Implementation of Clean NDN with Network Virtualization

Junwei Cao[1,2], Shuo Chen[1,3], Zhen Chen[1,2], Ge Ma[1,3], Ziwei Hu[4], Jing Zhou[4], Jing-hong Guo[4]

[1]Research Institute of Information Technology, Tsinghua University, Beijing, 100084, China
[2]Tsinghua National Lab for Information Science and Technology, Beijing, 100084, China
[3]Department of Automation, Tsinghua University, Beijing, 100084, China
[4]State Grid Smart Grid Research Institute, Beijing 102209, China

Corresponding email: jcao@tsinghua.edu.cn

Abstract. Content dissemination is the main usage of current Internet. Named Data Networking (NDN) is a paradigm shift from the traditional host-to-host network to the named content based network by entirely new designed network architecture and protocol. The data retrieval and routing are just based on the name of the data, instead of the location of the data. CCNx, which is the most popular realization of NDN, is designed based on TCP or UDP. While current implementation of NDN is an overlay TCP/IP based network, the major contribution of this paper is to propose a clean implementation of NDN by using Ethernet frame directly to encapsulate named data and a scheme for large-scale deployment of clean NDN over Software Defined Networking (SDN) based virtualized network platform. This paper demonstrate pure named content architecture design, the concrete implementation using Ethernet frame, large-scale deployment over SDN based virtualized network, and its brief evaluation compared with overlay based CCNx.

Keywords: Internet Architecture, Information-centric Networking, Named Data Networking, Content-Centric Networking, Software Defined Network, Network Virtualization.

1 INTRODUCTION

The original motivation of network is to guarantee point to point conversation between two entities. However, the functions of Internet and applications which run on it have changed dramatically and mainly converged to data distribution. Today's Internet is based on to ossifying TCP/IP protocol stack and static host-to-host conversation model to disseminate content whose volume grows rapidly. Information-centric networking (ICN) is a clean-slate approach to the architecture of network which focuses on the data itself rather than the specific physical location. Named Data Networking (NDN), also called Content Centric Networking (CCN), is one of the most popular architectures of ICN proposed by PARC [1]. Some related designs are DONA, TRIAD, PSIRP and etc [2]. NDN is a new network architecture that redesigns the model of network communication, from today's focus on *where* - addresses and hosts, to *what* - the content that users and applications care about. Because of this key feature, basic network functions such as routing, forwarding and security are named data based instead of address and connection based.

The common implementation of NDN, for example, CCNx is deployed upon traditional TCP/IP network. This overlay design is easy to be deployed, adapts to current Internet implementation and could utilize some mature strength of current network architecture including reliable communication, abundant routing optimization algorithms and so on. However, this overlay design deepens the protocol stack and produces some unnecessary overheads. In CCNx, content trunks are split and reassembled through TCP/IP stack. Besides, the routing process is executed twice in both NDN router and common IP router. We propose an implementation of NDN trunks over Ethernet frame protocol and name it as clean NDN implementation. The principle of clean NDN is that NDN protocol directly runs on layer 2 link protocol without IP address, corresponding with the original motivation of NDN to shift *where* to *what* model. In our work, CCNx code is modified to let content trunk be carried in Ethernet frame directly. The brief comparison between overlay NDN and clean NDN is shown as following table and figure 1. Figure 1 shows that the NDN trunk, which is the named data packet, is directly encapsulated in Ethernet frame.

Table 1. Comparison between overlay based NDN and clean NDN

	Overlay NDN	Clean NDN
Deployment	Easy	Hard
Overhead	Much	Little
Protocol stack	Deep	Shallow
Routing	Twice	Once

Large-scale testbed for network emulation is important for function validation and performance evaluation. In our implementation, the clean NDN is deployed over virtualized emulation platform. The technology of virtualization provides fast and elastic methods to deploy large-scale network testbed.

