ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-MOPHA044

DEVELOPMENT OF ETHERNET BASED REAL-TIME APPLICATIONS IN LINUX USING DPDK

G. Gaio†, G. Scalamera, Elettra-Sincrotrone Trieste S.C.p.A., Trieste, Italy

Abstract

In the last decade Ethernet has become the most popular way to interface hardware devices and instruments to the control system. Lower cost per connection, reuse of existing network infrastructures, very high data rates, good noise rejection over long cables and finally an easier maintainability of the software in the long term are the main reasons of its success. In addition, the need of low latency systems of the High Frequency Trading community has boosted the development of new strategies, such as CPU isolation, to run real-time applications in plain Linux with a determinism of the order of microseconds. DPDK (Data Plane Development Kit), an open source software solution mainly sponsored by Intel, addresses the request of high determinism over Ethernet by bypassing the network stack of Linux and providing a more friendly framework to develop tasks which are even able to saturate a 100 Gbit/s connection. Benchmarks regarding the real-time performance and preliminary results of employing DPDK in the acquisition of beam position monitors for the fast orbit feedback of the Elettra storage ring will be presented.

INTRODUCTION

For today's control systems Ethernet is an attractive option that can compete with and often overtake field-bus technologies. Setting up an Ethernet infrastructure is generally easier and less expensive than other communication networks. In the particle accelerator field, suppliers of devices or systems are replacing the conventional interfaces (direct I/O, serial lines, GPIB, etc.) with Ethernet links, especially in high performing instrumentation.

In time-sensitive applications such as feedback systems and, more recently, performance optimization applications using machine learning, it is fundamental to process synchronous data in real-time because their effectiveness scale linearly with the repetition rate. For this reason interfacing Ethernet based devices to the control system in the most efficient way is becoming even more important than in the past.

Real-time Networking

Since 2005 the new front-end computers installed in Elettra and later in FERMI run the GNU/Linux operating system. The kernel versions span from 2.4.25 running on the oldest PowerPC VME boards to the less older 3.14.58 installed on rack-mount servers.

Until the release of RT_PREEMPT on the mainline (2.6.23), the vanilla Linux kernel was unreliable predicting the execution of a task.

† giulio.gaio@elettra.eu

Even today, the network stack is unsuitable for developing time sensitive network applications [1]. The POSIX socket operations (system calls), which transfer control from the application layer to the kernel have significant overheads (e.g. context switch and CPU cache pollution). Moreover in the last fifteen years the network performance has grown faster than the one of the CPUs due the stagnation in the single thread performance. Interrupt moderation techniques try to mitigate the CPU load caused by high-end network interface cards (NIC) but at the cost of increasing the latency.

In order to overcome these limitations, for the FERMI and Elettra front-end computers involved in time critical applications we adopted RTAI and more recently Xenomai, which enable systems to perform real-time tasks. Since the majority of these applications exchange data through Ethernet, the drivers of the on-board NICs were modified to execute the interrupt handler in the RTAI/Xenomai domain and run arbitrary code bypassing the Linux network stack [2].

The downsides of this approach are the development time for patching Ethernet device drivers at every kernel upgrade over different architectures and the complexity of debugging real-time applications that usually run in kernel space. For these reason in the last years we have carried out a campaign to evaluate the real-time capabilities over Ethernet of the latest Linux kernel running in multi-socket servers.

LINUX TUNING

Thanks to the evolution of the Linux kernel, nowadays a system can be easily configured to prioritize low latency over throughput [3]. The most common customizations to perform are:

BIOS

- Enable turbo mode to allow the CPU to reach its maximum clock frequency.
- Disable CPU lower state to avoid the CPU turning to deeper sleep states.
- Disable hyper-threading because the logic cores that share resource with other logic cores can introduce latency.
- Disable virtualization and monitor options because they introduce latency in memory access.

Linux

- Remove a given CPU core from the general kernel Symmetric multiprocessor system (SMP) balancing and scheduler algorithm (*isolcpu*), pin the critical task to the reserved CPU core.

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-MOPHA044

- Inhibit the kernel from sending scheduling-clock interrupt to isolated CPU cores (*nohz_full*)
- Enable *HugePages*; larger but fewer pages are needed. This reduces the number of Translation Lookaside Buffers (TLBs, high speed translation caches) and consequently the time to translate a virtual page address to a physical page address.
- Disable Linux CPU governor which defines the CPU power-saving policies.

