

Load Distribution Method using Multicore Based NIC for High-Performance Computing System

Jae Woo Ahn*, Jong Beom Kim*, Won Seok Choi**, Jong Oh Kim**, Seong Gon Choi*

*Information & Communication Engineering, ChungBuk National University, Cheongju-si Chungcheongbuk-do, Korea

**Fisys Inc., 117 Dunsan-Daero Seo-Gu Daejeon, Korea

jwahni@cbnu.ac.kr, adieu@cbnu.ac.kr, wschoi@fisys.co.kr, jokim@fisys.co.kr, choisg@cbnu.ac.kr

Abstract— Cloud computing is becoming more important in Software Defined Networking (SDN) / Network Function Virtualization (NFV). Virtual Machine (VM) and various services require higher performance servers. Accordingly, the performance of the server also requires high capacity and high performance. As the performance of hardware increases, it is required to increase the speed in the data plane area. In order to increase the speed of the data plane area, it is necessary to adjust the core affinity for each flow. When passing through the kernel, the kernel performs packet header parsing. This is overhead in terms of speed. Therefore we propose a data plane acceleration technology with load distribution of multicore (28 core) based Network Interface Card (NIC) in the test environment that delivers packets at 20 Gbps.

Keywords— Data plane acceleration, Multicore based NIC, By-pass, Core affinity, Tile-Gx.

I. INTRODUCTION

With the increasing importance of SDN / NFV and cloud computing in recent years, high-performance NIC is becoming an essential element of network equipment. As the utilization of VM increases, the processing speed of VM becomes a necessity of network equipment. This changes the hardware performance of server to high capacity and high performance [1].

As hardware performance improves, research is being actively conducted on how to operate hardware more efficiently. As a typical research, there are studies that improve the speed by changing the method of packet processing [2]. Another approach is to estimate load usage by predicting core usage [3]. And there are ways to utilize core affinity [4].

Various studies are underway, but we focus on improving the performance of the data plane. To increase the speed of the data plane, it is necessary to adjust the core affinity for each flow. It means that there is a core in a specific VM that performs its function. When data is transferred to the core that performs other functions in the distribution of the flow to the core, the flow is transmitted back to the core through which the flow is to be processed through communication between the cores. At this point, performance degradation occurs in data processing. Therefore, adjusting the core affinity for each

flow is important. Additionally, when data passes the kernel, the kernel performs packet header parsing which is applied as an overhead in terms of speed. This problem can be solved by passing data directly from the NIC to the core. [2], [5]

Therefore, we propose a method to transfer core affinity by by-pass the kernel by using a multicore-based Network Processor (NP) card to 28 host cores.

II. RELATED WORKS

The communication configuration between TILE-GX36 and the host server is basically as shown in Figure 1. Multicore Programmable Intelligent Packet Engine (mPIPE) packets are processed in Packet Processing Thread and passed to PCIe Thread. Then, the data is transferred to the host server via the transactional I / O (TRIO).

mPIPE is a packet processing engine that can support packet header parsing, packet distribution, packet buffer management, load balancing, L4 checksum, packet collection, and packet transmission. In other words, it is a device corresponding to a network interface card in a general PC or server.

The TRIO interface is a Tile Programmed I/O that provides services to the PCIe controller of the TILE-GX processor to interface the tile processor with the PCI system. It supports Endpoint mode, and root complex mode. The root complex mode is supported only when it is operated as PCIe and it can be supported only as a host when running as StreamIO.

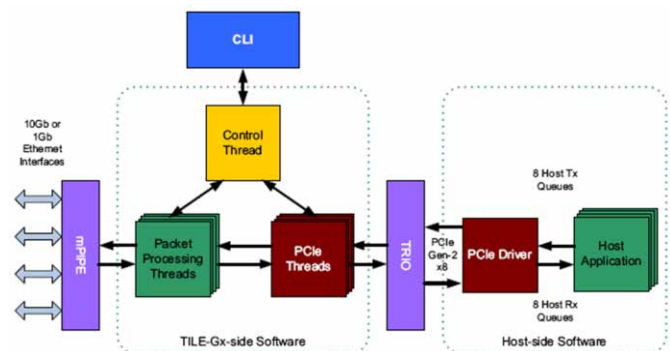


Figure 1. Software Defined NIC Architecture Diagram

TRIO uses Direct Memory Access (DMA). DMA is a method in which a direct memory access controller (DMAC)

