Implementation of a Programmable Service Composition Network using NetFPGA-based OpenFlow Switches

Seok Hong Min*, Yoon Cheol Choi*, Namgon Kim**, Wan Kim***, Oh Chan Kwon***, Byung Chul Kim *, Jae Yong Lee*, Dae Young Kim*, Jongwon Kim**, Hwhangjun Song***
*Dept. of Infocomm Eng., Chungnam National University, Daejeon, 305-764, Korea
{minsh, cyc79i, byckim, jyl, dykim}@cnu.ac.kr
**Dept. of Info. and Comm. Gwangju Institute of Science and Technology (GIST) Gwangju, 500-712, Korea
{ngkim, jongwon}@nm.gist.ac.kr
***Dept. of Computer Science and Engineering, POSTECH, Pohang, 790-784, Korea
{xviii, ochanism, hwangjun}@postech.ac.kr

Abstract— **Future Internet testbed project called FiRST has been started in Korea, where separated testbeds consisting of NetFPGA-based OpenFlow switches are connected to Korea research networks. This testbed will be used for researchers for testing new ideas and innovative protocols of Future Internet. To interconnect separated testbeds, user's layer 2 frames from one local testbed should be encapsulated at the sending OpenFlow switch and tunneled to the destination OpenFlow switch. In this project, we implemented NetFPGA-based OpenFlow Capsulator module and tested its performance. This hardware Capsulator shows enhanced performance compared to the user-level software Capsulator. On the given testbed network, we deployed programmable service composition and QoS routing function at the OpenFlow controller based on user requirements and network status. According to the QoS routing algorithm, optimal routing path can be calculated using service requirements table and network monitoring components. We believe that our suggested controller application can be effectively used for providing dynamic service composition and guaranteed video streaming services.**

Keywords- Future Internet, NetFPGA, OpenFlow, Dynamic Service Composition, QoS routing

I. INTRODUCTION

Future Internet has been a major issue for several years to enhance current Internet and support upcoming future applications. There are a number of interesting researches such as GENI [1], FP7 FIRE [2] and NwGN [3]. These are mainly focused on future Internet architecture and related testbeds to evaluate new protocols. Future Internet testbed requires some advanced concept, such as programmability, virtualization, end-to-end slice, federation, and network resource management. Virtualized programmable testbed makes it easy for researchers to apply new ideas or innovative new protocols.

A large-scale testbed project in Korea is named FiRST(Future Internet Research for Sustainable Testbed)[4], in which platforms based on NetFPGA-based OpenFlow switches are deployed and new service operation and control framework on dynamic virtualized slices is applied. We deployed four separated testbeds consisting of NetFPGA-based OpenFlow switches and connect these testbed to KOREN (Korea Advanced Research Network) [5].

To make an overlay network on the KOREN, user's layer 2 frames from one local testbed should be encapsulated at the sending OpenFlow switch and tunneled to the destination OpenFlow switch. To do this, we implemented NetFPGA-based Capsulator module and tested its performance. This Capsulator shows enhanced performance compared to the user-level software Capsulator.

On the given testbed networks, we deployed programmable service composition network and applied QoS routing based on user requirements and network status. QoS routing algorithm can calculate optimal routing path using service requirements table and network monitoring components. According to our service scenario, a user sends service requirements to OpenFlow controller first and records them in the table. Later, when a new packet arrives at the OpenFlow switch, it is forwarded to the controller. The controller examines the packet and applies QoS routing algorithm based on the service requirement and network conditions such as available bandwidth or network congestion status. OpenFlow controller inserts forwarding entries in the switch and afterwards packets will be processed according to the forwarding table. This table will be changed by OpenFlow controller whenever topology or network status changes.

We'll first introduce basic concept of NetFPGA module and OpenFlow protocol. After that, we'll explain testbed architecture and QoS routing control framework. Lastly, we'll show the implemented NetFPGA-based hardware Capsulator module and its performance.

II. RELATED WORK

In this Section, we first introduce NetFPGA platform and OpenFlow protocol. Then, we explain service composition related research activities.