Network

- Set the SMP IRQ affinity to match the PCIe routing between CPU sockets and NICs
- Configure the interrupt moderation policy to achieve the lowest latency in acquiring packets from the NIC

Bypassing Linux

Bypassing the Linux network stack and retrieve data from the network board in polling mode has become a common practice. Some software solutions are provided directly by vendors (VMA by Mellanox, OpenLoad by SolarFlare/Xilinx) or by the open source community. The most popular are NetMap, PF_Ring ZC and DPDK [4].

DPDK, initially developed by Intel in 2010, is supported directly by the Linux Foundation and sponsored by market leaders as ARM, Red Hat, AT&T and Ericsson. It is free of charge and supports many vendor cards with a maximum speed of 200 Gbit/s. A large community with many subprojects, an established roadmap, frequent meetings and the presence of the industry, assures a high level of reliability and support in the long term.

DPDK

DPDK is a framework providing Linux with a complete set of functions for receiving, sending and processing Ethernet packets and for assigning memory and CPU resources to real-time applications.

In order to reach the maximum performance in terms of latency and throughput, the user has to assign to DPDK a pool of NICs and CPU cores which become unavailable to the Linux kernel. A special driver maps the NIC PCIe addresses into user space memory so that the process of network card initialization and data processing is performed in a more friendly environment.

The DPDK application, which runs in user-space (see Fig. 1), is usually executed in two steps. The first configures the Environment Abstraction Layer (EAL), which is responsible for gaining access to low-level resources such as hardware (CPU cores and NIC ports) and memory space. In the second part the user code spins on a dedicated core waiting for incoming packets. Once the packets are available, data processing and retransmission can be managed by the same core or executed on other reserved cores. Lockless ring buffers and inter-process communication libraries assure efficient and consistent data transmission between processes running on different CPU cores.

Performance tests are regularly performed by the DPDK team with high-end Intel and Mellanox NICs and the results are publicly available [5]. These documents report the maximum attainable line rate in different configurations but no information is given on latency or jitter performance.

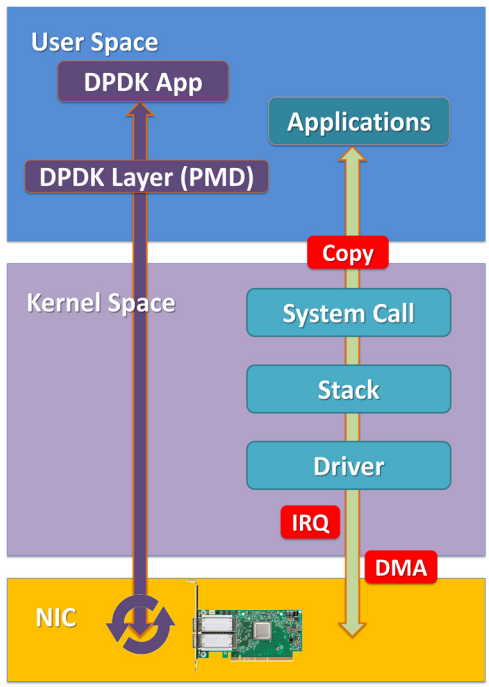

Figure 1: DPDK bypass scheme.

At the Elettra synchrotron light source the most jitter-sensitive real-time application based on Ethernet is the fast orbit feedback [6], a system that corrects the electron beam orbit at 10 KHz using 96 Beam Position Monitors (BPM) and 164 corrector magnets. In order to test DPDK in a real application, the fast orbit feedback network layout has been modified to duplicate the data stream coming out of BPM electronics and forward it to a single rack-mount server. The goal of the test was to validate the mainline Linux kernel together with the DPDK platform as a possible alternative to the present feedback architecture.

Revamping the Fast Orbit Feedback System

The fast orbit feedback is presently composed by twelve VME computers interconnected in a ring topology by means of a reflective memory system using fibre optics. Each of the VME CPUs is connected through a local 1GbE switch to eight Libera Electron BPM detectors [7] sending beam position measurements at 10 kHz rate. Each computer performs one twelfth of the whole calculations consisting in matrix products and digital filtering (1 PID + 8 notch filters centred on some harmonics of the 50 Hz up to 600Hz). The corrections are applied to 164 corrector magnet power converters by means of analog links driven by DAC cards installed on each computer.

