Custom UDP-Based Transport Protocol Implementation over DPDK

Dmytro Syzov, Dmitry Kachan, Kirill Karpov, Nikolai Mareev and Eduard Siemens
Future Internet Lab Anhalt, Anhalt University of Applied Sciences, Bernburger Str. 57, 06366 Köthen, Germany
{dmytro.syzov, dmitry.kachan, kirill.karpov, nikolai.mareev, eduard.siemens}@hs-anhalt.de

Keywords: High-Speed Data Transport, Packet Processing, User Space.

Abstract: As industry of information technologies evolves, demand for high speed data transmission steadily increases. The need in it can be found in variety of different industries – from entertainment (trend for increasing resolution of video-cast for example) to scientific research. However, there are several problems that hinder network application capabilities. One of them is slow packet processing due to significant overheads on system calls for simple network operations. There are hardware solutions, but from the economical point of view, using legacy equimpent is preferable due to high cost of updating network infrastructure. Thus, software solutions to these problems can be preferable. One of them is DPDK toolset which gives the ability to tailor network operations to the application. RMDT is a custom transport protocol aimed at high speed data transmission over lossy networks with high latency. The protocol is built over standard Linux UDP sockets. Thus it is heavily reliant on the networking stack performance. The goal of this work is to improve RMDT performance by means of DPDK in a 10G network and to assess the benefits of such an implementation.

1 INTRODUCTION

The nature of network operations on Linux OS, with overheads on system calls, several memory copies during recv() and send(), results in a low performance in cases of high speed connections. While there are multiple solutions to increasing packet processing rate, for example Receive Packet Steering [1], but they are usually aimed at TCP optimization, or maintaining many low-rate connections. If there is a need in high speed transmission and TCP is not fitting for cases of high latency and lossy network, out-of-the-box options are limited. Their functionality is also often dependent on a specific implementation by the manufacturer, thus making development of a widely applicable network utilities more expensive and harder to maintain.

DPDK [2] toolset aims at boosting packet processing performance by giving developers access to a low-level management of network stack. One of the main benefits is the avoidance of user space to kernel space switches. However it does not provide transmission protocols to use out-of-the-box.

The common bottleneck is receive performance, as in case of standard Linux network operations, packets have to go through multiple memory copy operations and additional management operations necessary for the correct delivery to applications. In case of DPDK, there is an opportunity to tailor these operations specifically to the application. Having more control over timings of various send- and receive- related operations can improve latency, improving performance in use cases such as streaming. Also such control can deliver more precise measurements of round trip time, which consequently can improve behavior of congestion control as standard kernel method can introduce fluctuations in the overall time of an operation.

This work attempts to adapt internal structure of the RMDT [3] protocol to DPDK library and to asses the benefits of DPDK over standard Linux approach. At this stage, the goal is to create a simple RMDT over DPDK implementation to test the possibility of improving its' perormance with DPDK. As the main measure of the efficiency in our tests we are using the achievable data rate. Comparison between synthetic packet generation tests and RMDT tests can show the difference in ratio of time spent on network operations to time spent on custom protocol functionality, allowing an assessment of the necessity to improve the implementation. Thus a simple test of a clean send and receive is to be performed as well.

2 RELATED WORK

As DPDK is a generally applicable network development kit, there is a large amount of projects implementing DPDK for a variety of goals. These include using DPDK to build a light-weight TCP/IP stack to achieve better efficiency with resource limited systems [4] as presented in a paper by R. Rajesh et al., building a high performance software router [5] as presented in a paper by Z. Li. M. Miao et al. developed and tested a self-tuning packet I/O aimed at dynamic optimization of data rate and latency by controlling a batch size in a high throughput network system [6]. As can be seen, an improvement in networking operations is in demand by different types of applications.

3 TESTBED DESCRIPTION

All tests have been performed in 10 GE Laboratory of Future Internet Lab Anhalt [3]. The core element here is the WAN emulator Netropy 10G [7] that can be used to create an emulation of WAN links with various impairments like packet losees, delay, reordering etc. It collects data regarding data passed through it and is used in this work to assess the resulting performance.

Servers, which are used in tests have following characteristics:

- Kernel: 4.15.0-45-lowlatency.
- NIC: 82599ES 10-Gigabit SFI/SFP+ by Intel Corporation.
- Memory: 64 GB DDR3.
- CPU: 2xIntel Xeon E5-2643 v4, 3.40GHz.