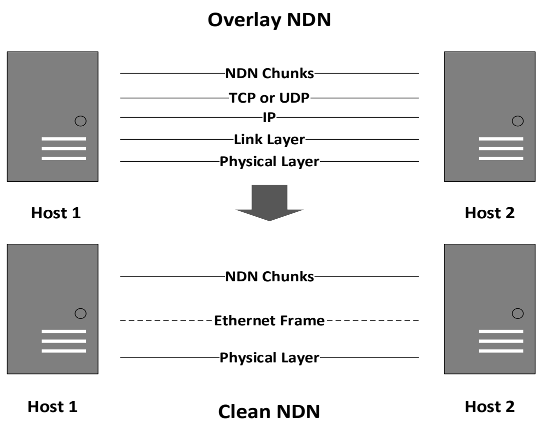

Fig. 1. Clean NDN model

The rest of paper is organized as follows. Section 2 gives some background about our research and experiment. Section 3 describes concrete implementation of clean NDN and the design of clean NDN network. Section 4 demonstrates deployment of clean NDN on virtualized network testbed. Section 5 shows simple function validation, performance evaluation. Section 6 concludes the paper and addresses the future work.

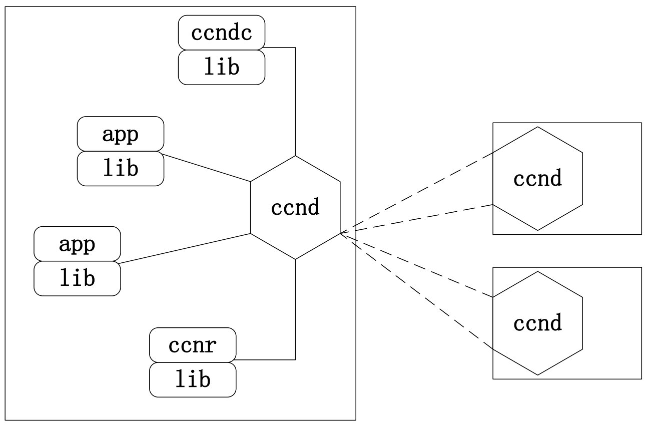

Fig. 2. NDN application development framework

2 BACKGROUND

2.1 CCNx

CCNx[1] is an open source project developed by PARC and a software prototype which implements NDN architecture. NDN focuses on named content, and the functionality of NDN is guaranteed by name based routing property. The specification of CCNx realizes the notion of NDN. It prescribes the format of interest and content object, naming and encoding specification of NDN packets, CCNx node model and so on. The CCNx node model implements three main data structure of NDN: Content Store (CS), Forwarding Information Base (FIB) and Pending Interest Table (PIT). The FIB structure is the forwarding table in NDN, while the basic structure of IP routing table is the set of destination IP and the next hop. Similarly, the FIB of NDN maintains the prefix of the name and the next hop. Content Store is the local cache of NDN node, which records the recent hot NDN packets. PIT keeps track of upstream data requests so that returned data can be sent back to the requesters. The CCNx node runs as a daemon named *ccnd* to process NDN protocol, take charges of forwarding interest referring to FIB table, cache NDN packets in CS and response to interest with content object according to PIT.

Current implementation of CCNx is an overlay network over TCP or UDP. The configuration of FIB is done by program named *ccndc*. It can bind NDN face with TCP or UDP socket. The transmission of interest and content object is through NDN face. In CCNx, the formatted NDN packet is sent through TCP and UDP sockets.

The NDN application development framework is shown as Figure 2. The configuration command program of CCNx, such as *ccndc*, CCNx applications bound with CCNx project, such as *ccnr*, which is the repository of files in NDN node model, are all based on CCNx client library named *libccn*. The CCNx client is a local socket communication program. The command or application message will be encapsulated as CCNx interest or content object and sent by local CCNx client with local socket communication to *ccnd*. The CCNx client will register in *ccnd* as face enstry in FIB. The *ccnd* daemon is the only communication portal among different CCNx hosts. If a developer wants to develop a new application of CCNx, he or she just needs to call the CCNx client library to connect with *ccnd* daemon and follows the CCNx naming and interest specifications. Thus the modification of *ccnd* exterior communication mechanism will not influence CCNx commands and applications.

