Low-Latency Physical-Layer Function Chaining Using Inter-Container Shared Memory for Fully Virtualized Access Networks

Takahiro Suzuki, Sang-Yuep Kim, Jun-ichi Kani and Tomoaki Yoshida

NTT Access Network Service Systems Laboratories, NTT Corporation, 1-1 Hikari-no-oka, Yokosuka-Shi, Kanagawa, 239-0847 Japan

tkhr.suzuki@ntt.com

Abstract: This paper proposes novel physical-layer function chaining utilizing inter-container shared memory for fully virtualized access systems. Our containerization of softwarized 10G-EPON physical coding sublayer functions reduces latency from 1.56 ms to 0.408 ms.

1. Introduction

With the emergence of more sophisticated applications, network equipment must efficiently meet different user requirements by replacing application specific integrated circuit (ASIC)-based dedicated systems with general-purpose components [1]. Broadband Forum (BBF) and Open Networking Foundation (ONF) have developed network function virtualization (NFV) and software-defined network (SDN) techniques for passive optical network (PON) systems [2, 3]. SDN-enabled broadband Access (SEBA) is a critical advance as it allows PON systems to be based on general-purpose servers and whitebox optical line terminals (OLTs) by abstracting differences in PON standards and development vendors. SEBA is being commercially deployed by some network carriers in order to reduce development cycle times and capital expenditure (CAPEX). To maximize these advantages, expansion of software functions continues to be studied. Related studies have examined and demonstrated the software implementation of dynamic bandwidth allocation (DBA) [4], and the aggregation switch has been softwarized. We further expanded the software region by using graphics processing units (GPUs) as accelerators to achieve real-time softwarization of 10G-EPON physical coding sublayer (PCS) [5] and 10-Gb/s coherent receiver digital signal processing (DSP) [6]. Furthermore, the software processing latency of 10G-EPON PHY has been improved to approximately 0.5 ms [7]. However, these physical-layer implementations are limited to processing in bare metal servers, and function chaining, which involves the transfer of physical-layer signals between multiple applications, has not been verified as shown in Fig. 1(a). We target the microservice architecture shown in Fig. 1(b) that virtually divides the execution environment by using containers for each function; it enables application development and function chaining for each function. In conventional containerization, data is transferred across the transport layer using relatively low-layer methods such as transmission control protocol (TCP) via virtual network interface card (vNIC), which greatly increases latency.

In this paper, we propose a novel physical-layer function chaining method that utilizes inter-process shared memory. To manage the timing of function chaining, it passes signal data between applications of physical-layer functions by adding update flags to the shared memory entries. Implementing the method in software significantly reduces the latency of inter-container communications and matches the performance of bare metal implementation.

2. Physical-layer function chaining method utilizing inter-container shared memory

Figure 2(a) shows a conventional inter-container communication method [8]. The memory region is split for each container. For data transfer of GPU memory from application (app) 1 to app 2, after accessing data in GPU memory region of container 1 and transferring it to CPU memory, app 1 outputs the data in the CPU memory from vNIC. It is input to the bridge via virtual Ethernet port (veth) in the host operating system (OS). The data is sent to app 2 in container 2 via veth and vNIC. Finally, the data is stored in the CPU memory region of container 2 and then transferred to GPU memory. TCP is used for this communication as a relatively low layer method. This method is sufficient for upper-layer applications that only need to pass relatively small amounts of data, but this causes an increase in latency for physical-layer functions that must transfer large amounts of data at high repetition rates.

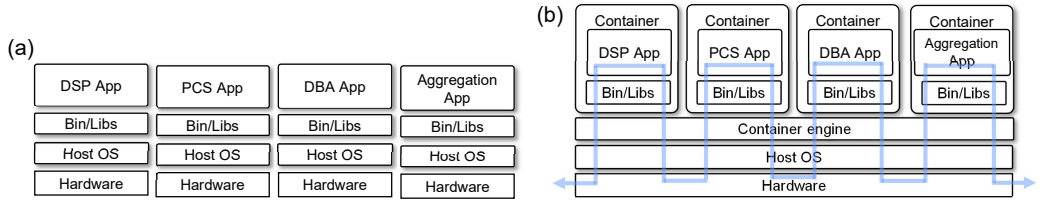

Fig. 1: (a) Bare metal execution of each softwarized application and (b) microservice architecture and chaining of each application with containerization in access systems.

