Distributed Packet Processing Architecture with Reconfigurable Hardware Accelerators for 100Gbps Forwarding Performance on Virtualized Edge Router

Satoshi Nishiyama, Hitoshi Kaneko, and Ichiro Kudo

***Abstract*—To implement virtualized service-edge-router functions on carrier networks using general-purpose servers, it is necessary to improve forwarding performance. The required forwarding performance of a service-edge-router reached more than 100-Gbps in bandwidth on carrier networks.**

In this paper, we propose a distributed architecture that involves reconfigurable hardware accelerators with high-level synthesis technology for virtualized service-edge functions to satisfy migration time constraints and improve forwarding performance. The proposed architecture prepares several circuit files for the hardware accelerators according to the utilizations of the network functions in advance and selects a suitable circuit file at the time of migration, instead of generating circuit files for the accelerators every time.

The evaluation of the proposed architecture showed that migration time can be almost the same as the time on configurations without hardware accelerators, and forwarding performance can be on the order of 100-Gbps when a general-purpose server exhibits 10-Gbps forwarding performance.

***Index Terms*—Virtualized service-edge router, hardware accelerator, field-programmable gate array, network function virtualization.**

I. Introduction

The forwarding performance of virtualized network functions has recently improved. For example, 10-Gbps or more forwarding performance of a packet-forwarding function based on 1 million OpenFlow rules can be achieved using a general-purpose server [1]. To connect such equipment to a carrier network, 100-Gbps or more forwarding performance is required for edge-router equipment placed on the edge of the carrier network.

A service-edge function is a function on the carrier networks. The service-edge functions are placed on the edge of carrier Internet-Protocol (IP) networks and provide several network services on carrier networks. To provide network services, a service-edge function generally can provide many packet-processing functions, such as IP multicast, tunnel encapsulation, or session control.

Network-oriented processors have been widely used to ensure the forwarding performance of a service-edge function. Toward network function virtualization (NFV), implementation of a service-edge function on servers that use general-purpose central processing units (CPUs) has been studied.

There are several methods of improving forwarding performance on general-purpose servers. One such method is software-acceleration such as Intel data plane development kit (DPDK) [2], [3]. Furthermore, hardware-acceleration methods that use alternative reconfigurable hardware devices, such as a field-programmable gate array (FPGA), have been extensively studied [4]-[7].

High-level synthesis (HLS) is a technology to efficiently implement functions on an FPGA. The HLS compilers generate register transfer level (RTL) logics that work on the FPGA from high-level programming languages such as C-language. In HLS, the compiler automatically generates the timing design necessary for RTL.

The achievable performance of the RTL generated by a HLS compiler depends on the compiler's ability. Recent HLS technology can implement packet-processing functions exceeding 100-Gbps performance by automatically designing a pipeline on the compiler. We investigated methods that enable the same function to be operated using both a general-purpose CPU and FPGA accelerator, using HLS.

In a previous study, we describe guidelines on how to deploy service-edge functions when the same function can be executed using a CPU and FPGA [8]. Since the resources of an FPGA are finite, it is impossible to execute processing on the FPGA for all packets. Therefore, it is important to design functions that operate on an FPGA to maximize transfer performance.

In the virtualization network function, virtual machine (VM) migration in a short time between several machines is expected [9]. If the function deployment at migration can be rearranged freely, it will be possible to maximize transfer performance considering the FPGA resources. However, for rearrangement of the function deployment for an FPGA, a circuit file of the FPGA is necessary for each function deployment. If FPGA circuit files are generated for every migration, migration cannot be completed within a realistic time.

In this paper, we propose an architecture to develop a service-edge function that combines a CPU and FPGA and enables over 100-bps processing during inter-chassis migration. We also evaluate the migration time and transfer performance with our architecture.

II. Implementation of Service Edge Function

The next generation network (NGN) is an integrated network that can provide multimedia services such as

Manuscript received December 30, 2016; revised March 30, 2017.

