IEEE.org IEEE Xplore IEEE SA IEEE Spectrum More Sites Subscribe Donate Cart Create Account Personal Sign In

Browse My Settings Help Institutional Sign In

Institutional Sign In

All

ADVANCED SEARCH

Conferences > 2012 Second Symposium on Netw...

vNFC: A Virtual Networking Function Container for SDN-Enabled Virtual Networks

Publisher: IEEE

Cite This PDF

Ryota Kawashima All Authors

4 Cites in Papers

3 Cites in Patents

1137 Full Text Views

Alerts

Manage Content Alerts

Add to Citation Alerts

Abstract

Document Sections

I. Introduction

II. SDN-ENABLED Virtual Datacenter Networks

III. vNFC Architecture

IV. Examples

V. Evaluation

Show Full Outline

Authors

Figures

References

Citations

Keywords

Metrics

More Like This

Footnotes

Downl PDF

Abstract:
Software-defined networks (SDN) has gradually been deployed on commercial networks such as datacenter networks. Current SDN is based on OpenFlow technology that is a set ... **View more**

Metadata

Abstract:
Software-defined networks (SDN) has gradually been deployed on commercial networks such as datacenter networks. Current SDN is based on OpenFlow technology that is a set ofnetwork flow control API for switch devices. For instance, network reachability between end-hosts (or virtual machines), packet filtering mechanisms, and status management of switches are enabled by the API. In practice, however, current OpenFlow-based SDN has following problems: no application-layer protocol support and switch-oriented flow control. Since OpenFlow targets L2-L4 flow handling, users have to arrange additional mechanism for upper-layer flow control. Furthermore, executing a lot of flow matching on a single switch (or virtual switch) can cause difficulty in network trace and overall performance degradation.This paper proposes a virtual networking function container (vNFC) that is a set of software implemented networking functions for VM-to-VM communications, and it is located between a virtual machine and a virtual network device of the host machine. vNFC enables not only lower-layer functions OpenFlow providing, but also upper-layer functions like application firewall in the same manner. That is, vNFC is a virtual machine dedicated flow handling function set. In addition, OpenFlow-compatible vNFC configuration protocol named OpenNF and vNFC controller are also presented. OpenNF provides communication path between each networking function and the controller for configuration and decision making.In this paper, architectural design and implementation of vNFC are presented, and also performance evaluation of using vNFC. The evaluation result shows that a lightweight networking function does not impact on the performance, but a function that frequently communicates with the controller incurs millisecond order cost per frame transmission.

Published in: 2012 Second Symposium on Network Cloud Computing and Applications

ng [MathJax]/extensions/MathMenu.js

Date of Conference: 03-04 December 2012

DOI: 10.1109/NCCA.2012.18

Date Added to IEEE *Xplore*: 07 March 2013

Publisher: IEEE

ISBN Information:

Conference Location: London, UK

Contents

I. Introduction

1

Networking function refers to a set of computation procedures composing a single service, such as encapsulation, encryption, or filtering.

Sign in to Continue Reading

In these days, software defined networks (SDN) or OpenFlow[1] has gained attention both in academic and business fields. SDN/OpenFlow enables dynamic configuration of an entire network from a software-implemented control brain in an open approach, and this flexibility is beneficial for cloud datacenter networks consisting of many switches, computing nodes. and users' virtual machines.

Authors

Figures

References

Citations

Keywords

Metrics

Footnotes

More Like This

Performance evaluation of shared library supporting multi-platform for overlay network protocol
2020 IEEE 9th Global Conference on Consumer Electronics (GCCE)
Published: 2020

Performance Evaluation of PTP in Switched Ethernet Networks
2023 14th International Conference on Computing Communication and Networking Technologies (ICCCNT)
Published: 2023