2.2 Related Work

Van Jacobson et al. [1] firstly proposed NDN architecture in 2009. Principles of NDN are illustrated in this paper, which could be treated as the guideline of NDN research. In the evaluation section of this paper, the NDN implemented over Ethernet is briefly mentioned. However, no open-source code of NDN directly over Ethernet is ever

[1] Http://www.ccnx.org/

released up till now. Yuan, Haowei et al provides detailed analysis of CCNx code and evaluation of CCNx code function efficiency in [11].

Junxiao Shi et al. [4] proposed a link protocol for NDN, NDNLP. NDNLP runs between the NDN chunks and underlying network protocol including TCP, UDP and Ethernet links. The NDNLP protocol receives NDN packets and sends it to lower links. NDNLP provides two main features: fragmentation and reassembling to support different size of packet, acknowledgement and retransmission to support reliable transmission. Current implementation is the mode of proxy which means that an independent daemon receives and forward NDN packets for *ccnd*. The daemon *ndnld* receives CCNx packets from *ccnd*, encapsulates and sends out them in the format of NDNLP, receives remote NDNLP packets, decodes them and sends them to local *ccnd*.

BJ Ko et al. [5] proposed an ICN based data center network architecture to resolve many pain points of current data center network. It also decouples the control plane and data plane. Luca Veltri et al. [6] discussed the integration of ICN and SDN and considered how OpenFlow architecture could be modified to support ICN function. Jing Ren et al. [7] discussed the role of virtualization in ICN deployment. They demonstrated that network virtualization could enable coexistence of different architecture of ICN. Dimitris Syrivelis et al. [13] proposed an ICN architecture with SDN support in forwarding functions but their work is also in the early step.

3 IMPLEMENTATION OF CLEAN NDN

In this section we will present the implementation of clean NDN and design of clean NDN network.

NDN face is bound with practical Network Interface Card (NIC) or virtual NIC. Different faces are directly connected by L2 link. We implement this work by revising CCNx code and build the virtual switched network on Linux servers. In the following section, we will first introduce the supporting technology and briefly demonstrate the modification of CCNx codes and configuration on the Linux server.

3.1 Enabling Techniques

3.1.1 Jumbo frame.

Although the default Maximum transmission unit (MTU) of Ethernet frame is 1500 bytes in most operation systems, the MTU can be changed. In the protocol of CCNx, the default size of packet is 4096 bytes. If the size of MTU is less than the size of CCNx pakcet, the CCNx packet will be split into different Ethernet frames. However, the Ethernet protocol does not support reorganization. Many NICs support larger Ethernet frames, which are called jumbo frames. Jumbo frames are Ethernet frames with MTU 1500 bytes. Conventionally, jumbo frames can carry up to 7000 bytes of MTU. Most Gigabit Ethernet NICs support jumbo frames.

3.1.2 AF_PACKET, Raw socket and libpcap

In our implementation, we use two methods to send and receive Ethernet frame: socket and *libpcap*. To make socket function support layer 2 communication, the configuration is shown as follows. AF_PACKET is the first parameter of function *socket(int socket_family, int socket_type, int protocol)*. This AF_PACKET socket is used to receive or send raw packets at device driver (OSI Layer 2). The *socket_type* could be SOCK_RAW for raw sockets or SOCK_DGRAM for cooked packets with the link level header removed.

Our implementation of clean NDN also supports *libpcap*[2] for transmitting Ethernet frame. *Libpcap* is developed for low-level packet capture and filter. It is supported by Linux and UNIX platform.