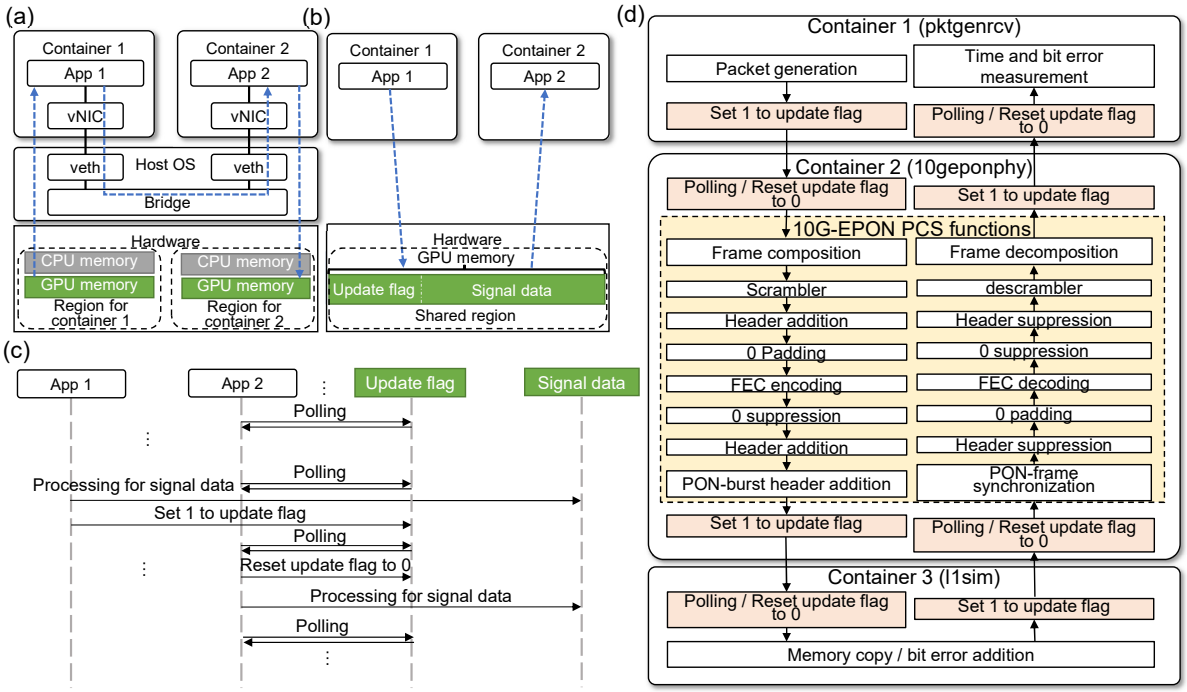

Fig. 2: (a) Function chaining with conventional inter-container communication, (b) proposed function chaining method with inter-container shared memory, (c) sequence diagram of the proposed method, and (d) verification configuration applying the proposed method to 10G-EPON PCS functions.

Figure 2(b) illustrates the proposed physical-layer function chaining method based on inter-container shared memory. This memory is divided to two regions: signal data and update flags. To implement the inter-container shared memory, we utilize the NVIDIA CUDA application programming interfaces (APIs) generally provided for inter-process communication (IPC). Specifically, after allocation of GPU memory by cudaMalloc, cudaIpcGetMemHandle creates an inter-process memory handle for the allocated GPU memory, and, from another process, cudaIpcOpenMemHandle opens an inter-process memory handle and gets a GPU shared memory pointer. In the inter-container shared memory, the signal data represents data passed between apps 1 and 2, and an update flag is added to manage the timing of passing data between apps. Update flag entry of 1 signifies that processing is completed and data can be passed, while 0 means that processing is not completed. Figure 2(c) shows the sequence diagram that illustrates how the update flag is utilized. App 2 repeatedly accesses GPU memory and checks whether the update flag is 1 by polling to confirm the timing of passing data from app 1. After completion of signal data processing, such as PCS functions, by app 1, the update flag is set to 1. When app 2 detects the change of the update flag to 1, the flag is reset to 0, and app 2 starts its processing of the data. After that, the polling is repeated again.

