Energy efficiency of DPDK

Konstantin Fedorov, Stefan Lachnit*
*Chair of Network Architectures and Services
School of Computation, Information and Technology, Technical University of Munich, Germany
Email: fedo@cit.tum.de, lachnit@net.in.tum.de

***Abstract*—This paper discusses methods for improving the energy efficiency of the Data Plane Development Kit (DPDK), a high-performance software developed by Intel for accelerated processing of network packets. The problems with its energy efficiency and approaches to their solution are also considered. Using DPDK makes it possible to achieve low latency, but the constant polling mechanism uses the CPU completely, which leads to inefficient use of resources. To reduce the load, we looked at various power management methods, as well as the associated implementation difficulties. We have reviewed dynamic voltage and frequency scaling (DVFS), low-power idle states (LPI), adaptive polling, and the application-level thread sleep method. The concept of increasing the flexibility of service cores, which allows for more efficient allocation of tasks, was also considered. In addition, this document discusses energy efficiency issues in the field of network function virtualization (NFV) as well as in 5G networks, where a mechanism such as hardware offload was also considered, which, together with the use of micro-sleep methods, can significantly reduce energy consumption.**

***Index Terms*—user-mode sleep states, NFV, PMD, EAL, DVFS, LPI, adaptive polling**

1. Introduction

The Data Plane Development Kit (DPDK) is a software product originally developed by Intel designed to speed up network packet processing. DPDK was initially created for telecommunications infrastructure, but today it is used in almost all areas, because it can provide high throughput and low latency for data transmission. DPDK is widely used in data centers, cloud systems, 5G networks and other areas where high-speed processing is required. The special feature of DPDK is that it operates in user-space, so it can directly interact with network equipment bypassing the kernel. Instead of the traditional method of processing packets through interrupts, DPDK uses an active polling mechanism called busy polling, in which processor cores constantly check for new data. However, busy polling always keeps threads active, even when there is little or no incoming data. Therefore, even under low network activity, the processor cores remain fully utilized, which leads to overheating of the CPU and reduces its lifespan. In data centers, network traffic usually has the characteristics of a "tidal effect ", meaning the amount of data fluctuates, with periods of high traffic, then low traffic. This behavior causes DPDK usage to result in significant energy losses. [1]

1.1. Important components of DPDK

1.1.1. Environment Abstraction Layer(EAL). EAL is an abstraction layer that provide a universal interface This enables to hide the differences between hardware platforms and OS from applications. [2] EAL is responsible for the DPDK initialization and startup process, allocation and management of low-level resources (memory, timers, etc.). In addition, DPDK provides mechanisms for binding cores to applications. [3]

1.1.2. Poll Mode Driver(PMD). PMD is a network interface driver that runs in user space. It can directly access the Network Interface Card's(NIC's) RX/TX queues. Standard driver implementations use an interrupt mechanism to process packets. This is inefficient because the system is forced to suspend the task at the time of interruption and switch the context, which spends system resources on interrupt maintenance and scheduling instead of completing the task itself. In addition, the system may be overloaded, thus the number of packets that it can process will be limited. By using the polling mode driver, the disadvantages of the interrupt method can be minimized. PMD runs in user space, but it has direct read and write access to the NIC port. This makes it possible to read packets from the network card without interrupting other processes. PMD constantly checks the network interface for incoming data, regardless of whether any packets have actually been received. This way much more packets can be processed, however packet processing applications will always use the CPU resources at full capacity, even if this is not necessary. [3]

1.2. Approaches To Speed Up Data Packet Processing

1.2.1. Zero copy. Many applications are intermediaries through which data is copied. Zero copy is a method that allows you to transfer data bypassing the application, copying data from a file on disk directly to a socket. This way it is possible to reduce the number of context switches between user space and the kernel. Using this method, the kernel copies data from a file on disk to a socket, avoiding copying through the application's memory. Zero copying allows you to transfer data about 65% faster than with the traditional approach. [4]

1.2.2. Busy polling. Busy polling is a feature designed for cases where low latency is necessary. Instead of an interrupt mechanism in which the processor waits for a

 doi: 10.2313/NET-2025-11-3_01

signal from the network card, DPDK uses this feature to continuously poll for new data. One of the main advantages of this approach is to avoid the overhead of system calls between user space and the kernel.

This allows DPDK to eliminate delays associated with interrupt processing, however, CPU cores will always be fully utilized regardless of the amount of work, which leads to excessive power consumption and reduced processor lifespan. [5]

1.2.3. Batch processing. The batch processing method allows processing multiple packets in one polling cycle, optimizing resource usage. It increases throughput and improves routing efficiency. [6] It could amortize the overhead of memory management, improve cache locality and prediction accuracy, which together significantly improves data processing rates and reduces CPU waiting time. [1] It also improves processor utilization by applying the same operations to a group of packets. [6]

2. Evaluation

This section will explore approaches to improve the Energy efficiency of DPDK. DPDK-based systems use various power management techniques that reduce energy consumption and increase energy efficiency. Some of these approaches will be explored below.