Software consists of two RMDT builds and two synthetic tests with pure packet generation and reception. Builds are for standard Linux networking stack and DPDK respectively. For an interface to UDP over DPDK, an already existing software was used – F-Stack [8].

4 SOFTWARE DESIGN

On Figure 1 the flow chart of a basic DPDK receiver functionality test is presented.

Here, the overall loop includes a basic, F-Stack provided, polling interface, which is derived from DPDK's own polling mechanisms. Apart from basic functionality, necessary for receiving packets via DPDK, additional checks are added to assure that data is received correctly. This functionality is put in the "Corruption check" box. A test for a sender is the same, but without polling for EPOLLIN.

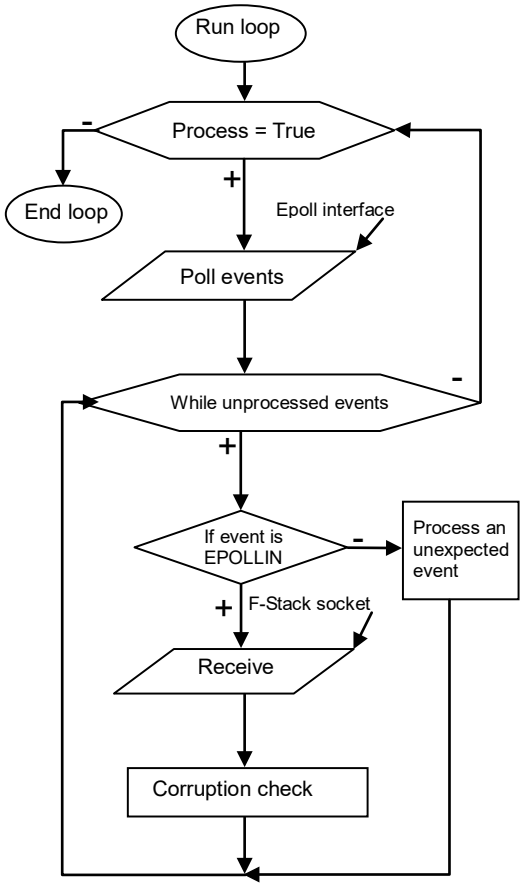

Figure 1: DPDK receive loop.

On Figure 2 the basic RMDT structure for the receiver side is presented.

Here, "Receive handler" is tasked with receive and some basic processing for both user data and service packets. Rest of the protocol functionality is put into "Transport control functionality" box. That includes tasks regarding sending service packets. However, sender functionality is not the aim of this work as it does not bottleneck RMDTs' overall performance in a point-to-point configuration with MTU of 1500 bytes (Ethernet standard [9]) and F-Stack is not optimized for the send process. Both parts work concurrently with the memory buffer and all of the stack is controlled by a master thread which provides protocol interface to an application. It shall be noted that to perform network operations a context switch from user space to kernel space has to be performed, which is one of the contention points.

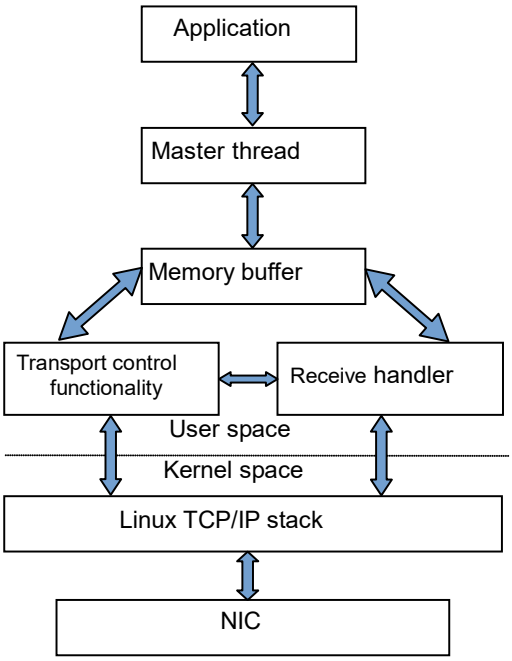

Figure 2: Simplified RMDT structure.