3.2 Implementation

In this section, we will demonstrate how to build a one-hop clean NDN network with two Linux hosts over Ethernet frame. CCNx code is enhanced to support Ethernet frame socket transmission and clean NDN face configuration.

3.2.1 CCNx code modification.

Unlike current NDNLP [4] implementation, we directly modified CCNx code. The major work can be summarized into two parts. The first part is to revise *ccnd* related code to support raw socket communication. The destination MAC address is set to be broadcast because there is no protocol like ARP to get the destination address. The second part is to revise *ccndc* related code to add clean NDN face support. We just add some flags or macros to decide the type of protocol or faces and extra Ethernet frame transmitting functions. The practical modification is case branch and the mount of modification is rather little.

Our modification of CCNx code is based on version 0.7.0 and is under development. The current released code could support Ethernet face registration and basic communication. Transmission efficiency and other functions such as neighbor discovery and synchronization will be developed in the future research. Our code is released on Github[3].

3.2.2 One-hop configuration.

This section will demonstrate the configuration for one-hop direct connection of two Linux hosts.

We choose Ubuntu Server 13.04 as the experiment platform. Our server's CPU is Intel Core i7-3770 3.4 G @ 3.40GHz and RAM is 4G. NIC is Intel 82579 GbE which supports 1000M Ethernet. Key points of configuration are listed as follows:

[2] http://www.tcpdump.org/
[3] http://github.com/chenatu/ccnx-underlay

- NIC should support jumbo frame of larger MTU, for example 7000.
- The clean NDN should be detached from TCP/IP protocol stack, in case that the NIC would receive TCP/IP packets that clean NCN face could not handle.
- The installation process of clean CCNx is the same as the common CCNx. The configuration of Ethernet face and processes of basic communication experiment could be referred in the README of the released codes on Github.

3.3 Clean NDN Network Design

The clean NDN nodes cannot run on conventional TCP/IP network, because common router or switch does not support Ethernet frame without MAC address. Bong Jun Ko et al. proposed an information-centric architecture for data center networks. It demonstrates a design of local area network for clean NDN network cluster. We propose a local area network design for clean NDN referring to this work, shown as figure 3.

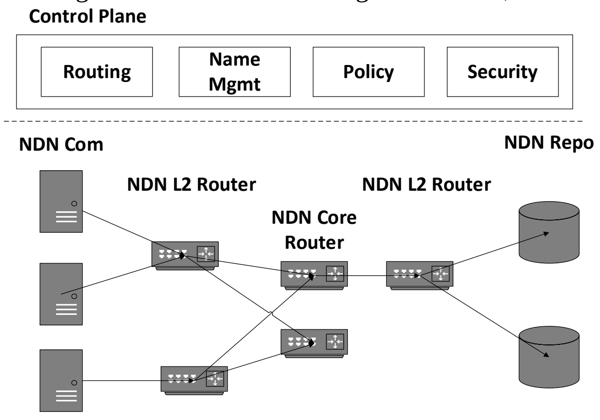

Fig. 3. Local area clean NDN network design

In this local area clean NDN network (LACNN), all the nodes are *ccnd* nodes, which follow the same NDN protocol specification. The NDN trunk underlying link is direct link layer. Routing, forwarding, security, and other network problems are resolved in NDN architecture itself. Although all the nodes run *ccnd*, they focus on different usage and their roles in the network are different.

- For NDN core router, it takes charge of routing and forwarding NDN packets from the border NDN nodes. Its main job is routing, so it may have better FIB lookup performance and have larger forwarding buffer.
- For NDN level 2 router, it is the bridge between border and core network. It may have relatively poorer FIB lookup performance than NDN Core Router but have larger Content Store (CS), for caching hot content locally for better content delivery speed. Here level 2 does not mean link layer.

- Applications of NDN are deployed at the border of the network. NDN Com node is common client following NDN protocol. NDN Repo is server model for public storage.
- The upper control plane takes charge of configuration of *ccnd* FIB, monitors states of important routers, manages the naming and dynamically controls data traffic. [14] The control model will be illustrated in next section.