The authors are with the NTT Network Service Systems Labs, Nippon Telegraph and Telephone Corporation, 3-9-11, Midori-cho Musashino-shi, Tokyo, Japan (e-mail: nishiyama.s@lab.ntt.co.jp, kaneko.hitoshi@lab.ntt.co.jp, kudo.ichiro@lab.ntt.co.jp).

doi: 10.18178/ijfcc.2017.6.2.485

Internet and telephone based on IP technology. The Telecoms and Internet converged Services and protocols for Advanced Network (TISPAN) has prepared key documents on NGN specifications [10].

A service-edge function is classified into C-plane and U-plane processing functions in the NGN. The C-Plane processing functions execute control functions such as the access relay function and process-control packets. The U-plane processing functions execute transfer functions such as resource control enforcement function (RCEF), and process high-speed packet forwarding with flow-control, encapsulation, and so on.

The U-plane processing functions of a service-edge function consists of a combination of multiple header identifications and multiple actions such as filtering and policing. Note that processing functions according to the type of service subscribed by the user are provided for each user on service-edge functions.

There are two types of configurations for developing these functions, i.e., a centralized deployment configuration represented by servers and a distributed deployment configuration represented by carrier-grade routers (Fig. 1). In the centralized deployment configuration, the CPU and dynamic random access memories (DRAMs) execute both C-plane and U-plane processing. In the distributed configuration, the CPU and the DRAMs execute C-plane and U-plane processing with network processors.

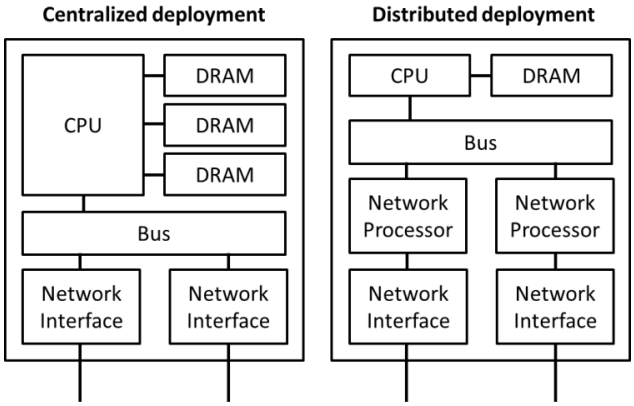

Fig. 1. Centralized and distributed deployment configurations.

In the distributed deployment configuration, it is possible to execute U-plane processing by distributed network processors to improve forwarding performance. However, carrier-grade routers use dedicated network processors for the routers, so it is difficult to migrate functions between routers that use different network processors.

Therefore, we discuss methods of combining FPGA accelerators with a general-purpose server to implement the distributed deployment configuration using general-purpose devices in this paper.

III. FPGA Accelerator Configuration

In this section, we describe the configuration for virtualized service-edge functions with a CPU and FPGAs. In the method of combining different devices, it is necessary to consider how to distribute the processing.

There are two types of task-distribution methods for distributing processing with FPGAs and a CPU: the CPU distributes tasks and FPGAs distribute tasks. The former is used for image processing and video encoding, and OpenCL is a conventional method [11], [12]. The latter has been used conventionally for separating the control and data plains in routers and has been demonstrated by OpenFlow [13] in recent years.

The former method has a bottleneck in distribution processing by the CPU so it is difficult to improve packet-transfer performance. In this paper, we examine the configuration on the latter method (Fig. 2).

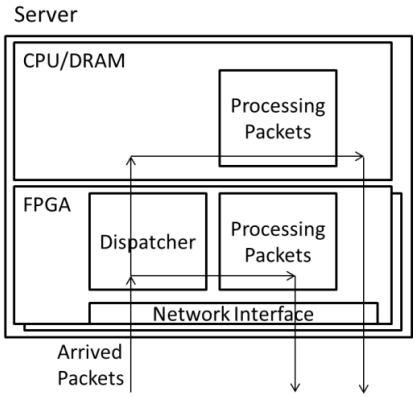

Fig. 2. Distribution configuration for packet processing.