In order to implement F-Stack into RMDT protocol some changes to the protocols' networking subsystem have to be made. F-Stack requires a separate loop function to be run on a dedicated CPU core and that function has to be static. Thus, due to OOP structure of RMDT, all functionality regarding receiving and sending packets has to be moved to a separate thread that is not a direct part of any class in RMDT stack (a global static function).

In the modified structure, additional blocks for send and receive loops represent separate threads which have to run on dedicated cores and perform receive polling and sending via DPDK (Figure 3). These threads are separated from the overall RMDT structure and transmit received data via Single-Producer/Single-Consumer queues, while threads that were handling network operations previously are now polling said queues. Receive/send loop flow is similar to the one presented in figure 1, but with addition of interprocess communication after receiving or before sendig of each batch of packets. Here, switch to kernel space is not needed as DPDK works fully in user space.

5 TEST RESULTS

Firstly, basic DPDK tests have been performed without RMDT to assess the capabilities of hardware while working with DPDK. At this stage both pure

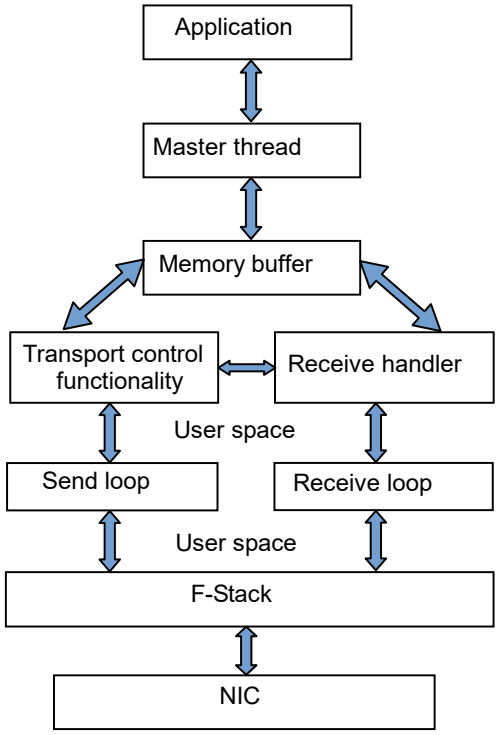

Figure 3: RMDT over DPDK structure.

send and pure receive in a case of point-to-point data transmission have been tested. In the simplest configuration, presented in the previous section, sender was able to achieve up to 6 Gbps, while receiver was capable of achieving maximum link capacity of 10 Gbps (unlike standard TCP/IP, which bottlenecked at receive). However, during testing, certain fluctuations in performance were noticed with sending data, when the rate would drop to 5.2 Gbps or less frequently vary between 5.2 and 6 Gbps. Possible reason for this could be an additional memory copy operations in F-Stack sending interface. The exact cause for such behavior was not studied in this work. Further tests with RMDT were performed only for a DPDK-based receiver. Sender used the standard Linux TCP/IP stack as in multithreaded configuration it was able to achieve 10 Gbps rates, unlike the F-Stack/DPDK test.

Subsequent tests with RMDT were performed – at first with a standard TCP/IP stack to compare it with a DPDK-based RMDT. Standard RMDT showed datarate of 6 Gbps. The bottleneck in such configuration is the receiver as in a test in point-to-multipoint configuration with two receivers, 10 Gbps datarate was achieved. In a test with DPDK-based RMDT, peack achieved datarate was 8 Gbps, althought it was observed to behave inconsistently, sometimes dropping to 6 Gbps. This behavior can be