2.1. Dynamic Voltage and Frequency Scaling (DVFS)

DVFS is a technology that can be used to adjust the frequency and voltage of the processor depending on the current workload. This helps reduce temperature and power consumption. DVFS prevents system overheating, which avoids hardware damage. Decreasing the frequency increases energy efficiency but also limits the number of instructions executed per unit of time, accordingly reducing performance. [7] Simultaneously reducing both voltage and frequency can be applied for energy savings but also decreases the overall power with which the system operates. [8]

2.2. Low-Power Idle (LPI)

LPI reduces power consumption by disabling CPU subcomponents when there are no active tasks to process. When the system receives new data or requires actions, LPI reactivates previously disabled subcomponents. Because of LPI the CPU can switch to power-saving states (C-states) [9] during idle, thereby significantly reducing power consumption. [7] The use of standard implementations of such methods as LPI and DVFS has the following disadvantages:

1) CPU usage is always at its maximum, so it is impossible to adjust the frequency correctly depending on the load. [1]
2) It has been found that changing the processor frequency affects the packet transmission delay, but the optimal frequency that provides a balance between minimizing latency and power consumption is unknown. [1]
3) Switching to low-power states can take from tens to more than 100 microseconds, and besides, the CPU's exit from this state requires an interrupt, which leads to additional delays. [1]
4) Decreasing the CPU frequency may negatively affect the performance of other applications. [7]

2.2.1. C-states. Modern processors have the ability to transition into power-saving C-states. In the C0 state, the processor executes instructions, while in all other states, it switches to idle mode. The higher the C-state number, the more processor subcomponents will be disabled to save power. Intel processors provide special instructions for entering power-saving states. These commands help to reduce energy consumption by transitioning into deeper C-states. Standard instructions such as MONITOR/MWAIT and PAUSE [10] allow the core to switch to C1 and C6 states, and UMONITOR/UMWAIT and TPAUSE instructions [10] enable the core to enter C0 substates: C0.1 and C0.2. [11] Similarly, AMD processors provide similar MONITORX and MWAITX instructions for entering and exiting these low-power states. [12] The advantage of the C0.1 and C0.2 states is that their exit latency is significantly lower compared to, for example, the C1 state: C1 requires 2.2 more time to exit than C0.1 and about 1.8 more time than C0.2. Another advantage of these new instructions is that they do not require the intervention of the operating system and can be executed at the user-space level. In addition, the energy consumption in these substates is lower compared to the C0 state. Both substates consume about 30% of polling power, and C0.2 state is approximately 15% more power-efficient than C0.1. TPAUSE provides even faster exit delay compared to UMWAIT. TPAUSE is well suited for scenarios in which data arrives at fixed intervals, whereas UMWAIT will be better for situations in which a change in memory location is expected. Tests of the DPDK application with 16 cores showed that without traffic, the system consumes approximately the same amount of power when using C1 or C0.1/C0.2 and it was possible to save about 22% of energy. In a scenario that simulated real conditions at the lowest traffic rate, energy consumption was reduced by 4%. [11]

2.3. Adaptive Polling

Adaptive Polling is a method that allows applications to dynamically adjust the polling frequency of a network interface depending on the current network activity. Its main goal is to minimize the load on the processor during low network activity and maintain high performance during peak load periods. An increase in the polling interval leads to a decrease in the number of CPU cycles used to process a network packet. This increases the efficiency of applications by up to 60% and reduces the power consumption of the cores responsible for data transmission, and also reduces system power consumption by 6.5 watts with a low network load. Experimental data shows, that with a load of 10 Gbit/s, processor cores spend only 50% of the time processing packets, which indicates high optimization. In addition, energy savings reach 20 kWh per year due to lower energy consumption. Using dynamic polling frequency allows you to significantly reduce the

 doi: 10.2313/NET-2025-11-3_01

frequency of surveys. At 1 Gbit/s, the polling frequency is reduced by 34–87 million polls per second, which corresponds to a 97–99.94% reduction. At 10 Gbit/s, the reduction reaches 12–65 million polls per second, which is 90.82–99.28% lower than the standard values. When testing IMIX (Internet MIX) traffic, it was found that processing efficiency is increased by 50%, and the processor load is reduced to 16% of the total operating time. [13]

2.4. Application-Level Thread Sleep Method

The application-level thread sleep method allows a thread to enter a short waiting state if no packet is received after polling. This way, during periods of low activity, processor cycles can be freed up. Functions such as usleep and nanosleep in Linux, or special functions in DPDK (e.g., rte_pause() or rte_delay_us_block()) allow precise control of thread pausing, reducing the waiting time. However, the difficulty of this method lies in the fact that it is hard to determine the optimal "sleep" duration. This is due to the fact, that the duration of the sleep depends on various factors, such as packet size, frequency and overall traffic load. To solve this problem, a dynamic algorithm was developed that evaluates the packet size and transmission rate in real time and also determines the appropriate sleep duration. A modified Kalman filter prediction algorithm was applied to estimate the packet size and transmission rate. This allows for accurate prediction of idle time and reduces the number of empty polling cycles. Experimental results show that using this approach reduces CPU load by more than 80% with a slight decrease in transmission performance. For example, with a packet size of 512 bytes, the transmission speed drops by about 4.3%, and with 768 bytes by about 4.7%. Thus, this model can be applied to various network devices, such as 5G network packet processors, which will significantly reduce power consumption. [1]

2.5. Service Cores

DPDK provides a service cores mechanism that enables dynamic task distribution among CPU cores. This functionality is integrated into EAL, which provides an API that allows applications to manage service cores during runtime.