A. *NetFPGA Platform*

The NetFPGA platform is a network hardware accelerator that can handle packet processing at line rate without CPU participation. The gateware of the NetFPGA is designed in a modular fashion to allow users to modify or reconfigure modules to implement other useful devices. Fig. 1 shows the NetFPGA pipeline which consists of eight receive queues, eight transmit queues, and user data path which includes input arbiter, output port lookup and output queues [6].

We can develop and add our modules to the user data path appropriately. The register interface allows software programs running on the host to exchange data with hardware module.

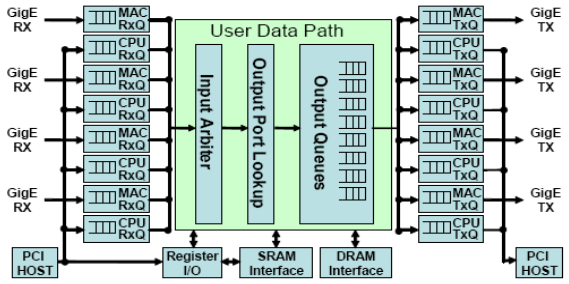


Figure 1. NetFPGA Reference Pipeline

B. *OpenFlow Switch*

OpenFlow provides an open protocol to program the flow table in various switches and routers. OpenFlow controller can add or delete forwarding entries in the switch, so packets are forwarded according to the centralized controller's decision. The OpenFlow switch classifies packets into flows based on 10-tuple to find the appropriate actions associated with the flow. An OpenFlow switch consists of a Flow Table, Secure Channel to the remote controller and OpenFlow protocol [7].

All the intelligence of network traffic control resides in the controller and dedicated OpenFlow switch is a dumb datapath element that forwards between ports as ordered by the controller. Experimenters might control the complete network of OpenFlow switches and run multiple independent experiments on different sets of flows.

C. *Service Composition*

Service composition is a way of building a new application by composing existing component services. It allows efficient provisioning and improved reusability of components in building applications. Although service composition was a concept originally researched in the Web community, today it is utilized by a variety of communities. Multimedia community builds large scale multimedia systems via service composition [8]. In networking community, new networking architectures are introduced which composes network functions to form new protocols and services [9]. RBA [10] and SILO [11] organize communications by composing functional blocks. Also, Ganapathy et al. [12] encompass functions that are typically placed on end-systems and on routers.

III. NETFPGA/OPENFLOW-BASED PROGRAMMABLE NETWORK

In this section, we explain NetFPGA/OpenFlow-based programmable network architecture and implemented functional elements in detail. They are GUI for network and flow visualization, QoS routing control module and NetFPGA-based hardware-accelerated Capsulator.

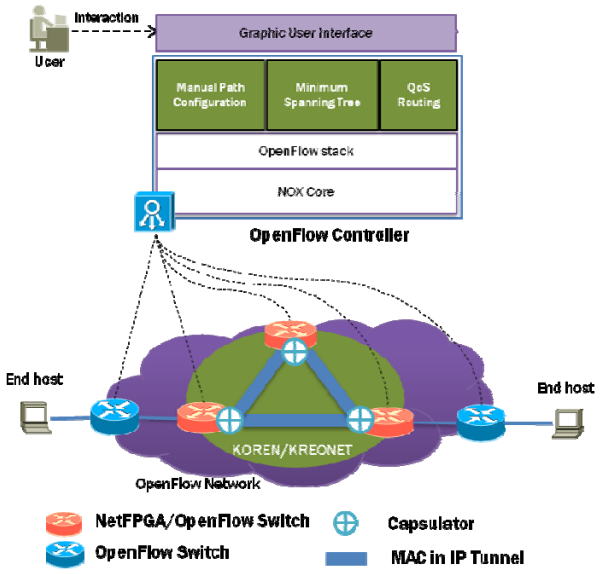


Figure 2. Overall testbed architecture

A. *Overall Architecture*

Network substrate consists of interconnections between NetFPGA/OpenFlow switches as shown in Fig. 2. If users sends networking requirements, OpenFlow controller builds programmable network using network programmability functions to satisfy those requirements. Programmable network can be provided by composing flow's route dynamically. For example, packets can be routed via designated node for a specific service or light loaded path based on their priority. The way for providing programmable network is to make minimum spanning tree between all the OpenFlow switches, to make a tree for satisfying the QoS metrics, or to make traffic route manually by user's control over the network resources. To make an overlay network on the KOREN, tunneling between OpenFlow switches are implemented by using NetFPGA-based Capsulator.