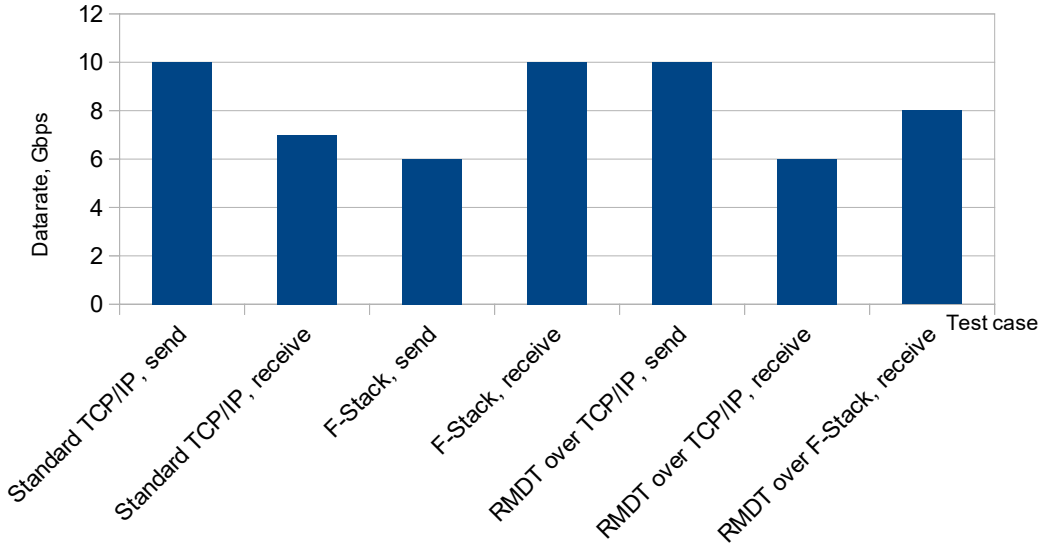

Figure 4: Test results.

explained by unoptimized inter-process communication between F-Stack loop and main RMDT threads. This can be observed by comparison of a clean DPDK test. One of the main reasons are additional memory copy operations. However, even in an unoptimized state, the increase in performance can be seen. A summary of the test results can be seen on Figure 4.

6 CONCLUSIONS

The demand for tools providing high speed data transmission grows, thus leading to development of new SDKs that revise outdated approaches to network applications as for example DPDK/F-Stack does. In this work an attempt to modify a custom UDP-based transport protocol to use DPDK capabilities was made with a goal of increasing performance in a 10G network in a point-to-point configuration with 1500 bytes MTU.

Tests showed an increase in performance in copmarison to standard Linux TCP/IP stack, but full link utilization was not achieved due to the fact that current RMDT structure does not yet fully use DPDK capabilities.

7 FUTURE WORK

In order to continue tests with RMDT over DPDK, significant changes have to be made to the protocol's structure. In particular, better memory management should be implemented. With an improved version additional tests in a 10G and 40G network could be made.

Another possible continuation of this work is developing and testing transport-related applications that could use DPDK functionality for better performance. Network probing algorithms, for example, might improve with lower latency and more stable measurements.

ACKNOWLEDGMENTS

This work has been funded by Volkswagen Foundation for trilateral partnership between scholars and scientists from Ukraine, Russia and Germany within the project CloudBDT: Algorithms and Methods for Big Data Transport in Cloud Environments.

REFERENCES

[1] "Scaling in the Linux Networking Stack", kernel.org, 2018 [Online]. Available: https://www.kernel.org/doc/Documentation/networking/scaling.txt, Accessed on: Dec 01, 2018.

[2] "Data plane development kit", dpdk.org, 2018 [Online]. Available: https://www.dpdk.org/about/, Accessed on: Dec 01, 2018.

[3] "Big Data Transmission | F I L A", fila-lab.de, 2018 [Online]. Available: https://fila-lab.de/index.php/our-work/big-data-transmission/, Accessed on: Dec 01, 2018.

[4] R. Rajesh, K. B. Ramia, and M. Kulkarni, "Integration of LwIP stack over Intel (R) DPDK for high throughput packet delivery to applications," in 2014 Fifth International Symposium on Electronic System Design, 2014, pp. 130-134.

[5] Z. Li, "HPSRouter: A high performance software router based on DPDK," in 2018 20th International Conference on Advanced Communication Technology (ICACT), 2018, pp. 503-506.

[6] M. Miao, W. Cheng, F. Ren, and J. Xie, "Smart batching: A load-sensitive self-tuning packet I/O using dynamic batch sizing," in 2016 IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS), 2016, pp. 726-733.

[7] "Apposite Technologies Netropy WAN Emulators", Apposite Technologies.

[8] "F-Stack | High Performance Network Framework Based On DPDK", f-stack.org, 2018 [Online]. Available: http://www.f-stack.org/, Accessed on: Dec 01, 2018.

[9] C. Hornig, "A standard for the transmission of IP datagrams over ethernet networks," 1984.