takes charge of data transfer instead of a processor to perform high-speed data transfer between a memory and an I/O device. DMA transfers a large amount of data from the physical memory of network card to the PCI address space. And DMA transfers a large amounts of data from the PCI address space to the physical memory to the memory card. If the DMA is occupying the memory bus, the CPU can access data in the main and auxiliary caches. However, data in main memory is temporarily inaccessible. This cycle-stealing reduces the speed of the CPU, but passing input/output operations to the DMA improves overall system performance. It uses the physical address when using DMA, but it also uses Direct Virtual Memory Access (DVMA). With DVMA, it is possible to transfer data between two memories mapped devices with virtual addresses without CPU or memory intervention.

Figure 2 shows the internal operation of Packet Processing threads. The Packet Processing Thread receives incoming data from the mPIPE and performs various packet processing functions. Packet processing functions include passing the packet, dropping the packet, and redirecting to an mPIPE Ethernet port.

In Figure 3, the PCIe thread transfers data from the Packet Processing Threads to the host by DMA. T2H (Tile-to-Host) packet queue and H2T (Host-to-Tile) packet queue. NetQ queuing is an one-to-one mapping between Packet Processing Threads and NetQ queues. NetQ queues and PCIe threads are many-to-one mappings.

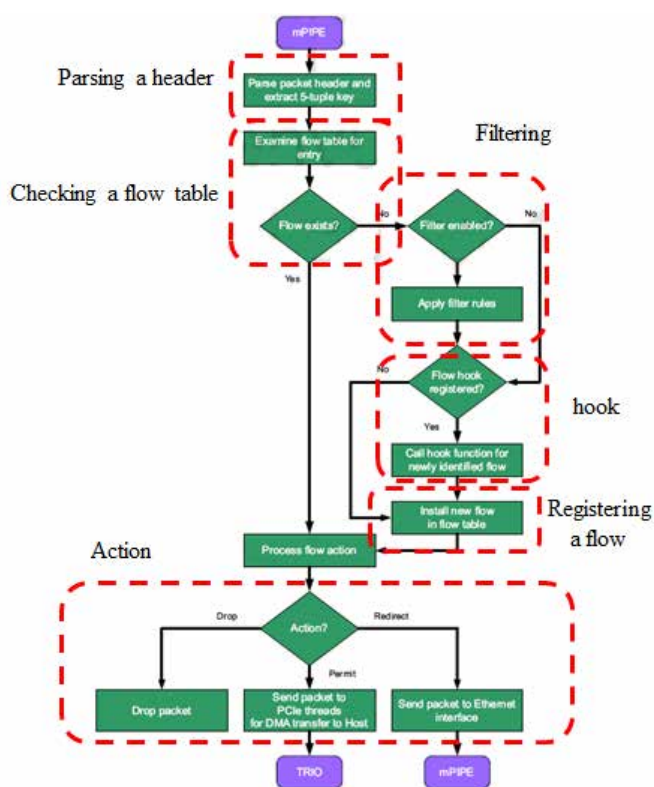


Figure 2. SDNIC packet processing Flow Diagram

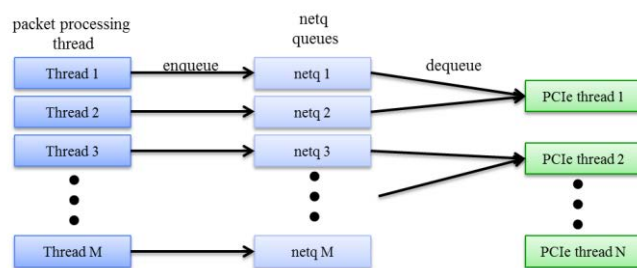


Figure 3. T2H Thread Multiplexing Diagram with NetQ queue

III. LOAD DISTRIBUTION METHOD

In the manual provided by the existing Tiler side, only contents up to 16 cores are covered. In this paper, we propose a method to extend it to 28 cores. Figure 4 shows the processing of data from mPIPE to TRIO in the proposed configuration. The incoming data to mPIPE is stored in memory. In addition, it allows the processing of TRIO from Packet Worker Thread (7 core) to generate and distribute 4 packets first in first out (FIFO) for each Packet Worker Thread. TRIO Driver (28 core) is sent to 28 threads, and each thread is transferred to Host-side PCIe Driver through TRIO. The PCIe Driver uses the channel to identify the queue. Then, the core affinity is matched to the core and the queue is transferred in a 1:1 mapping.