Show More

Loading [MathJax]/extensions/MathMenu.js

2012 Second Symposium on Network Cloud Computing and Applications

vNFC: A Virtual Networking Function Container for SDN-enabled Virtual Networks

Ryota Kawashima

ACCESS CO., LTD.
Tokyo, Japan
Email: kawa1983@ieee.org

***Abstract*—Software-defined networks (SDN) has gradually been deployed on commercial networks such as datacenter networks. Current SDN is based on OpenFlow technology that is a set of network flow control API for switch devices. For instance, network reachability between end-hosts (or virtual machines), packet filtering mechanisms, and status management of switches are enabled by the API. In practice, however, current OpenFlow-based SDN has following problems: no application-layer protocol support and switch-oriented flow control. Since OpenFlow targets L2-L4 flow handling, users have to arrange additional mechanism for upper-layer flow control. Furthermore, executing a lot of flow matching on a single switch (or virtual switch) can cause difficulty in network trace and overall performance degradation.**

This paper proposes a virtual networking function container (vNFC) that is a set of software implemented networking functions[1] for VM-to-VM communications, and it is located between a virtual machine and a virtual network device of the host machine. vNFC enables not only lower-layer functions OpenFlow providing, but also upper-layer functions like application firewall in the same manner. That is, vNFC is a virtual machine dedicated flow handling function set. In addition, OpenFlow-compatible vNFC configuration protocol named OpenNF and vNFC controller are also presented. OpenNF provides communication path between each networking function and the controller for configuration and decision making.

In this paper, architectural design and implementation of vNFC are presented, and also performance evaluation of using vNFC. The evaluation result shows that a lightweight networking function does not impact on the performance, but a function that frequently communicates with the controller incurs millisecond order cost per frame transmission.

I. Introduction

In these days, software defined networks (SDN) or OpenFlow[1] has gained attention both in academic and business fields. SDN/OpenFlow enables dynamic configuration of an entire network from a software-implemented control brain in an open approach, and this flexibility is beneficial for cloud datacenter networks consisting of many switches, computing nodes, and users' virtual machines.

OpenFlow has been thought as a essential API of controlling network flows because it can manipulate each frame finely such that deciding the forwarding port on-the-fly based on dynamically set flow rules, altering destination address, or pushing/popping VLAN or MPLS tag. However, considering large-scaled cloud networks providing IaaS/PaaS/SaaS or multi-tenant datacenter networks, current SDN/OpenFlow lacks service-centric control mechanisms. For example, user dedicated QoS/TE services, security services, and accounting services should be provided as unified open services. Besides, SDN/OpenFlow concentrates its functionalities to OpenFlow switches and controllers, which can cause complex flow management for each switch and extra communication cost between switches and the controller.

We have studied about transparent manipulation of communication flows at socket-layer within application process image[2]. This time, the notion of the transparent manipulation approach is applied to virtual networks on datacenter networks as a virtual networking function container (vNFC), and also OpenFlow-compatible vNFC control API (OpenNF) is proposed. vNFC is a set of networking functions located between a virtual machine and a virtual networking interface on the host machine, and each networking function is transparently applied to VM transmitting/receiving frames. If vNFC contains encryption and logging functions, each transmitting frame from the VM is encrypted, and then recorded the src/dest addressed and the frame size before transmitting to the physical network. Each vNFC's functionality can be dynamically configured by a controller program with OpenNF API that is put on OpenFlow vendor messages.

In this paper, architectural design and implementation of vNFC are presented, and also performance evaluation of using vNFC. The evaluation result shows that a lightweight networking function does not impact on the performance, but a function that frequently communicates with the controller incurs millisecond order cost per frame transmission.

The rest of the paper is organized as follows. Section II gives basic description of virtual networks with SDN/OpenFlow. Section III describes the architecture and implementation details of vNFC and OpenNF, and section IV illustrates some example networking functions and usage scenarios. The performance overhead of the proposed system is evaluated in section V. In section VI, the proposed approach is compared with related studies. Section VII concludes this study and clarifies future studies.

II. SDN-enabled Virtual Datacenter Networks

Current datacenter networks hold many private user/tenant networks that are constructed as virtual networks. In order

[1]Networking function refers to a set of computation procedures composing a single service, such as encapsulation, encryption, or filtering.