The design above is the simplest model of LACNN. It covers the brief design principles of clean NDN network and illustrates a blueprint of large-scale clean NDN network deployment. The practical deployment could follow this layered architecture and LACNN will be deployed over virtual network.

4 VIRTUALIZED DEPLOYMENT OF CLEAN NDN

Network virtualization and OpenStack technology give a more programmable and flexible approach to control states of network. The deployment of clean NDN over virtualized platform has the following advantages:

- Elastic and large-scale deployment: The large-scale experiment is hard to conduct and costs much before the SDN era. With the help of virtualization technology, several physical hosts could virtualize thousands of virtual hosts according to the experiment needs. Furthermore, new virtual hosts and virtual network components can be created quickly based on the virtual template.
- Programmable network: The flow table of virtual L2 switch is programmable. Thus, we could easily change the topology and link state of the network. This feature is important for clean NDN deployment because there is no protocol like ARP in clean NDN network now and the virtual *ccnd* host could not know the destination MAC address. Thus all the Ethernet frames in clean NDN are broadcast. Network routing is processed by upper *ccnd* based on CS, FIB and PIT. The programmable flow table controls the topology of virtual *ccnd* nodes and plays the role as multiple network wires.

In our virtualized network testbed, we use a series of virtualized techniques. We choose KVM (Kernel-based Virtual Machine) as the virtualization infrastructure for creating virtual hosts. The overall platform is based on OpenStack[4] project. In our testbed, we mainly focus on computing component for virtual hosts and network component for virtual network.

OpenStack is a series of cloud computing management components to provide an infrastructure as a service (IAAS). Computing, network and storage of data center can be virtualized as a pool which can be monitored and controlled conveniently by administrator through dashboard.

Component Nova is used as the controller of KVM virtual hosts. It mainly handles the requests for creation, deletion and configuration of virtual hosts.

[4] http://www.openstack.org/

Component Neutron is used as controller of virtual network. It mainly handles the control command of the network. OpenvSwitch[5] is the virtual switch software we choose to connect the virtual hosts. It supports OpenFlow protocol to control the flow table. We use Neutron to control the whole virtual network, change the topology and link state of network as we need. Figure 4 shows the basic deployment of virtual network testbed.

The design of the virtualized testbed follows the principle of SDN, which separates control plane and data plane. Jing Ren et al. proposes a short term and a long term approach to use OpenFlow architecture to support ICN. In our deployment, we also refer to this two-step approach. In short term, the L2 virtual switch is the conventional Ethernet Switch. The Neutron Service controls the flow table of the virtual Switch to set the topology and link state of the data network. The management network is based on IP network. The NDN routing is handled by upper NDN virtual node according to CS, PIT and FIB. The virtual switch just focuses on forwarding Ethernet frames based on the flow table. The common virtual switch just handles the NDN packet as Ethernet Frame. The process of forwarding in Ethernet layer and routing in *ccnd* are still redundant.

In long term approach, the open-source OpenvSwitch will be modified to integrate with *ccnd*. The flow table will be modified to be FIB in *ccnd*. The virtual switch could resolve the Ethernet frame, extract the prefix of the NDN packet, and submit to *ccnd* to forward this packet according to CS, PIT and FIB. In this case, the MAC address of Ethernet frame is useless, because the flow table is replaced by FIB in NDN model. The spanning tree protocol (STP) is useless because NDN resolves the loop problem in its own architecture. In control plane of SDN, IP network is replaced by clean NDN network. The OpenFlow protocol is extended by clean NDN network configuration API including [6] :

- Namespace broadcast and configuration
- Face registration, deletion
- Real time FIB, PIT monitor and controller
- CS monitor, policy controller and cache manager
- Public key management

The brief comparisons between short term and long term approach are listed in table 2.