As a result, the process of generating the packet FIFO in the mPIPE Packet Worker Thread must match the core affinity and speed up packet processing.

In the Packet Worker Thread, packet FIFO is generated by checking only 5-tuple information without processing all the packet headers in order to generate efficient packet FIFO. This approach can reduce the consumption of resources consumed in packet processing. And because it creates a packet FIFO based on the core affinity, it can meet the requirements for core affinity. These operations are implemented using the SDNIC control thread. SDNIC provides control plane functionality through the Application Run-Time.

The packet FIFO is transmitted to the host via TRIO. The host server side also uses SDNIC to share T2H and H2T packet memory. And the host application operates in polling mode.

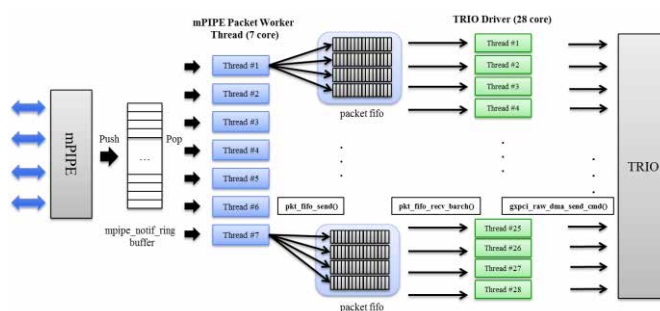


Figure 4. Tile-side architecture of proposed system

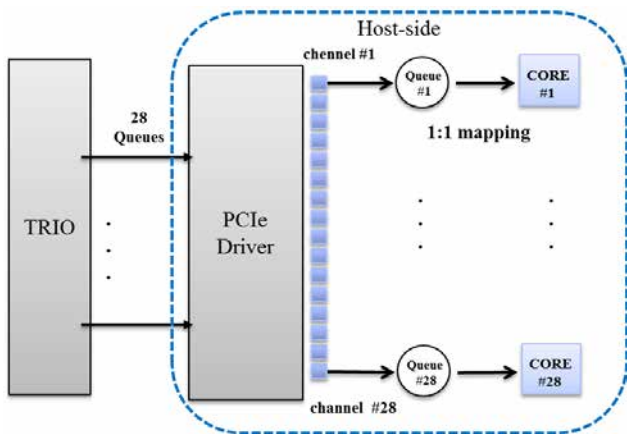


Figure 5. Host-side architecture of proposed system

The basic concept of handshaking for polling mode communication is as follows. The controller indicates its status through the busy bit in the status register. The controller sets the busy bit to 1 when busy and the busy bit changes to 0 when it is ready to accept the next command. In this way, host-side and tile-side communication are achieved.

In other words, packets arriving at mPIPE are transmitted to TRIO in accordance with core affinity from 7 Packet Worker Threads to 28 threads. TRIO transfers to 28 queues to Host Server. It is called by-pass. In the Host Server, NetQ generated according to the core affinity is transmitted through TRIO. When arriving at the Host Server, the threads are separated by channels and distributed to the core by a thread. As a result of distributing according to the core affinity, unnecessary communication between the cores is eliminated and the usage of the CPU can be reduced.

IV. IMPLEMENTATION RESULTS

Figure 6 shows the CPU load test environment. Rx of Xgbe sends 20Gbps of traffic per port (port1 and port2). Traffic generated by the Traffic Generator is sent to the NIC. The specification of the host server is Intel Xeon CPU E5-2650 v2 @ 2.60 GHz, and PCI Express 3.0 is used.