978-0-7695-4943-9/12 $26.00 © 2012 IEEE
DOI 10.1109/NCCA.2012.18

IEEE computer society

Exhibit 1043
Microsoft Corporation v. Edge Networking Systems, LLC
003

to compose flexible virtual networks regardless of the architecture of physical network substrates, two types approaches, *hop-by-hop* and *edge-overlay*, are used. With the hop-by-hop type, a controller directly controls each switch by OpenFlow in order to determine next hop of the tenant flows. On the other hand, there are tunnel end points (TEP) on both end-(physical)-host with the edge-overlay type for carrying tenant flows on L2 over L3 tunneling protocol, such as GRE[4], VxLAN[5] and STT[11]. Since the later approach does not require OpenFlow-enabled network appliances, many datacenters adopt the approach at this time.

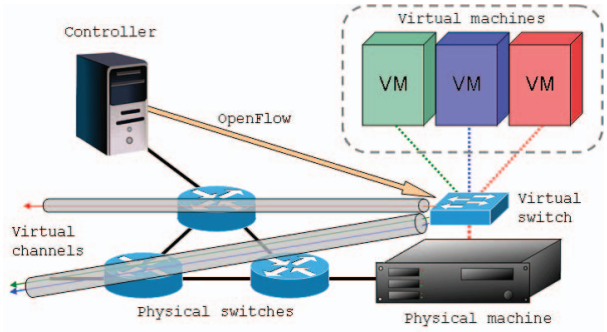

Fig. 1. A typical SDN/OpenFlow-enabled virtual network (edge-overlay)

Figure 1 depicts a typical architecture of virtual datacenter network with edge-overlay approach. Three users' VMs are running on a same physical host, and each VM is connected to a same OpenFlow-enabled virtual switch like Open vSwitch[3]. Data frames come from VMs are tagged with some ID like VLAN or VNI based on the tunneling protocol before transmitting. Therefore, each tenant traffic is logically separated by the ID. The tagged frame is transmitted to its destination based on flow rules. The flow rules are decided by an OpenFlow controller at each flow start (*proactive mode*) or previously set to the virtual switch (*inactive mode*).

In practice, there are remained several challenges to deploy the illustrated model into practical datacenter networks. For example, many virtual switches does not handle application-layer protocols, performance of the virtual switches decrease if many VMs are attached or many flow rules are set, current OpenFlow (1.3) does not support TEP configuration, and edge-overlay approach can cause inefficient packet fragmentation without MTU adjustment on the physical/virtual machines.

III. vNFC Architecture

In this section, the architectural design and implementation of vNFC and OpenNF are described. vNFC is designed to introduce VM instance dedicated L2–L7 networking functionality onto each virtual network flow.

A. Model

Figure 2 shows an architectural model of vNFC on the SDN-enabled virtual networks. vNFC is located between a virtual machine and a corresponding virtual network device

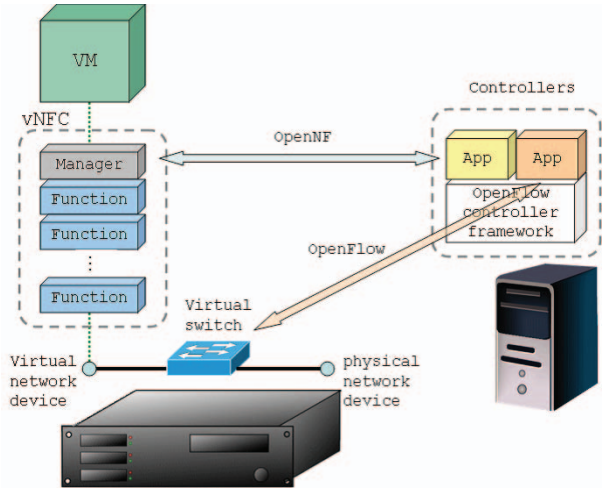

Fig. 2. An architectural model of vNFC with OpenNF

transparently to them. That is, the VM itself thinks that it directly interacts with the device, but actually vNFC transparently intermediates their interaction.

vNFC consists of arbitrary networking functions, a networking function manager, networking function libraries, OpenNF client, and an OpenNF controller. Networking functions can be thought as VM-dedicated networking appliance such as firewall, traffic shaper, monitoring, or encryption. The networking function manager handles each networking function (loading, unloading, and configuration) under the direction of the OpenNF controller.