B. *GUI for Network and Flow Visualization*

To visualize flows over the OpenFlow network connection and to control the flow's path manually, we implemented GUI on the OpenFlow controller by extending ENVI.

C. *QoS Routing Control Framwork*

QoS routing control framework is an essential part to provide an experimental environment that guarantees QoS for experimenters in Future Internet testbed. When the experimenter performs service composition experiment on OpenFlow testbed, the experimenter may request a slice with

service requirement such as bandwidth, delay and computing power. As a result, the slice can be consist of nodes that have an ability to perform service composition, and the slices can work independently each other. Thus, service requirement can be sufficiently satisfied during total service time, and then the routing path with satisfying the service requirement is dynamically computed.

The proposed QoS routing control framework is illustrated in Fig. 3. QoS routing control framework can be divided into four components: ENVI, LAVI, Network monitoring component and QoS routing component. ENVI (Extensible network visualization & control framework) [13] is developed to visualize OpenFlow network situation. The experimenter is able to check the status of link connection on its own slice via ENVI in real time. LAVI which is one of NOX components plays a role as a backend server of ENVI. ENVI and LAVI exchange OpenFlow network information through secure channel. ENVI sends service requirements to LAVI in NOX controller, and then LAVI delivers this information to QoS routing component. Network monitoring component observes network conditions such as available bandwidth, link delay, packet loss and queue status of each OpenFlow switch in real time, and then sends this information to QoS routing component periodically. QoS routing component dynamically computes the optimal routing path based on both service requirements by experimenter and network conditions of all OpenFlow switches. In other words, QoS routing component finds routing path to satisfy service requirement of all slices under time-varying network condition, and then updates flow table of corresponding OpenFlow switch by QoS routing information.

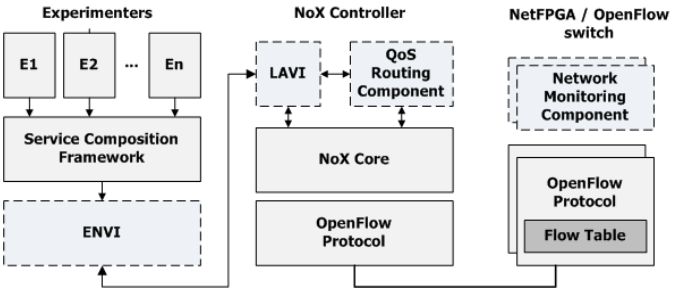


Figure 3. QoS routing control framework

D. NetFPGA-based hardware-accerelated Capsulator

We deployed separated local testbeds at four sites(Seoul, Daejeon, Gwangju, Pohang) using NetFPGA-based OpenFlow switches and applied tunneling scheme to forward OpenFlow frames intact from one OpenFlow site to another OpenFlow site through the KOREN because there are no OpenFlow aware nodes now in the KOREN. We implemented NetFPGA-based Capsulator for high throughput MAC-in-IP encapsulation.

Since user data path between hardware logics is 64-bits wide running, we made 64-bits IP encapsulation header format as shown in Fig. 4. Source (destination) IP address is set as transmitting (receiving) capsulator's IP address. The easiest way to get a MAC address of gateway which is connected to OpenFlow switch is to set register manually using CLI (Command Line Interface). But, it cannot support automatic and dynamic topology configuration because user has to input MAC address information whenever topology changes. To support automatic MAC address table construction, we implemented new application using shell script and JAVA-based GUI as follows. For this, NetFPGA-based OpenFlow switch is connected to the same gateway from both NIC and NetFPGA cards as shown in Fig. 5 and kernel-level ARP table is used as follows.

(1) Obtain gateway MAC address by looking up ARP table of kernel-level OS.