The control commands listed above do not mean to replace the distributed network model of NDN. It is easy to control FIB using SDN. However, PIT is not suitable for central control because large amount of and volatile state of entries in PIT. NDN still operates in distributed mode, but if needed, it can be controlled by SDN. Some of the problems are hard to be resolved by distributed solutions. For example, name prefix is distributed among the network by name flooding in NDN. Obviously, it is troublesome to broadcast some public service name prefix to the whole network. In this case, the central control plane name broadcast is a good option. For another example, in case of the fluctuation of network flow, it is easy to deal with load balance by central control.

[5] http://openvswitch.org/

Table 2. Comparison between short term and long term approach

	Short term	Long term
Virtual switch NDN FIB support	No	Yes
Manage-ment network protocol	IP	NDN
OpenFlow NDN API	No	Yes

In current research stage, the short term approach is adopted and the evaluation in the following section is based on the short term approach. The physical host is HP Z220 with Intel Core i7-3770 3.4 G, RAM 16G, Intel 82579 GbE, OS Ubuntu-desktop 13.04 and KVM. The virtual OS is Ubuntu-server 12.04 because of the requirement of OpenStack. We choose oneStack[6] which is a tool to fast deploy OpenStack service. The virtual network is deployed over 5 physical hosts which emulates 16 virtual hosts.

5 EVALUATION

5.1 Experiment Setting

Two cases are discussed in short term approach. The further evaluation will be done in future work based on long term approach.

- Case 1 - internal hosts: Two NDN vNodes in the same physical host are connected with one NDN vSwitch.
- Case 2 - crossing hosts: Two NDN vNodes in different physical hosts are connected with one physical switch through NDN vSwitch.

5.2 Bulk Data Transmission

Clean NDN and NDN over TCP are compared on network throughput in the two cases above. CCNx repository *ccnr* is used to store the bulk data and command *ccngetfile* is used to fetch file from *ccnr*. Because the network throughput is only concerned, the bulk data is first cached in local memory. This can be done by using *ccngetfile* locally first. The data then will be cached in local CS.

Two cases run successfully over the testbed. The concrete evaluation of clean NDN and overlay based NDN will be discussed in future paper.

[6] https://code.google.com/p/onestack/

6 CONCLUSIONS AND FUTURE WORK

In today's Internet, high efficiency of data transmission is still very difficult and important. We design and implement the clean NDN model. It decouples from traditional address related infrastructure. As we know, it is the first prototype of clean NDN stack based on SDN. The reduction of the protocol stack also improves the network transmission efficiency.

The experimental deployment of NDN with SDN gives us the flexibility to utilize central and distributed control of network. Network virtualization helps us to fast and easily deploy large-scale testbed. It makes us to create an assigned network as we need with commodity machines.

Future work of clean NDN will be summarized in three sections. 1) We will continue to develop CCNx codes on more functions, extending faces on other link protocols and optimizing efficiency. 2) We will modify OpenvSwitch and OpenStack to integrate *ccnd* and to extend OpenFlow protocol in long term approach. Evaluation of long term approach will be done in future paper. 3) The naming mechanism of NDN can integrate networking, storage and computing. A new architecture of distributed computing will be proposed in future.