Transmitted frames from the VM are passed to vNFC and after the processing of networking functions. These frames are delivered to the virtual switch via the virtual network device, and the virtual switch forwards them to a virtual port based on the flow rules.

The OpenNF controller is built on an OpenFlow controller framework, such as NOX[6], Floodlight[7], and Trema[8] in order to facilitate protocol implementation. The detail of OpenNF is written in later.

The heart of the design of vNFC is off-loading VM-dedicated functions from the virtual switch and realizing application-layer function support. This off-loading mitigates the overload of the virtual switches and simplifies their flow rules and implementation.

B. OpenNF

OpenNF is used to manage networking functions of vNFC, and deliver specific data between vNFC and the OpenNF controller. Since OpenFlow has been regarded as an essential API for controlling network flows, OpenNF is designed to take advantage of its protocol design and implementation. Concretely, every OpenNF messages are conveyed as OpenFlow's vendor/experimenter messages (`OFPT_VENDOR/OFPT_EXPERIMENTER`), which enables the OpenNF controller to be implemented on the OpenFlow

Exhibit 1043
Microsoft Corporation v. Edge Networking Systems, LLC
004

controller frameworks and collaborate with OpenFlow controller applications.

TABLE I
OPENNF MESSAGE TYPES

Message type	Description
NFCT_UPGRADE_REQUEST	Request to use OpenNF
NFCT_UPGRADE_REPLY	Permit to use OpenNF
NFCT_FUNCTION_LIST_REQUEST	Request using functions
NFCT_FUNCTION_LIST_REPLY	Currently using functions
NFCT_LOAD_FUNCTION	Register specified function
NFCT_UNLOAD_FUNCTION	Unregister specified function
NFCT_UPDATE_FUNCTION	Update function's setting
NFCT_CUSTOM	Function-dedicated message

Table I lists OpenNF message types and their descriptions. `NFCT_UPGRADE_REQUEST` and `NFCT_UPGRADE_REPLY` messages are used to initiate OpenNF communication, and after the exchange of upgrade messages, the OpenNF controller sets up the functionality of vNFC using `NFCT_LOAD_FUNCTION` or `NFCT_UPDATE_FUNCTION` message. Additionally, `NFCT_CUSTOM` message is provided for networking function developers to convey function specific messages.

