessary to carefully study the abstraction layer design because an abstraction API can have considerable impact on module design and software codes.

## Acknowledgment

Our research output described in this paper is a part of outcome of the O3 Project [18] funded by Ministry of Internal Affairs and Communications in Japan. We would like to express special thanks for conducting research.

## References

[1] NFV White paper, http://portal.etsi.org/NFV/NFV_White_Paper.pdf
[2] Network Functions Virtualisation (NFV); Architectural Framework, online http://www.etsi.org/deliver/etsi_gs/NFV/001_099/002/01.01.01_60/gs_NFV002v010101p.pdf
[3] Network Functions Virtualisation (NFV); Virtualisation Requirements, online http://www.etsi.org/deliver/etsi_gs/NFV/001_099/004/01.01.01_60/gs_NFV004v010101p.pdf
[4] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: enabling innovation in campus networks," SIGCOMM CCR, 38(2):69–74, 2008.
[5] OF-config 1.2, online https://www.opennetworking.org/sdn-resources/onf-specifications/openflow-config
[6] libvirt, online http://libvirt.org/index.html
[7] E. Guillen, A. M. Sossa, E. P. Estupiñán, "Performance Analysis over Software Router vs. Hardware Router: A Practical Approach," Proceedings of the World Congress on Engineering and Computer Science 2012 Vol II, WCECS 2012, October 24-26, 2012, San Francisco, USA
[8] Ssang-Hee Seo and In-Yeup Kong, "A Performance Analysis Model of PC-based Software Router Supporting IPv6-IPv4 Translation for Residential Gateway," Journal of Information Processing Systems, vol. 1, no. 1, pp. 62~69, 2005.
[9] A. Bianco, R. Birke, L. Giraudo, and N. Li, "Multistage Software Routers in a Virtual Environment," Proc. IEEE Globecom 2010, Miami, FL, December 2005, pp. 1-5.
[10] Linux NAPI, online http://www.linuxfoundation.org/collaborate/workgroups/networking/napi
[11] K. Salah , A. Qahtan, "Implementation and experimental performance evaluation of a hybrid interrupt-handling scheme," Computer Communications, v.32 n.1, p.179-188, January, 2009
[12] Intel DPDK, online http://www.intel.com/go/dpdk
[13] Kernel-based Virtual Machine, online http://www.linux-kvm.org/page/Documents
[14] Rusty Russell, "virtio: towards a de-facto standard for virtual I/O devices," ACM SIGOPS Operating Systems Review, v.42 n.5, p.95-103, July 2008
[15] Luigi Rizzo, "Netmap: a novel framework for fast packet I/O," Proceedings of the 2012 USENIX conference on Annual Technical Conference, p.9-9, June 13-15, 2012, Boston, MA
[16] B. Pfaff, J. Pettit, T. Koponen, K. Amidon, M. Casado, and S. Shenker, "Extending networking into the virtualization layer," in HotNets, 2009.
[17] Asterisk, online http://www.asterisk.org
[18] O3 project, online http://www.ntt.co.jp/news2013/1309e/130917a.html