A packet that has arrived from network interfaces is first input to the distribution-function unit (Dispatcher) in the FPGA. The Dispatcher has a forwarding table to identify packet-header information, such as a destination IP address and a port number, and transfers the packet to the CPU or FPGA according to the result of identification with the lookup table. Then, U-plane processes, such as encapsulation and rate-limit, are executed on the packet (Processing Packets). Finally, the packet is output from the network interfaces.

In this paper, it is assumed that the lookup table is assigned for each network service. The capacity of lookup tables becomes increasingly necessary as the number of users that are provided network services increases.

IV. Virtualized Service Edge on Distributed System

There are many types of FPGA-accelerator products, and the capacities of the FPGAs are different. The lookup-table capacities for each network service on an FPGA can be maximized by generating a circuit file of the FPGA by taking into account the desired network requirements and FPGA capacity.

Let us consider VM migration carried out between general-purpose servers equipped with accelerators having different FPGA capacities. The VM-migration procedure is as follows (Fig. 3).

1) Exchange configuration information between servers.
2) Transfer the VM image from the source server to the destination servers. This VM image includes information of the lookup table. The circuit image stored in the FPGA is not transferred.
3) Generate a circuit image so that the size of the circuit fits the capacity of the FPGA, write the circuit image, and transfer lookup-table information to the FPGA.
4) Switch the route from the source server to the destination

server.

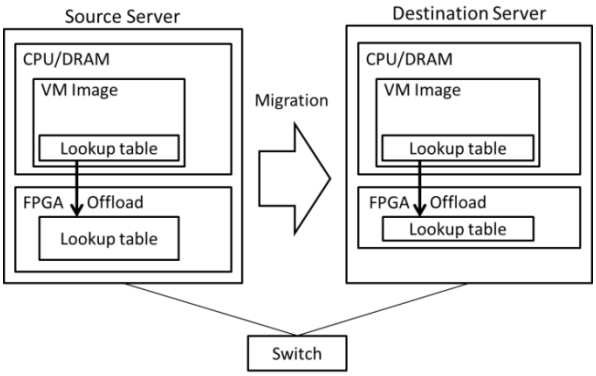

Fig. 3. Distribution configuration for packet processing.

Let us also consider maximizing forwarding performance in this configuration. Assuming that the packet-forwarding performance of the CPU is T_c (bps), packet-forwarding performance of the FPGA is T_f (bps), and ratio of the traffic processed by the FPGA to the input traffic is β. If T_f is sufficiently larger than T_c, the forwarding performance of the entire system T_a is modeled as follows.

$$T_a = T_c/(1-\beta) \quad (1)$$

For example, if the FPGA can handle 9,000 users in the FPGA against a total of 10,000 users and the traffic bandwidth of each user is even, $\beta = 0.9$ and $T_a = 10T_c$. This means that the forwarding performance with accelerators is 10 times better compared with the case without FPGA accelerators.

To improve forwarding performance, it is preferable to ensure the lookup-table capacity in the FPGA to maximize β. The lookup-table capacity is limited by the amount of circuit resources on the FPGA. Therefore, to maximize forwarding performance, it is necessary to generate an FPGA circuit in which the lookup-table capacity fits the capacity of the FPGA.

As mentioned above, a service-edge function consists of various network-service functions and has a lookup table for each network service. To generate an FPGA circuit having a lookup-table capacity optimized to each FPGA that has various resource capacities, HLS and circuit generation according to each FPGA are necessary for every combination of FPGA and lookup table. Therefore, it is difficult to ensure migration time to be within several minutes while improving forwarding performance when circuit generation is required for every migration.

V. Proposed Architecture

In this section, we propose an architecture that involves a FPGA circuit-file server that can manage multiple circuit-files for an FPGA to improve forwarding performance and ensure short migration time. The configuration of the architecture is shown in Fig. 4.

A method of the VM migration on this configuration is as follows.