3. Performance evaluation

To evaluate the latency performance of the proposed method in combination with softwarized 10G-EPON PCS, we implemented the container configuration shown in Fig. 2(d), which allows evaluation of the communication between three containers. The verification configuration was mainly composed of three containers named *pktgenrcv*, *10geponphy*, and *l1sim*. *10geponpy* was the container for implemented 10G-EPON PCS functions. Loopback evaluations were performed by adding the function of PON-burst header addition to downstream PCS functions. For performance measurements, *pktgenrcv* was responsible for packet generation and time and bit error measurement, while *l1sim* had performed memory copy and bit error addition. Three other containers, *mps-daemon*, *exclusive-mode*, and *memmng*, were launched for initialization. The NVIDIA multi-process service (MPS), which allows multiple processes to use a single GPU without interfering with each other, was launched by *mps-daemon*. For this service, the GPUs must be set in exclusive mode by *exclusive-mode*. *memmng* creates the inter-container shared memory used by the containers. As the container platform, docker and docker-compose tool were utilized. To build and run GPU-accelerated containers, NVIDIA Container Toolkit was installed, and the docker image of nvidia/cuda:11.4.3-base-ubuntu20.04 (published on Dockerhub) was utilized. The server was equipped with two Intel Xeon E5-2699v4 CPUs and an NVIDIA Tesla A100 GPU.

Figure 3(a) shows the terminal screen when launching containers. The command of ”docker-compose -d” successfully launch the six containers mentioned above, and the listed state retrieved by the command of ”docker-compose ps” also shows that five containers are in the up state. Note that only exclusive-mode containers exit after processing is complete. Figure 3(b) shows the latency performance of just function chaining without any