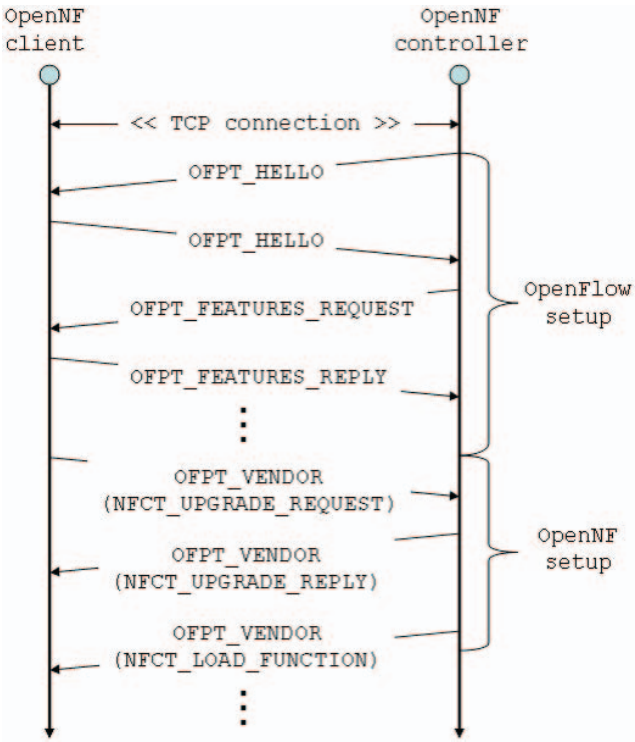

Fig. 3. A sequence of OpenNF communication

Figure 3 gives a message sequence of OpenNF communication. After establishment of the TCP connection, the OpenNF client (vNFC) and the OpenFlow controller set up OpenFlow connection starting from `OFPT_HELLO` messages, and then vNFC is recognized as a common switch client by the controller. At this point, vNFC sends a custom message (`NFCT_UPGRADE_REQUEST`) as a `OFPT_VENDOR/EXPERIMENTER` message in order to start OpenNF communication. Then the OpenNF controller can know available networking function list and capabilities.

```
struct nfc_header {
    struct ofp_header header; /* OpenFlow header */
    uint32_ experimenter;     /* OpenNF ID */
    uint32_ nfc_type;         /* OpenNF msg type */
    uint8_ body[0];           /* OpenNF msg body */
};

struct nfc_upgrade {
    struct nfc_header nfc;
    uint8_t version;
    uint8_t pad[5];
    uint16_t functions_len;
    struct nfc_function functions[0];
};

struct nfc_function {
    uint64_t id;
    uint8_t name[NFCFT_FUNCTION_NAME_LEN];
    uint64_t capability;
};

struct nfc_load_function {
    struct nfc_header nfc;
    uint16_t index;
    uint8_t pad[6];
    struct nfc_function function;
};

struct nfc_unload_function {
    struct nfc_header nfc;
    uint64_t id;
};
```

Fig. 4. A definition of OpenNF message structures

Structures of OpenNF messages are shown in figure 4. A `nfc_header` structure corresponds to `ofp_vendor/experimenter` structure of OpenFlow. The `nfc_type` member specifies OpenNF message type (NFCT_*), and the content of `body` varies from message types.

C. Implementation

This section describes implementation details of vNFC that transparently resides between the virtual machine and the virtual network device. As shown in figure 5, vNFC hooks particular system calls invoked by a virtual machine monitor like QEMU[9][10] at the *syscall hook* layer. Modern operating

Exhibit 1043
Microsoft Corporation v. Edge Networking Systems, LLC
005

systems provide hooking mechanisms for their system calls, for example, `LD_PRELOAD` environment variable is available in Linux. The *interface* layer provides wrapper functions of system calls for portability of vNFC's core functionality. To avoid recursive system call hooking, the *interface* layer utilizes `syscall()` and `dl*()` system calls.

First, vNFC tracks QEMU's accesses to TUN/TAP device ("/dev/net/tun") to distinguish network operations from other operations. Simultaneously, *NF Manager* invokes *OpenNF client* thread and the client immediately starts communication with the OpenNF controller to obtain vNFC configurations. *NF Manager* sets up each networking function based on the configurations and arranges a set of function pointers to the underlying networking function.