1) Exchange configuration information between servers.
2) Notify the configuration-management server of the FPGA type and lookup-table information.
3) The configuration-management server has multiple circuit images and selects a circuit-file from the FPGA capacity and required lookup-table size.
4) Transfer the circuit file of FPGA from the file server to the destination server.
5) Transfer the VM image from the source server to the destination servers. This VM image includes information of the lookup table.
6) Write the circuit file to the FPGA of the destination server and register the lookup-table information from the CPU to the FPGA.
7) Switch the route from the source server to the destination server.

The file management server prepares several circuit files for the hardware accelerators according to the utilizations of the network functions in advance and selects a suitable circuit file at the time of migration. The utilization of the network functions tends to the number of users that uses network services.

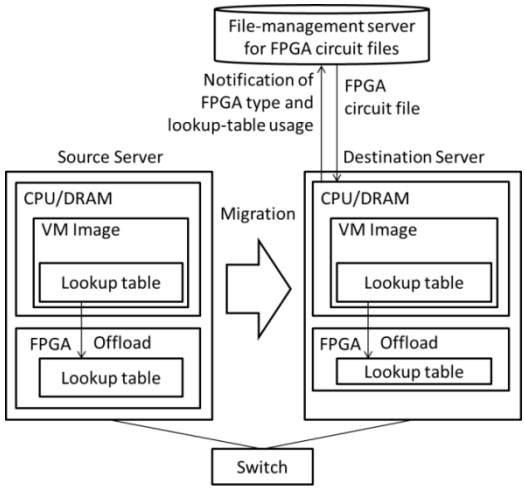

Fig. 4. Configuration of proposed architecture.

With this method, the number of FPGA circuit files stored in the file-management server limits the optimization of forwarding performance. One way to optimize performance involves generating and storing FPGA circuit files for all lookup tables on the file server in advance. However, the capacity of the file server is limited, so all circuit files cannot be stored. Therefore, it is required that the file server prepares several circuit files corresponding to several lookup-table capacities and selects a circuit-file that optimizes forwarding performance.

Define the value s as the number of network-service types provided by the service-edge function, c as the capacity of the FPGA circuit resource, r_i ($i = 1, \ldots, s$) as the FPGA resource consumption per user by service i, N_i ($i = 1, \ldots, s$) as the number of users that subscribe to service i, and n_i ($i = 1, \ldots, s$) as the number of users that can be configured to the lookup table on the FPGA. The restriction of FPGA capacity is expressed as follows.

$$c > \sum_{i=1}^{s}(r_i n_i) \quad (2)$$

Also, assuming that the user traffic bandwidth γ_i is constant for each service i and β is determined by the ratio of

the number of subscribers N_i to that processed by the FPGA accelerator n_i, traffic offload ratio β is expressed as follows.

$$\beta = \sum_{i=1}^{s} \gamma_i \min(n_i, N_i) / \sum_{i=1}^{s} \gamma_i N_i \quad (3)$$

In step 3 of this method, a circuit file, which maximizes β in Eq. (3), is selected from previously prepared circuit files. To prepare circuit files that previously maximized β for any number of subscribed users, the lookup-table capacities of the circuit files are required to satisfy the max-min condition,

$$\max_{n} \min_{N} f(n, N) \text{ s.t. } c > \sum_{i=1}^{s} (r_i n_{i,j}), g(N_1, \ldots, N_m) > 0$$

$$f(n, N) := \max_{j} \beta(n^j, N),$$

where m is the number of previously prepared circuit files, $n_{i,j} > 0$ ($i = 1, \ldots, s$ and $j = 1, \ldots, m$) is lookup-table capacity, the number of service i users that can be configured to the circuit file j, $g(N_1, \ldots, N_m) > 0$ is a function defining the domain of the number of users N_i of each service, and

$$n^j := [n_{1,1} \cdots n_{(s,1)}]^{\mathrm{T}}$$

$$n := [(n^1)^{\mathrm{T}} \cdots (n^m)^{\mathrm{T}}]^{\mathrm{T}}$$

$$N := [N_1 \cdots N_m]^{\mathrm{T}}.$$

VI. Migration Time and Forwarding Performance

A. Migration Time Evaluation

In this section, we determine if migration including FPGA accelerators is possible with a sufficiently small increase in migration time compared with the migration time in the configuration without accelerators with the proposed architecture.