A process for replacing the exiting power converters and BPM detectors have started a few years ago. In order

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-MOPHA044

to recover spare parts before the full upgrade of the storage ring [8], some prototypes of the new devices with Ethernet interface (10 GbE for the BPM electronics, 1 GbE for the power converters) will be installed in the existing storage ring.

As the present feedback infrastructure is partially unsuitable to host the new devices, an upgrade of the feedback architecture is ongoing using the new technology based on DPDK.

Test Setup

The twelve original 1GbE local switches have been replaced by four more powerful devices (Extreme X440 G2, 48 port 1GbE + 4 10GbE ports). Every switch is configured to connect the BPMs of three of the twelve storage ring sectors partitioning data in three VLANs.

The traffic of the VLANs is mirrored to one 10GbE port and sent through fibre optic cable in the main server room. An Intel dual socket server (Xeon E5 2637, 3.5 GHz, dual CPU socket, four cores per socket, 16Gbyte RAM, Ubuntu 18.04, Kernel 4.15.0, DPDK 18.02) is connected to the fibre cables by means of four 10GbE ports (Intel X710). The first two 10GbE ports are paired to CPU 1, the other two to CPU 2.

Every 100.2 µs (feedback repetition period) ninety-six FPGAs inside the BPM detectors fire synchronously one UDP packet of 80 bytes that is routed to the server. There is no other load on the switches so the jitter added by the network is assumed to be very low.

The difference between the feedback repetition period and the measured time between the arrival of two consecutive bunches of 24 packets (belonging to one fourth of the BPMs) on a single 10GbE port is a realistic estimation of the jitter per cycle of the system (see Fig.2).

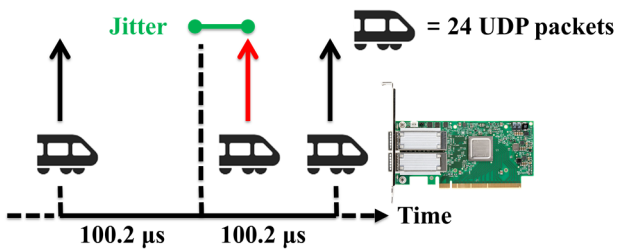

Figure 2: Jitter estimation.

A test program which performs the statistics using a POSIX socket connection or the DPDK layer is executed in four scenarios. Seven of the eight cores are isolated leaving to Linux only the core 0. Every test is run for 100 seconds. In the last two tests core 0 is overloaded with the utility "*stress*", which imposes a configurable amount of CPU, memory, I/O, and disk stress on the system.

- **Test 1**: single process running on core 0 connected to one 10GbE port through a socket, no load on core 0 Linux.
- **Test 2**: single process running on core 2 connected to one 10GbE port through a socket, no load on core 0 Linux.
- **Test 3**: single process running on core 2 connected to one 10GbE port through a socket, maximum load on core 0 Linux.
- **Test 4**: four processes running on four isolated cores (2, 3, 5, 6) connected to four reserved 10GbE ports through DPDK, maximum load on core 0 Linux (see Fig. 3).

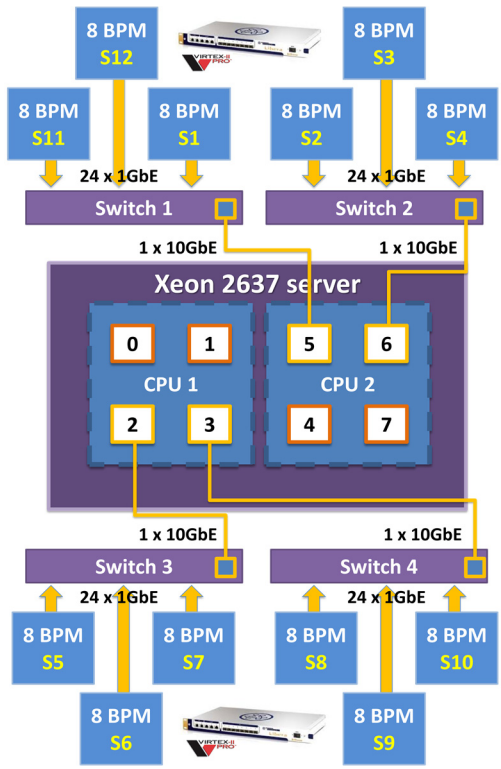

Figure 3: Block diagram of the DPDK test setup in the global orbit feedback system.