(2) Fill the register of NetFPGA-based OpenFlow switch using obtained gateway MAC address and user-input network information

(3) Download OpenFlow_capsulator bit file and load OpenFlow kernel module using *insmode* command.

(4) Run script for OpenFlow switch operation.

Also, for proper ARP operation, if the ARP request frame is received at the Capsulator from gateway on KOREN, it should be delivered to the OS kernel or processed in the NetFPGA module internally. Currently, kernel-level OS ARP operation cannot be supported, so we implemented internal ARP processing logic in the NetFPGA as follows.

(1) Using register interface, enter MAC and IP address of Capsulator for ARP reply.

(2) After monitoring an entered frame at the Capsulator's port, check if it is an ARP request or not by comparing MAC frame type and destination. If it is an ARP request, activate arp flag and record MAC and IP address.

(3) If the arp flag is set to 1, ARP reply frame is made using an entered MAC and IP information and transmitted to the Capsulator port.

<table>
<tr><td>CTRL</td><td colspan="8">user_data_path</td></tr>
<tr><td></td><td>Bits 63-56</td><td>Bits 55-48</td><td>Bits 47-40</td><td>Bits 39-32</td><td>Bits 31-24</td><td>Bits 23-16</td><td>Bits 15-8</td><td>Bits 7-0</td></tr>
<tr><td>0x00</td><td colspan="6">mac_dst 48</td><td colspan="2">mac_src_hi 16</td></tr>
<tr><td>0x00</td><td colspan="4">mac_src_lo 32</td><td colspan="2">mac_ethertype 16</td><td>ip_ver 4
ip_header_len</td><td>ip_ToS 8</td></tr>
<tr><td>0x00</td><td colspan="2">ip_total_length 16</td><td colspan="2">ip_id 16</td><td colspan="2">ip_flags 3
ip_flag_offset 13</td><td>ip_TTL</td><td>ip_prot 8</td></tr>
<tr><td>0x00</td><td colspan="3">ip_header_checksum 16</td><td colspan="4">ip_src 32</td><td colspan="2">ip_dst_hi 16</td></tr>
<tr><td>0x00</td><td colspan="3">ip_dst_lo 16</td><td colspan="2">udp_src 16</td><td colspan="2">udp_dst 16</td><td colspan="2">udp_length 16</td></tr>
<tr><td>0x00</td><td colspan="3">udp_checksum 16</td><td colspan="6">capsulator_reserved 48</td></tr>
</table>

Figure 4. Encapsulation header format

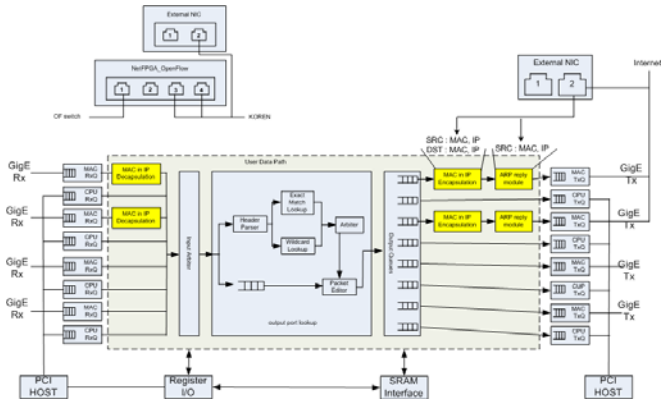
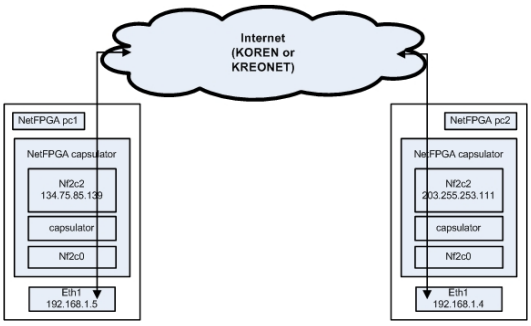