In the VM image synchronization between the CPU and the DRAMs, if it is assumed that the bandwidth that can be ensured between the servers is about 100 Mbps and the size of the VM image is about 2 Gbytes, synchronization can be expected to take several minutes [9]. The increase in migration time compared to that in the configuration without an FPGA accelerator is the time required for steps 3, 4, and 6 in the previous section. It is considered sufficiently practical if this additional time is almost equivalent to the synchronization time, i.e., several minutes.

In step 3, if the number of circuit files is sufficiently small, from several to 10, the selection of the circuit file will be completed within seconds, even if all circuit files are linearly searched. In step 4, it can be assumed that the size of the circuit file is about several hundred Mbytes at most, so the transfer of the circuit file can be completed in ten seconds if network bandwidth is ensured to be 100 Mbps. In step 6, because the previously generated circuit files are prepared with the proposed architecture, it is expected to be completed within a minute as a writing time to the FPGA.

Therefore, the total time increase is several minutes, which is considered within a practical range.

B. Forwarding Performance Evaluation

In this section, we evaluate the proposed architecture by numerical simulation of the traffic offload ratio and forwarding performance in respect of the number of FPGA circuit files on the file management server. For use in numerical examples, assume that the range of the number of users N for each service is as follows.

$$g(N_1, \ldots, N_m) = k - \sum_{i=1}^{s} (a_i N_i) > 0$$

This expression is a condition in which the weighted sum of the number of users of each service is less than a certain value k.

Fig. 5 shows a numerical example of traffic offload ratio β processed by the accelerator with the proposed architecture. In this example, it is assumed that the number of service types $s \in \{2,3,4\}$, and each service consumes the same circuit resources and traffic bandwidth per user, $r_i = a_i = 1, \gamma_i = 1, c = k = 1000$. In this example, for any number of service types, the traffic offload ratio β in the accelerator improves, compared with only preparing a circuit file, with the proposed architecture preparing multiple circuit files.

In the NGN architecture, the number of network services is usually two or three, such as Internet access, voice and video multicast. Therefore, the traffic offload ratio can be 0.7 or more if the number of circuit files can be more than 10 files.

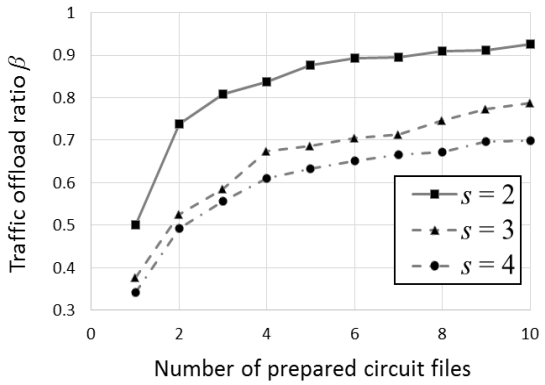

Fig. 5. Numerical example of traffic offload ratio β.

Forwarding performance is derived from Eq. (1) with β obtained in Fig. 5. Fig. 6 shows the forwarding performance derived when $s = 2$. In the evaluation, it was assumed that the transfer performance T_c = 10 Gbps for a general-purpose server alone. As a result, 100-Gbps forwarding performance can be achieved with the proposed architecture if at least 8 circuit files are prepared when the number of network services $s = 2$.

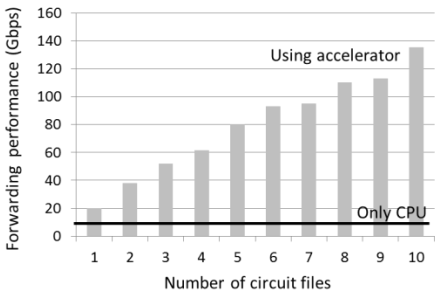

Fig. 6. Numerical example of forwarding performance.

