Reducing Network Traffic Storage Overhead: A Hardware-Accelerated Lossless Data Compression System

Puguang Liu, Shuhui Chen
College of Computer Science and Technology, National University of Defense Technology

ABSTRACT

Skyrocketing network traffic volume brings significant overhead of traffic storage. We propose a hardware-accelerated lossless data compression system to reduce traffic storage overhead. The prototype evaluation validates the excellent resource efficiency and performance scalability of our traffic compression system.

CCS CONCEPTS

• **Networks** → *Network management*; • **Hardware** → **Hardware accelerators**.

KEYWORDS

Traffic Storage, Hardware Acceleration, Lossless Data Compression

ACM Reference Format:
Puguang Liu, Shuhui Chen. 2022. Reducing Network Traffic Storage Overhead: A Hardware-Accelerated Lossless Data Compression System. In *Asia-Pacific Workshop on Networking (APNet '22), July 1–2, 2022, Fuzhou, China.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3542637.3542639

1 INTRODUCTION

Network traffic storage is the base of many applications, including network tracing, network forensics, traffic analysis, etc. For example, network service providers, such as cloud service providers and Internet service providers (ISPs), will store their traffic for advanced applications. Importantly, the network traffic volume is skyrocketing nowadays, which brings huge storage overhead. It is significant to reduce the overhead of network traffic storage.

Existing methods mainly focus on reducing in-network traffic volume to facilitate data transfer. These methods only compress limited parts of packets to maintain data transfer. Some studies proposed removing redundant strings in the packet payload while keeping the protocol header unchanged [1, 2]. In contrast, packet header compression (HC), such as IPv6 header compression, compresses the specific protocol header without modifying the packet payload. However, these in-network traffic reduction methods may not suit traffic storage perfectly. Traffic to be stored does not need to be transferred and can be compressed totally. Lossless data compression is an effective way to reduce data volume. In this paper, we explore the possibility of using lossless data compression to compress traffic. Besides, software-based compression is not cost-effective due to its huge CPU overhead. Many studies proposed using the field programmable gate array (FPGA) to accelerate data compression workloads [3, 4]. Compared to CPUs, FPGA accelerators have lower power consumption and can achieve higher throughput by better parallel pipeline processing.

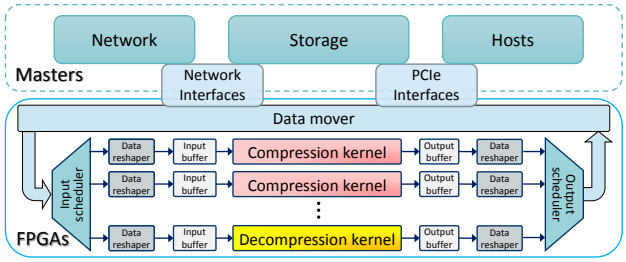

Figure 1: The architecture of our traffic compression system.

In this paper, we propose a hardware-accelerated lossless data compression system to reduce traffic storage overhead. We design an adaptive architecture supporting scalable compression and decompression performance. To optimize hardware accelerator design, we build a resource-performance model for (de)compression kernels. Preliminary evaluation shows the excellent resource efficiency and performance scalability of the proposed system.

2 TRAFFIC COMPRESSION SYSTEM DESIGN

2.1 System Architecture

As shown in Figure 1, the proposed system is designed to support both the network interface and the PCIe interface, which is highly suitable for FPGA-based SmartNICs. The system can compress raw traffic from network and hosts, and store compressed traffic to storage via network or PCIe. Similarly, stored traffic can be decompressed anytime. In our prototype design, (de)compression kernels are based on LZ4, a fast lossless compression algorithm [5]. Note that our system is adaptable to most compression algorithms, and not limited to LZ4. The numbers of compression and decompression kernels can be easily configured to provide various compression and decompression performances. In the hardware architecture, the data mover is responsible for transmitting data between the FPGA accelerator and masters. Input and output schedulers based on the round-robin are designed to schedule the I/O data for multiple kernels. The data path width and the clock frequency of the data mover differ from those of (de)compression kernels. Thus, data reshapers are designed to adjust the data flow width. Data buffers are used for transferring data across clock domains.

2.2 Resource Performance Modeling

The compression algorithm designed for the software platform may not fit the hardware. Besides, algorithm parameter configurations will influence resource consumption and system performance. In

APNet '22, July 1–2, 2022, Fuzhou, China

ACM ISBN 978-1-4503-9748-3/22/07.
https://doi.org/10.1145/3542637.3542639

this paper, we build a performance-resource model to guide the hardware design. It concerns compression ratio, throughput, available resources, the design goal, etc.

LZ4 uses the hash table to search repeated data. The compression ratio performance is determined by the searching capability. It is mainly related to three parameters, including the hash function (H), the hash table size (S_{HT}), and the history buffer size (S_{HB}). H and S_{HT} decide the searching accuracy of the hash table. S_{HB} constrains the searching range. We denote the compression ratio determined by these parameters as $CR(S_{HB}, S_{HT}, H)$.

The system throughput is decided by kernel throughput and the number of integrated kernels. For a system with N_C compression kernels and N_D decompression kernels, its compression throughput (T_C) and decompression throughput (T_D) can achieve

$$\begin{cases} T_C = N_C \cdot P_C \cdot f, \\ T_D = N_D \cdot P_D \cdot f, \end{cases} \tag{1}$$

where P_C and P_D mean the degrees of parallelism of compression and decompression kernels, and f is the design frequency.