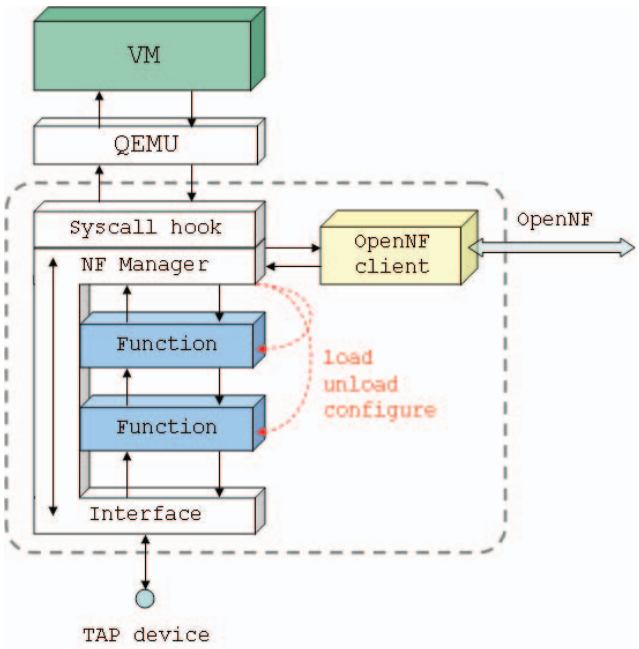

Fig. 5. An implementation architecture of vNFC

A networking function is implemented as a dynamically linkable shared library to realize runtime composition of networking functions according to the configuration. Figure 6 represents a template pseudo code of a networking function library. The presented `nf_*` functions are exported (opened), and *NF Manager* or upper-layer libraries can call them. Note that each library function does not invoke corresponding system call (e.g. `writev()` is not called directly within `nf_writev()`), and instead the function pointers to the underlying library are invoked for flexible composition of library sequence.

IV. Examples

This section gives three example scenarios of vNFC usage, VM-dedicated firewall, traffic shaping, and fragment prevention caused by tunnel encapsulation.

```
static functions *next;

void nf_init( functions *f ) {
    /* Initialize library */
    nf_config(c);
    nf_next(f);
}

void nf_next( functions *f ) {
    next = f; /* Set function pointers to */
}             /* the underlying library */

void nf_config( configuration *c ) {
    /* Update configurations */
}

ssize_t nf_writev(int fd, const struct iovec *iov, int count)
{
    /* NF processing ... */

    return next->writev(fd, iov, count);
}

ssize_t nf_read(int fd, void *buf, size_t count) {
    ssize_t n = next->read(fd, buf, count);

    /* NF processing ... */
    return n;
}
...
```

Fig. 6. A template code of a networking function library

A. VM-dedicated firewall

Simple packet filtering can be realized on virtual switches and filtering rules can be set using OpenFlow. However, such a mechanism causes high performance overhead and operational cost (e.g. if 100 VMs are attached to a single switch and 10 filtering rules are required for each VM, $(1000+\alpha^2)$ times of flow matching is required for each incoming/outgoing frame).

A firewall implemented as vNFC networking function allows dedicated filtering rules for each virtual machine instance. In the above example, only $(10+\alpha)$ rules are required for each frame. In addition, vNFC enables also application-layer firewall, such that `HTTP GET` requests are allowed but `PUT/POST` requests have to be denied. This VM-dedicated firewall is beneficial for datacenter providers because they can provide custom firewalls by themselves for their users.

[2]The virtual switch may insert additional auto-learned rules

Exhibit 1043
Microsoft Corporation v. Edge Networking Systems, LLC
006

B. Traffic Shaping

Network traffic can be controlled at user-space by introducing appropriate delays and a daemon that coordinates global traffic rate. Trickle[12] has proved this approach is effective for applications' traffics. Considering a virtual machine is an application process of the host machine, Trickle's approach can be applied to virtual network traffic by implementing Trickle equivalent feature as a networking function and the daemon program (*trickled*) as an OpenNF controller.

C. Fragment Prevention

Tunneling like VxLAN and STT provides a connectivity between L2 virtual networks over L3 networks, however, this encapsulation can cause IP packet fragmentation because the size of VM's frame and tunneling header can exceed the MTU size of the physical link.