VII. Conclusion

In this paper, we proposed an architecture to improve forwarding performance for a service-edge function on the configuration using an FPGA as the accelerator of the virtualization network function. High forwarding throughput and short migration time are achieved by the proposed architecture.

The proposed architecture prepares several circuit files according to the utilizations of the network functions in advance and selects a suitable circuit file at the time of migration instead of generating circuits for the FPGA every time. Numerical examples from our evaluation showed that migration time of several minutes, which is almost the same as the time on configurations without hardware accelerators, and forwarding performance on the order of 100 Gbps are possible when the performance of a general-purpose server is 10 Gbps. The implementation method for automation of generating FPGA circuit files is future work.

References

[1] Lagopus switch. [Online]. Available: https://lagopus.github.io/

[2] Intel. Data plane development kit. [Online]. Available: http://dpdk.org

[3] Y. Ohara *et al.*, "Revealing the necessary conditions to achieve 80Gbps high-speed PC router," in *Proc. the Asian Internet Engineering Conference 2015*, pp. 25-31, 2015.

[4] K. Blaiech *et al*., "Data plane acceleration for virtual switching in data centers: NP-based approach," *2014 IEEE 3rd International Conference on Cloud Networking*, pp.108-113, 2014.

[5] J. F. Zazo *et al*., "A PCIe DMA engine to support the virtualization of 40 Gbps FPGA-accelerated network appliances," *ReConFigurable Computing and FPGAs*, pp.1-6, 2015.

[6] P. Paglierani *et al*., "High performance computing and network function virtualization: A major challenge towards network programmability," *IEEE International Black Sea Conference on Communications and Networking*, pp. 137-141, 2015.

[7] G. Brebner *et al*., "High-speed packet processing using reconfigurable computing," *IEEE Micro*, vol. 34, no. 1, pp. 8-18, 2014.

[8] S. Nishiyama, T. Osaka, H. Kaneko, and I. Kudo, "A study of virtualized service edge functions using accelerators, netrosphere: towards the transformation of carrier networks," *IEICE Technical Report*, vol. 115, no. 483, pp.335-340, 2016.

[9] K. Mochizuki, H. Yamazaki, and A. Misawa, "Bandwidth guaranteed method to relocate virtual machines for edge cloud architecture," in *Proc. APNOMS*, TS8-3, Sep. 2013.

[10] ETSI, "NGN functional architecture V1.1.1," ES 282 001, 2005.

[11] A. Munshi, "The opencl specification," *IEEE Hot Chips 21 Symposium (HCS)*, 2009.

[12] J. Stone *et al*., "OpenCL: A parallel programming standard for heterogeneous computing systems," *Computing in Science and Engineering*, vol. 12, pp. 66-73, 2010.

[13] N. McKeown *et al*., "OpenFlow: Enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69-74, 2008.

Satoshi Nishiyama is a research engineer in NTT Network Service Systems Laboratories. He received his B.E. and M.E. in system control engineering from Tokyo Institute of Technology, Japan in 2007 and 2009. Since 2009, he has been working at NTT corporation as a research engineer. He currently carries out research and development of next generation network and carrier network architecture.

Hitoshi Kaneko is a senior research engineer in NTT Network Service Systems Laboratories. He received his B.E. in electrical engineering in 1987 from Tokyo Institute of Technology. He joined Software Laboratories, NTT, in 1987, where he studied CHILL programing language. He joined NTT Network Service Systems Laboratories in 2001, where he studied edge router architecture.

Ichiro kudo is a senior research engineer in NTT Network Service Systems Laboratories. He received his B.E. in electrical engineering in 1998 and his M.E. in informatics in 2000 from Kyoto University. He joined business communications headquarters, NTT WEST, in 2000 and work on the construction of a IP network between financial institutions. He is currently investigating and developing network security technology using the edge router, DPI, and security controller for the next-generation NTT-NGN.