When testing, the Application Run-Time (ART) Framework is used on the Host server. ART Framework designed for high-performance data plane applications. ART provides thread management, memory management network Input/Output, PCIe Input/Output, and command line interface.

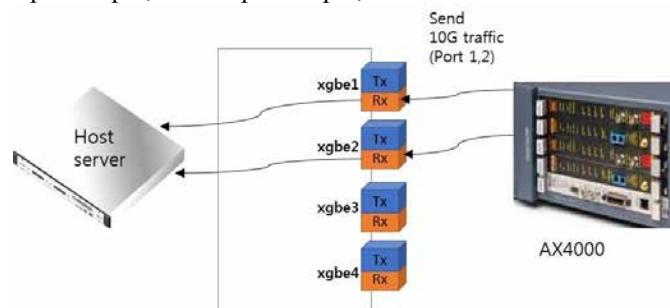


Figure 6. CPU load test environment

Queue	Rx Pkts	Rx Errors	Rx pps	Rx Gbps	Drop Pkts	Drop gbps
0	137430444	0	560605	0.780	0	0.000
1	136905587	0	558455	0.777	0	0.000
2	137696039	0	561589	0.782	0	0.000
3	10937840	0	446076	0.621	0	0.000
4	138048714	0	563115	0.784	0	0.000
5	137580399	0	561149	0.781	0	0.000
6	138106881	0	563476	0.784	0	0.000
7	110221348	0	449597	0.626	0	0.000
8	137595799	0	561277	0.781	0	0.000
9	137640657	0	561616	0.782	0	0.000
10	109984376	0	448488	0.624	0	0.000
11	109846323	0	448101	0.624	0	0.000
12	137218069	0	559636	0.779	0	0.000
13	138045315	0	563045	0.784	0	0.000
14	109858986	0	448123	0.624	0	0.000
15	110099793	0	449194	0.625	0	0.000
16	137610009	0	561250	0.781	0	0.000
17	137678586	0	561534	0.782	0	0.000
18	110096638	0	449023	0.625	0	0.000
19	110242668	0	449492	0.626	0	0.000
20	137544407	0	560965	0.781	0	0.000
21	137188757	0	559671	0.779	0	0.000
22	110295777	0	449839	0.626	0	0.000
23	109859066	0	448198	0.624	0	0.000
24	137594410	0	561275	0.781	0	0.000
25	138006838	0	563036	0.784	0	0.000
26	110210266	0	449482	0.626	0	0.000
27	110634696	0	451314	0.628	0	0.000
Total	3522617688	0	14368635	20.001	0	0.000

Figure 7. Packet transmission rate test result

In the Host Server, use the ART pumping program to match the core affinity. According to the host CPU affinity setting, the interrupt is divided evenly into 28 interrupt requests (IRQs). Then check the usage for each CPU using the TOP program. TOP program is a program that helps you to check information of Linux system and information of running processes in real time.

Figure 7 shows the test results. For each queue, you can check a number of packets coming into Rx, errors, packet per second (PPS), the rate of incoming packets to Rx, the number of dropped packets, and the drop rate. In the Rx Gbps column, it can be estimated that data is transmitted in total 20 Gbps from 28 queues from 0 to 27.

Figure 8 shows the results of the tests for each CPU usage. As shown in Figure 7, when processing core affinity with 28 CPUs of data coming in at 20Gbps, the result is less than 5% of 28 CPUs with a maximum of 3.9% and a minimum of 1.0%.

As a result of the above two tests, we can see that the usages of each core of CPU are less than 5% when the TOP program is monitored while the packet arriving at 20Gbps is directly transferred to the Host server and 28 core pumping is in progress. Therefore, the result shows that the proposed system efficiently distributes high-speed data to each core of the CPU.