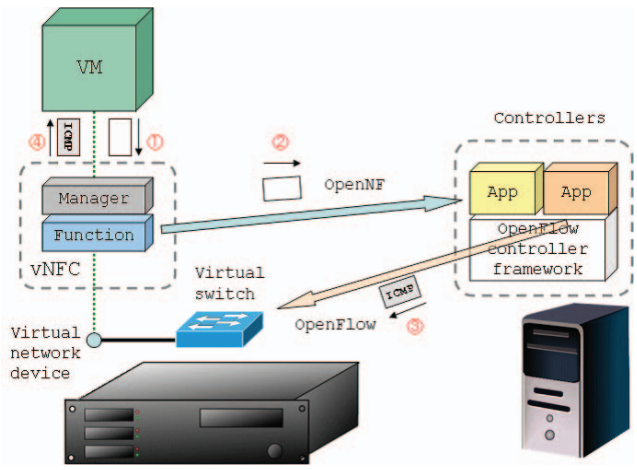

Fig. 7. A sequence of fragment prevention networking function

This fragmentation can be prevented using vNFC as follows. First, a vNFC networking function of the sender side checks each frame size, and it redirects size-exceeded frame to the OpenNF controller. Then, the controller creates an ICMP message (type = 3, code = 4, and appropriate next-hop value) based on the received frame header, and sends the message to the virtual switch as a `OFPT_PACKET_OUT` OpenFlow message. The virtual switch forwards the message to the source VM via the virtual network device. After that, the protocol stack of the source VM adjusts the transmitting frame size according to the informed next-hop value.

A specific example is that vNFC tracks FTP or streaming traffics, and redirects the frame to the controller if the frame size exceeds 1464 bytes for putting the frame into a single VxLAN packet.

V. Evaluation

In this section, we evaluate the performance overhead of vNFC and OpenNF controller. Here, vNFC networking functions are categorized as *standalone* and *controller* types. The standalone-type functions process incoming/outgoing frames within the library only (e.g. encryption). The controller-type functions ask the vNFC controller how to handle the frames. Table II and III show the experimental environment and the performance evaluation results. The experiment was conducted on a single host machine that both QEMU (VM) and the vNFC controller (Floodlight) are running on.

The performance overhead of the standalone-type was measured as follows: (i) Record a current time before calling `writev()` function of the *syscall hook*. (ii) Record a time again at the *interface* library before corresponding system call invocation. (iii) Calculate the time difference. The result shows that the overhead of the *syscall hook*, *NF manager*, *null* networking function[3], and the *interface* library is 1.6 to 2.0 μs ($\pm 1\sigma$) per transmitting frame, and this overhead can be negligible at the actual VM-to-VM communication.

The overhead of the controller-type function requires additional communication cost compared to the standalone-type, and this cost takes millisecond orders per frame transmission. In addition, further communication cost are added if the controller runs on another host. Therefore, it is reasonable that the vNFC networking functions do not communicate with the controller for 'every' incoming/outgoing frames but rely on it at certain cases such as first frame arrival in a flow.

TABLE II
Experimental environment

Host machine	
CPU	Intel Core i7 (2.67 GHz)
Memory	12 GBytes
Virtual machine	
VMM	QEMU 1.12
VM image	linux-0.2.img[9]
Application	ping
Packet size	512 Bytes
No. packets	50
vNFC Controller	
Framework	Floodlight 0.85

TABLE III
Performance overhead of vNFC

Type	Average (μs)	Worst (μs)	Stdev (μs)
Standalone	1.83	2.72	0.22
Controller	3255.29	11940.1	4106.44

VI. Related Work

NetOpen networking services[13] are a self-contained functions built over OpenFlow-based networks for supporting networking features. It provides L2 switching and VLAN translation services using OpenFlow protocol. The notion of networking services are similar to networking functions of vNFC, however, NetOpen targets lower-layer services that OpenFlow targets. In addition, NetOpen services are executed in both the controller and switches, and there is no VM-dedicated service executors.

[3]Null networking function is implemented as a shared library, and its exported functions just invoke the underlying library's functions.

Exhibit 1043
Microsoft Corporation v. Edge Networking Systems, LLC
007

RouteFlow[14] is a routing dedicated open API that provides centralized IP routing services. Like vNFC, RouteFlow coexists with OpenFlow and provides own specific API, however, RouteFlow focuses on a mapping of virtual networks on physical networking infrastructures with OpenFlow approach. Meanwhile, vNFC focuses on VM (instance) dedicated functionality rather than whole virtual network controlling.