The consumed on-chip resources include lookup tables (LUTs), flip flops (FFs), block RAMs, etc. LUTs and FFs consumed by the system can be calculated by

$$\begin{cases} LUT_{sum} = N_C \cdot LUT_C(P_C) + N_D \cdot LUT_D(P_D), \\ FF_{sum} = N_C \cdot FF_C(P_C) + N_D \cdot FF_D(P_D), \end{cases} \tag{2}$$

where $LUT(\cdot)$ and $FF(\cdot)$ are LUTs and FFs consumed by one kernel, denoted as the functions of the degree of parallelism. The block RAM is used in implementing the hash table and the history buffer (decompression kernels only need the history buffer). Its usage can be estimated by

$$RAM_{sum} = N_C \cdot (S_{HB} + S_{HT}) + N_D \cdot S_{HB}. \tag{3}$$

For our prototype system, its design goal is to maximize the system throughput. Besides, there are three design constraints: 1) consumed resources cannot exceed the available resources; 2) the compression ratio cannot be less than the target (CR_0); 3) the ratio (α) of compression throughput to decompression throughput is fixed. According to Equations 1, 2, and 3, the optimized system performance can be obtained:

$$\underset{N_C, N_D, P_C, P_D, S_{HB}, S_{HT}, H, f}{\operatorname{argmax}} (T_C, T_D), \tag{4}$$

subject to

$$\begin{cases} CR(S_{HB}, S_{HT}, H) \geqslant CR_0, \\ LUT_{sum} \leqslant LUT_{available}, \\ FF_{sum} \leqslant FF_{available}, \\ RAM_{sum} \leqslant RAM_{available}, \\ T_C = \alpha \cdot T_D. \end{cases} \tag{5}$$

We fine-tune algorithm parameters and implement kernels based on the performance optimization model. Specifically, we use more LUTs and FFs to increase the degree of parallelism, which will not consume extra block RAMs.

3 PRELIMINARY EVALUATION

We implement our prototype system on the Intel Arria 10 FPGA chip (10AX048H2F34E2SG). As the most-consumed resource, the block RAM becomes the bottleneck of throughput enhancement. We compare the block RAM utilization and memory efficiency with the existing designs with comparable compression ratios. We evaluate our design with the Calgary corpus. As shown in Table 1, our design achieves state-of-the-art memory efficiency. In other words, given the same memory resources, our system can obtain the highest throughput.

Table 1: The block RAM usage, throughput, and the memory efficiency of different acceleration kernels.

	Scheme	Block RAMs (Kbits)	Throughput (MB/s)	RAM efficiency (Kbits per MB/s)
Compression	MLZ4 [4]	2484	228.88	10.853
	Vitis [3]	1908	287.00	6.648
	Ours	1500	536.80	**2.79**
Decompression	MLZ4	720	247.96	2.904
	Vitis	1152	368.00	3.13
	Ours	520	660.76	**0.79**

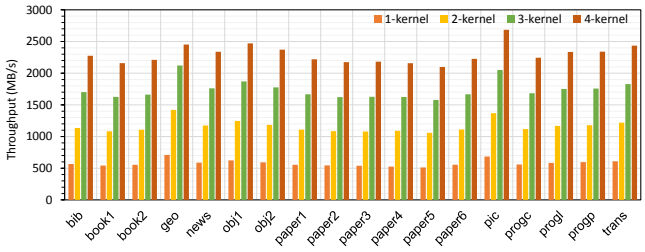

Figure 2: The compression throughput of the prototype system with different number of compression kernels evaluated by various data of the Calgary corpus.

We validate the performance scalability of the proposed design by the end-to-end compression test. We evaluate the end-to-end compression throughput when the number of accelerating kernels changes from 1 to 4. As shown in Figure 2, with 4 kernels, the maximum end-to-end throughput of the proposed design can achieve about 2.7 GB/s (*pic*). Significantly, the end-to-end system throughput scales up well with kernels increasing.

4 CONCLUSION

We propose a hardware-accelerated lossless data compression system to reduce traffic storage overhead. The proposed system is adaptable for various application scenarios. The evaluation results prove its desirable resource efficiency and performance scalability. In the future, we will explore the acceleration of various compression algorithms, improve system compatibility, and design multi-accelerator traffic compression architecture.

REFERENCES

[1] Ahmad Beirami, Mohsen Sardari, and Faramarz Fekri. Packet-level network compression: Realization and scaling of the network-wide benefits. *IEEE/ACM Transactions on Networking*, 24(3):1588–1604, 2016.

[2] Ashok Anand, Vyas Sekar, and Aditya Akella. Smartre: An architecture for coordinated network-wide redundancy elimination. In *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*, SIGCOMM '09, page 87–98, New York, NY, USA, 2009. Association for Computing Machinery.

[3] Xilinx. Vitis data compression library. Website, 2022. https://www.xilinx.com/products/design-tools/vitis/vitis-libraries/vitis-data-compression.html.

[4] Weiqiang Liu, Faqiang Mei, Chenghua Wang, Maire O'Neill, and Earl E Swartzlander. Data compression device based on modified LZ4 algorithm. *IEEE Transactions on Consumer Electronics*, 64(1):110–117, 2018.

[5] Yann Collet. LZ4–Extremely fast compression. Website, 2022. https://github.com/lz4/lz4.