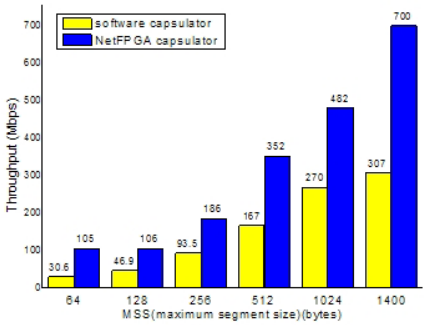
Figure 5. User data path of NetFPGA-based Capsulator

IV. EXPERIMENTATION

First, we tested NetFPGA-based hardware Capsulator using iperf [14] program for various TCP MSS (maximum segment size) on the local testbed topology as shown in Fig. 6. The throughput can be greatly enhanced compared to the user-level software Capsulator as shown in Fig. 6.



(a) Topology



(b) Test results

Figure 6. Local testbed topology and performance results

V. CONCLUSION

In this paper, we introduced flow-based Openflow protocol and Future Internet testbed using programmable NetFPGA system. We implemented hardware Capsulator for MAC-in-IP tunneling to interconnect separate local Openflow testbeds. Our NetFPGA-based hardware Capsulator shows enhanced performance compared to the user-level software-based Capsulator.

We also deployed programmable service composition network and applied QoS routing based on user requirements and network status. According to the QoS routing algorithm, optimal routing path of each slice can be calculated using user service profile and network monitoring components.

We'll deploy MediaX Library for developing media-oriented component service using this testbed. For MediaX Library service, Capture/Display library for heterogeneous input/output device, communication library for flexible media contents and component service module for compressing, decompressing, mixing, tiling and display services will be developed. We believe that our suggested controller application can be effectively used for providing these dynamic service compositions and guaranteed video streaming services.

ACKNOWLEDGMENT

This paper is one of results from the project (2009-F-050-01), "Development of the core technology and virtualized programmable platform for Future Internet" that is sponsored by MKE and KCC. We'd like to express our gratitude for the concerns to support for the research and development of the project.

REFERENCES

[1] GENI: Global Environment for Network Innovations, http://www.geni.net/

[2] FIRE: Future Internet Research and Experimentation, http://cordis.europa.eu/fp7/ict/fire/

[3] Shuji Esaki, Akira Kurokawa, and Kimihide Matsumoto, "Overview of the Next Generation Network," NTT Technical Review, Vol.5, No.6, June 2007.

[4] Jinho Hahm, Bongtae Kim, and Kyungpyo Jeon, "The study of Future Internet platfom in ETRI", The Magazine of the IEEK, Vol.36, No.3, March, 2009.

[5] KOREN: Korea Advanced Research Network, http://koren2.kr/koren/eng/

[6] G. Adam Covington, Glen Gibb, John Lockwood, and Nick McKeown, "A Packet Generator on the NetFPGA Platform", IEEE Symposium on. Field-Programmable Custom Computing Machines (FCCM), April 2009

[7] http://www.openflowswitch.org

[8] J. Van der merwe and C. Kalmanek, "Network programmability is the answer! What was the question again? Or, is there really a case for network programmability?", in Proc. Workshop on Programmable Routers for the Extensible Services of TOmorrow (PRESTO), May 2007.

[9] K. Nahrstedt and W.-T Balke, "A taxonomy for multimedia service compostion," in Proc. of ACM Multimedia'04, Oct. 2004.

[10] R. Branden, T. Faber, and M. Handley, "From protocol stack to protocol heap - Role-Based Architecture," in Proc. of 1st Workshop on Hot Topics in Networking (Hotnets-1), Oct. 2002.

[11] R. Dutta, G. N. Rouskas, I. Baldine, A. Bragg, and D. Stevenson, "The SILO architecture for Service Integration, controL, and Optimization for the Future Internet," in Proc. of IEEE International Conference on Communications (ICC'07), Jun. 2007.

[12] S. Ganapathy and T. Wolf, "Design of a network service architecture," in Proc. of 16th IEEE International Conference on Computer Communications and Networks (ICCCN'07), Aug. 2007.

[13] http://www.openflowswitch.org/wk/index.php/ENVI_Extension_-_Tutorial

[14] iperf, http://sourceforge.net/projects/iperf/