VTL[15] provides transparent extension mechanism of virtual machines' traffic by providing packet handling toolsets, and enables various networking services including subnet tunneling, stateful firewall, local TCP acknowledgments, and TCP keep-alive for virtual networks. Both VTL and vNFC provide transparent functionality for virtual networks, however, VTL does not provide global configurations from a controller that is a essential feature of software defined networks.

FreeNA[2] enables transparent extension of applications' communication data at the socket-layer by inserting networking functions. FreeNA differs from vNFC in that it extends the functionality of application processes rather than virtual machines, it does not support L2/L3 services, and no global configuration mechanism.

VII. Conclusion

SDN/OpenFlow enables dynamic and flexible configuration of physical/virtual networks in an open approach. Considering datacenter networks that hold many tenant (virtual) networks, however, current approaches do not provide application-layer flow control nor tenant-dedicated custom functions.

This paper proposed the virtual networking function container (vNFC) that is an extensible framework for network providers. vNFC transparently manipulates VM's frames to realize various services, such as VM-dedicated firewall, traffic shaping, or Fragment prevention. These services simplify the tasks of OpenFlow switches, and bring additional functionality to virtual networks. OpenNF enables runtime composition of vNFC features by utilizing existing OpenFlow protocol and controller frameworks.

The architectural design and implementation, some example scenarios, and the performance overhead of vNFC are presented in this paper. The evaluation results show that lightweight networking functions do not impact on the performance, but functions that frequently communicate with the controller incur millisecond order cost per frame transmission.

Further study and evaluation are necessary for vNFC and OpenNF such as VM-migration support, kernel-based QEMU support, and evaluation of overhead reduction of virtual switches when vNFC is used.

References

[1] N. McKeown, T. Andershnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, J. Turneron, and H. Balakris, "OpenFlow: Enabling Innovation in Campus Networks", ACM Computer Communication Review, Vol. 38, Issue 2, pp. 69-74, April 2008.

[2] Ryota Kawashima, "Study on Transparently Extending Networking Services Framework for Adaptive Communications", Ph.D. thesis, The Graduate University for Advanced Studies (SOKENDAI), 2010.

[3] Open vSwitch, http://openvswitch.org/

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

[5] M. Mahalingam, D. Dutt, P. Agarwal, L. Kreeger, T. Sridhar, M. Bursell, and C. Wright, "VXLAN: A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks", RFC draft, 2012.

[6] NOXRepo, http://www.noxrepo.org/

[7] Floodlight OpenFlow Controller, http://floodlight.openflowhub.org/

[8] Trema, http://trema.github.com/trema/

[9] QEMU, http://wiki.qemu.org/Main_Page

[10] Fabrice Bellard, "QEMU, a Fast and Portable Dynamic Translator", USENIX Annual Technical Conference, 2005.

[11] B. Davie, Ed. and J. Gross, "A Stateless Transport Tunneling Protocol for Network Virtualization (STT)", RFC draft, 2012.

[12] M.A. Eriksen, "Trickle: a userland bandwidth shaper for Unix-like systems", In Proc. of USENIX Annual Technical Conference, 2005.

[13] Namgon Kim and JongWon Kim, "Building NetOpen Networking Services over OpenFlow-based Programmable Networks", The International Conference on Information Networking, 2011.

[14] M.R. Nascimento, C.E. Rothenberg, M.R. Salvador, C. Correa, S. Lucena and M.F. Magalhaes, "Virtual Routers as a Service: The RouteFlow Approach Leveraging Software-Defined Networks", In 6th International Conference on Future Internet Technologies 2011 (CFI 11), Seoul, Korea, June 2011.

[15] John R. Lange and Peter A. Dinda, "Transparent Network Services via a Virtual Traffic Layer for Virtual Machines", In Proc.s of the 16th international symposium on High performance distributed computing, pp. 23–32, 2007.

View publication stats

Exhibit 1043
Microsoft Corporation v. Edge Networking Systems, LLC
008