top - 14:34:36 up 1 day, 4:52, 4 users, load average: 1.14, 1.00, 0.59									
Tasks: 558 total, 1 running, 558 sleeping, 0 stopped, 0 zombie									
Cpu0	3.2%	0.4%	0.0%	96.4%	0.0%	0.0%	0.0%	0.0%	0.0%
Cpu1	3.2%	0.4%	0.0%	96.4%	0.0%	0.0%	0.0%	0.0%	0.0%
Cpu2	2.8%	0.0%	0.0%	97.2%	0.0%	0.0%	0.0%	0.0%	0.0%
Cpu3	2.3%	1.2%	0.0%	96.2%	0.0%	0.0%	0.0%	0.4%	0.0%
Cpu4	2.4%	0.4%	0.0%	96.8%	0.0%	0.0%	0.0%	0.4%	0.0%
Cpu5	2.0%	0.4%	0.0%	97.7%	0.0%	0.0%	0.0%	0.0%	0.0%
Cpu6	2.0%	0.8%	0.0%	97.3%	0.0%	0.0%	0.0%	0.0%	0.0%
Cpu7	1.9%	0.8%	0.0%	97.3%	0.0%	0.0%	0.0%	0.0%	0.0%
Cpu8	1.6%	0.0%	0.0%	98.4%	0.0%	0.0%	0.0%	0.0%	0.0%
Cpu9	1.6%	0.0%	0.0%	98.4%	0.0%	0.0%	0.0%	0.0%	0.0%
Cpu10	1.0%	0.0%	0.0%	98.0%	0.0%	0.0%	0.0%	0.0%	0.0%
Cpu11	1.0%	0.5%	0.0%	98.4%	0.0%	0.0%	0.0%	0.0%	0.0%
Cpu12	2.2%	0.0%	0.0%	97.8%	0.0%	0.0%	0.0%	0.0%	0.0%
Cpu13	1.1%	0.5%	0.0%	98.4%	0.0%	0.0%	0.0%	0.0%	0.0%
Cpu14	1.0%	1.0%	0.0%	98.0%	0.0%	0.0%	0.0%	0.0%	0.0%
Cpu15	1.4%	0.0%	0.0%	97.6%	0.0%	0.0%	0.0%	0.0%	0.0%
Cpu16	3.5%	0.8%	0.0%	95.7%	0.0%	0.0%	0.0%	0.0%	0.0%
Cpu17	3.9%	0.8%	0.0%	95.3%	0.0%	0.0%	0.0%	0.0%	0.0%
Cpu18	3.1%	0.8%	0.0%	96.2%	0.0%	0.0%	0.0%	0.0%	0.0%
Cpu19	3.4%	0.4%	0.0%	96.3%	0.0%	0.0%	0.0%	0.0%	0.0%
Cpu20	3.5%	0.8%	0.0%	95.7%	0.0%	0.0%	0.0%	0.0%	0.0%
Cpu21	3.9%	1.2%	0.0%	95.0%	0.0%	0.0%	0.0%	0.0%	0.0%
Cpu22	3.4%	0.4%	0.0%	96.2%	0.0%	0.0%	0.0%	0.0%	0.0%
Cpu23	3.0%	0.8%	0.0%	96.2%	0.0%	0.0%	0.0%	0.0%	0.0%
Cpu24	1.1%	0.5%	0.0%	97.8%	0.0%	0.0%	0.0%	0.5%	0.0%
Cpu25	1.7%	0.0%	0.0%	98.3%	0.0%	0.0%	0.0%	0.0%	0.0%
Cpu26	1.5%	0.0%	0.0%	97.3%	0.0%	0.0%	0.0%	0.0%	0.0%
Cpu27	1.5%	1.0%	0.0%	97.3%	0.0%	0.0%	0.0%	0.0%	0.0%
Cpu28	1.0%	0.0%	0.0%	100.0%	0.0%	0.0%	0.0%	0.0%	0.0%
Cpu29	0.0%	0.0%	0.0%	100.0%	0.0%	0.0%	0.0%	0.0%	0.0%
Cpu30	0.0%	0.0%	0.0%	100.0%	0.0%	0.0%	0.0%	0.0%	0.0%
Cpu31	0.0%	0.0%	0.0%	100.0%	0.0%	0.0%	0.0%	0.0%	0.0%
Mem: 7951652k total, 4920820k used, 3030832k free, 3773732k buffers									
Swap: 3964920k total, 0k used, 3964920k free, 516068k cached									

Figure 8. Usage test result at each core

V. CONCLUSION

This paper proposes a system that efficiently processes packets sent to mPIPE at high speed and transfers the packets to the host in accordance with the core affinity, thereby reducing the CPU usage of the host server and efficiently managing the resources of the server. We changed the way to migrate to TRIO from the existing mPIPE Packet Worker Thread and actually tested the CPU usage on the host.