Results

The test results are summarized in Table 1. In more than 1 % of the cycles the POSIX socket connection missed the deadline of 100 µs.

In **Test 1** the maximum elapsed time between two consecutive bunches is 7.5 ms with lost packets.

In **Test 2** the maximum elapsed time between two consecutive bunches of packets is 0.8 ms with no packet lost.

In **Test 3** the results are affected by the load added by the program "*stress*" on core 0 reserved to Linux. The maximum elapsed time between two packets is 12.1 ms.

Table 1: Jitter Distribution with Socket Connection

Jitter	Test1	Test2	Test3
< 10 µs	95.1%	91.6%	88.7%
< 50 µs	0.27%	4.8%	6%
< 100 µs	3.2%	2.74%	4.4%
< 200 µs	1.37%	0.81%	0.8%
< 1000 µs	0.0045%	0.0003%	0.04%
> 1000 µs	0.001%	0	0.028%
Lost packet	0.058%	0	0

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-MOPHA044

In **Test 4** the maximum elapsed time between two bunches of packets is below 40 µs in all cores (see Table 2). The processes on core 2 and 3 are affected by more jitter with respect to core 5 and 6. The cache pollution done by "stress" on core 0 (cores 0, 1, 2 and 3 are on the same CPU socket) is the most probable cause.

Table 2: Jitter Distribution with DPDK

Jitter	Core 2	Core 3	Core 5	Core 6
< 1 µs	0	0	0	0
< 2 µs	99.91%	99.88%	99.98%	99.98%
< 3 µs	0.046%	0.05%	0.015%	0.013%
< 4 µs	0	0%	0	0
< 5 µs	0.032%	0.0598%	0.0018%	0.0023%
< 10 µs	0.026%	0.0038%	0.0004%	0.0006%
< 20 µs	0	0	0	0
< 30 µs	0	0.0006%	0.0002%	0.0002%
< 40 µs	0.006%	0	0	0
> 40 µs	0	0	0	0

CONCLUSIONS

Nowadays hard real-time performance over Ethernet in Linux can be achieved easier than in the past. The development time for bypassing the Linux network stack is no more worth thanks to a number of available production-quality vendor-neutral software platforms. The first tests of employing DPDK for an alternative fast orbit feedback architecture at Elettra is promising. The next step will be the integration into DPDK of the code for the feedback calculations performed with AVX-512 instructions.

REFERENCES

[1] T. Høiland-Jørgensen *et al.*, "The eXpress data path: fast programmable packet processing in the operating system kernel", in *Proc. CoNEXT '18*, Heraklion, Greece, Dec. 2018, pp. 54-66. doi: 10.1145/3281411.3281443

[2] L. Pivetta, G. Gaio, R. Passuello, and G. Scalamera, "The FERMI@Elettra Distributed Real-time Framework", in *Proc. ICALEPCS'11*, Grenoble, France, Oct. 2011, paper THDAUST03, pp. 1267-1270.

[3] Low Latency Performance Tuning for Red Hat Enterprise Linux 7, https://access.redhat.com/sites/default/files/attachments/201501-perf-brief-low-latency-tuning-rhel7-v1.1.pdf

[4] S. Gallenmüller *et al.*, "Comparison of frameworks for high-performance packet IO", in *Proc. ACM/IEEE 2015*, Oakland, CA, USA, 2015. doi:10.18429 10.1109/ANCS.2015.7110118

[5] NIC's Performance Report with DPDK, http://fast.dpdk.org/doc/perf

[6] M. Lonza, D. Bulfone, R. De Monte, V. Forchi, and G. Gaio, "Status of the ELETTRA Global Orbit Feedback Project", in *Proc. EPAC'06*, Edinburgh, UK, Jun. 2006, paper THPCH091, pp. 3003-3005.

[7] Instrumentation Technologies, http://i-tech.si

[8] E. Karantzoulis, A. Carniel, R. De Monte, S. Krecic, and C. P. Pasotti, "Status of Elettra and Future Upgrades", in *Proc. IPAC'18*, Vancouver, Canada, Apr.-May 2018, pp. 4054-4056. doi:10.18429/JACoW-IPAC2018-THPMF010