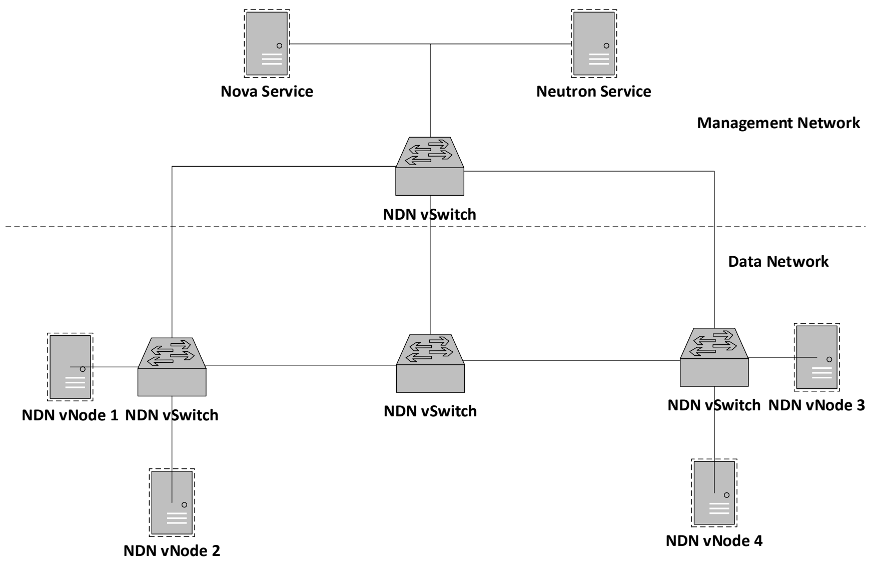

Fig. 4. Virtualized deployment of NDN

7 ACKNOWLEDGMENTS

This work was supported in part by Ministry of Science and Technology of China under National 973 Basic Research Program (No. 2013CB228206), National Natural Science Foundation of China (grants No. 61472200 and No. 61233016), and State Grid R&D project "Research on the Architecture of Information Communication System for Internet of Energy" (grant No. SGRIXTKJ[2015]253).

8 REFERENCES

1. Jacobson, V., et al.: Networking named content. In: Proceedings of the 5th international conference on Emerging networking experiments and technologies. ACM, pp. 1-12. Rome, Italy (2009)
2. Ahlgren, B., et al.: A survey of information-centric networking. Communications Magazine. IEEE 50.7, pp. 26-36(2012)
3. Feamster, N., Jennifer, R., Ellen, Z.: The Road to SDN: An Intellectual History of Programmable Networks. ACM SIGCOMM Computer Communication Review. pp. 87-98(2014)
4. Shi, J.X., Zhang, B.C.: NDNLP: A Link Protocol for NDN. NDN Technical Report NDN-0006 (2012)
5. Ko, B.J., et al.: An information-centric architecture for data center networks. In: Proceedings of the second edition of the ICN workshop on Information-centric networking. ACM，New York (2012)
6. Veltri, L., et al.: Supporting information-centric functionality in software defined networks. In: Communications (ICC), 2012 IEEE International Conference on. IEEE, Ottawa (2012)
7. Ren, J., et al.: The Role of Virtualization in Information-centric Network Deployment. E-LETTER. Vol. 8, No. 4 (2013)
8. McKeown, N., et al.: OpenFlow: enabling innovation in campus networks. ACM SIGCOMM Computer Communication Review 38.2. pp. 69-74 (2008)
9. Pfaff, B., et al.: Extending Networking into the Virtualization Layer. In: The Workshop on Hot Topics in Networks, Hotnets (2009)
10. Braun, S., et al.: CCN & TCP co-existence in the future Internet: Should CCN be compatible to TCP?. In: IFIP/IEEE International Symposium on Integrated Network Management : 5th International Workshop on Management of the Future Internet (ManFI), Ghent, Belgium (2013)
11. Yuan, H.W., Tian, S., and Patrick, C.: Scalable NDN forwarding: Concepts, issues and principles. In: Computer Communications and Networks (ICCCN), 2012 21st International Conference on. IEEE，Munich, Germany (2012)
12. Koponen, T., et al.: Architecting for innovation. ACM SIGCOMM Computer Communication Review 41.3. pp. 24-36 (2011)
13. Syrivelis, D., et al.: Pursuing a software defined information-centric network. Software Defined Networking (EWSDN), 2012 European Workshop on. IEEE (2012)

14. Cao, J.W., Wen, Z., and Wei, T.: Dynamic control of data streaming and processing in a virtualized environment. Automation Science and Engineering. IEEE Transactions on 9.2, pp. 365-376 (2012)