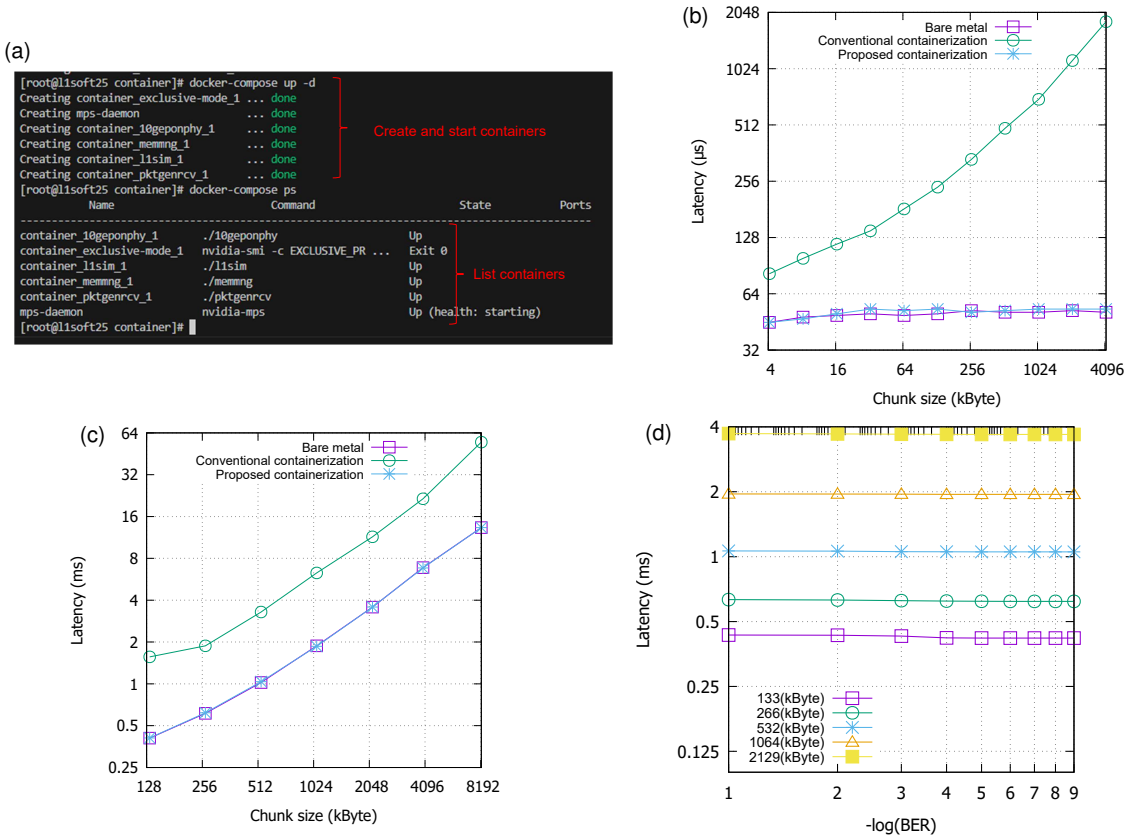

Fig. 3: (a) Terminal screen when launching containers, (b) latency performance of just function chaining without 10G-EPON PCS functions, (c) latency performance of function chaining including 10G-EPON PCS functions, and (d) Latency versus BER for each chunk size.

processing, which means the verification configuration that directly connects *pktgenrcv* with *l1sim* in Fig. 2(d). We compared three methods, the conventional containerization, the method proposed in Section 2, and bare metal implementation. The bare metal implementation ran each function on multiple processes and passed signal data by sharing global memory without containerization. Latency was measured while changing the data length of the transfer unit, chunk size. While the latency of the conventional method increases with the chunk size, that of the proposed method is significantly low and constant regardless of the chunk size; it matches that of the bare metal implementation. Figure 3(c) shows the latency when running 10G-EPON PCS functions. For example, when the chunk size is about 128 kBytes, our containerization proposal achieved performance similar to that of bare metal and compared to conventional containerization, it reduced the latency from 1.56 ms to 0.408 ms which means that 74 % latency reduction is achieved. Additionally, the impact of BER on latency was measured as shown in Fig. refeva(d) since the computation load of the forward error correction (FEC) algorithm can increase depending on the number of bit errors. The results show that latency was basically insensitive to BER.

4. Conclusion

This paper proposed low-latency physical-layer function chaining using inter-container shared memory. It achieved low and constant latency for inter-container communication for various chunk sizes, and containerization with 10G-EPON PCS functions demonstrated latency reductions, from 1.56 ms to 0.408 ms, compared to the conventional method.

References

1. NTT Docomo, “White Paper 5G Evolution and 6G (Version 5.0)” https://www.docomo.ne.jp/english/corporate/technology/whitepaper_6g/
2. BBF, sTR-384 Cloud Central Office Reference Architectural Framework, Jan. 2018.
3. S. Das, ”From CORD to SDN Enabled Broadband Access (SEBA) [Invited Tutorial],” in Journal of Optical Communications and Networking, vol. 13, no. 1, pp. A88-A99, Jan. 2021.
4. D. R. Mafioletti et al., ”Demonstration of a low latency bandwidth allocation mechanism for mission critical applications in virtual PONs with P4 programmable hardware,” 2022 Optical Fiber Communications Conference and Exhibition (OFC), pp. 1-3, 2022.
5. T. Suzuki, et al., ”Demonstration of Fully Softwarized 10G-EPON PHY Processing on A General-Purpose Server for Flexible Access Systems”, Journal of Lightwave Technology, vol. 38, Issue 4, pp. 777-783, Feb. 2020.
6. S. -Y. Kim, T. Suzuki, J. -I. Kani and T. Yoshida, ”Demonstration of Real-time Coherent 10-Gb/s QPSK Reception Implemented on a Commodity Server,” 2021 European Conference on Optical Communication (ECOC), pp. 1-4, 2021.
7. T. Suzuki, S. -Y. Kim, J. -i. Kani and T. Yoshida, ”Low-Latency PON PHY Implementation on GPUs for Fully Software-Defined Access Networks,” in IEEE Network, vol. 36, no. 2, pp. 108-114, March/April 2022.
8. A., Gabriele, T. Cucinotta, L. Abeni, and C. Vitucci. ”Comparative Evaluation of Kernel Bypass Mechanisms for High-performance Inter-container Communications.” In CLOSER, pp. 44-55. 2020.