The testing result shows that the CPU of Host server usage is kept at less than 5% in a test environment that delivers packets at 20 Gbps. This confirms that it is an efficient packet transmission method by by-pass with the core affinity.

There is a continuing need for research on changing networking methods and the resulting networking equipment. Continued research is needed on an efficient way to process more packets with more cores at a faster rate.

ACKNOWLEDGMENT

This work was supported by “Human Resources Program in Energy Technology” of the Korea Institute of Energy Technology Evaluation and Planning (KETEP), granted financial resource from the Ministry of Trade Industry & Energy, Republic of Korea. (No. 20164030201330)

This research was supported by the MOTIE (Ministry of Trade, Industry & Energy) and the KIAT (Korea Institute for Advancement of Technology) through N0001132.

*corresponding author is S.G. Choi (choisg@cbnu.ac.kr)

REFERENCES

- [1] I. Enesi, E. Zanj, S. Kokonozi, B. Zanj, “Performance Evaluation Of Statefull Load Balancing In Predicted Time Intervals And CPU”, Smart Technologies, IEEE EUROCON 2017 17th International Conference.
- [2] N. Hanfod, V. Ahuja, M. Fareens, D. Ghosal, M. Balman, E. Pouyoul, B. Tierney, “Analysis of the Effect of Core Affinity on High-Throughput Flows”, Network-Aware Data Management (NDM), 2014 Fourth International Workshop.
- [3] K. Suksomboon, M. Fukushima, S. Okamoto, M. Hayashi, “A Dilated-CPU-Consumption-Based Performance Prediction for Multi-Core Software Routers”, NetSoft Conference and Workshops (NetSoft), 2016 IEEE
- [4] J. Ye, S. Li, T. Chen, M. Wu, L. Liu, “Core affinity Code Block Schedule to Reduce Inter-Core Data Synchronization of SpMT”, High Performance Computing and Communications, 2014 IEEE 6th Intl Symp on Cyberspace Safety and Security, 2014 IEEE 11th Intl Conf on Embedded Software and Syst (HPCC,CSS,ICSS), 2014 IEEE Intl Conf.
- [5] J. Zhang, X. Lu and D. K. Panda, “High-Performance Virtual Machine Migration Framework for MPI Applications on SR-IOV enabled InfiniBand Clusters”, Parallel and Distributed Processing Symposium (IPDPS), 2017 IEEE International.



Jae Woo Ahn received B.S. degree in the College of Electrical & Computer Engineering, Chungbuk National University, Korea in 2017. He is currently a M.S. candidate in School of Electrical & Computer Engineering, Chungbuk National University. His research interest is network programming.



Jong Beom Kim received B.S. degree in the College of Electrical & Computer Engineering, Chungbuk National University, Korea in 2017. He is currently a M.S. candidate in School of Electrical & Computer Engineering, Chungbuk National University. His research interest is network programming.



Won Seok Choi received B.S. and Ph.D. degree in the College of Electrical and Computer Engineering, Chungbuk National University, Korea in 2008 and 2014 respectively. He is Senior Researcher in Fisys Inc. His research interests include Energy saving network, SDN, NFV and NGN.



J Jong Oh Kim received B.S. and M.S. degree in Electronics Engineering from Kyeongbuk National University in 1990 and 1992 respectively. He is currently CEO in Fisys Inc. His research interests are SDN, NFV.



Seong Gon Choi received B.S. degree in Electronics Engineering from Kyeongbuk National University in 1990, and M.S. and Ph.D. degree from KAIST, Korea in 1999 and 2004, respectively. He is currently a professor in the College of Electrical & Computer Engineering, Chungbuk National University. His research interests include smart grid, IoT, mobile communication, high-speed network architecture and protocol.