The DPDK Service core library is a software abstraction that allows you to hide the features of specific hardware or software. An important advantage of service core is that applications can distribute tasks between cores depending on the platform and environment. This way, you can compensate for the differences between the capabilities of different platforms. [14]

The developer can assign a task to a service core, and it will be able to independently determine whether the required hardware is present. If there is no such hardware, then the task will be executed via software. Each service runs on a dedicated service core, although the same core can serve multiple services. They are scheduled in a simple round-robin run-to-completion. If there are too many services on a single core, the processing latency of some services will significantly increase. In the standard DPDK implementation, the distribution of new services across service cores is done manually by the developer. This is a difficult task, and incorrect service assignment can reduce the throughput of individual cores and decrease their efficiency. For example, it is critical for packet processing services to maintain reliable throughput during packet transmission and reception. This creates the need for a way to migrate services between cores so that each can get enough CPU cycles without compromising other services. Using a load balancer allows each service core to theoretically run at about 100% of the CPU load, excluding the overhead incurred when switching between services. There are two strategies for implementing a load balancer: a static load balancer and a dynamic load balancer. [3]

- **Static load balancing.** assumes that the assignment services to cores is predefined (at the compilation stage or earlier). In the current DPDK implementation, developers themselves determine which cores will be used for service tasks and manually distribute services between them. There is a way to automate this process: When running, the DPDK application can scan which services are available and distribute them evenly across the cores.
- **Dynamic load balancing.** offers dynamic assignment of services to cores based on a continuous assessment of which core is best suited to execute a service. If there is a more suitable core, the service should be migrated there. Dynamic load balancing is more appropriate for DPDK, as CPU resource usage is constantly varying. When the load of the CPU decreases, the load balancer can migrate all services to fewer number of cores, and the remaining cores can either be disabled or put into idle state to save energy. A predefined activation threshold of the load balancer (for example, 70% of CPU load) ensures that the cores operate efficiently, preventing excessive peak frequency usage and freeing resources when load decreases. When fewer cores are active, the processor can redistribute energy and increase their clock frequency, which speeds up the performance for tasks requiring high performance in a single thread. [3]

3. Scopes of application

3.1. Energy Efficiency in Virtualized Network Functions (NFV)

Network Function Virtualization (NFV) allows the replacement of specialized hardware with virtual network functions (VNF) running on standard servers. This approach significantly reduces capital and operating costs, simplifying the scaling and maintenance of network infrastructure.

However, with all the advantages of NFV, energy efficiency remains an important issue, especially when using resource-intensive accelerators like DPDK, which often lead to energy overruns.

As part of NFV's energy efficiency research, various software platforms for high-speed network traffic processing compared, such as DPDK-OVS (Open vSwitch

 doi: 10.2313/NET-2025-11-3_01

with DPDK support), Click Modular Router, and Netmap VALE. DPDK demonstrates consistently high power consumption (about 138 watts) regardless of the load level, as it uses PMD, which makes processor cores work at 100% even in the absence of traffic. When processing virtual I/O (for example, when transferring packets via QEMU to virtual machines), DPDK-OVS turns out to be less efficient than other solutions. In addition, when running virtual network functions such as IDS/IPS Snort or Bro, the platform demonstrates lower energy efficiency compared to alternatives, especially when working with large packets and low or variable load. Although DPDK-OVS provides the highest performance and minimal latency, Netmap VALE shows greater potential in environments where power consumption is critical. [15]

3.2. 5G Energy Efficiency and User-plane Functions (UPF)

A network packet is the main component of a network. Micro-sleeps can be applied to all types of packet processing, but they are especially useful in the User Plane Function (UPF). UPF is one of the functions of the network, and it is a separate VNF function. Its main role is to forward packets to and from the Internet. Packets are usually processed by the kernel, but this approach has many disadvantages. System calls and copying packets to and from the kernel space lead to high overhead, which makes it difficult to scale and increase the packet transfer rate. Therefore, the DPDK network interface is used to process packets bypassing the kernel.

However, DPDK has the above-described problems that need to be solved: busy waiting consumes more energy than necessary and always loads the core at 100%, regardless of the load. In addition, DPDK doesn't provide information to make a decision on reducing power consumption. Several improvements have been made to resolve these issues. One of the methods is to switch to an idle state using the UMONITOR or UMWAIT instructions described earlier. This allows server processors to enter these states while waiting for network events. These instructions allow you to reduce power consumption even during very short idle periods than before, and can also be more efficient at higher data transfer speeds. When using user-mode sleep states, you can see the CPU thread load using the measurement functionality.

There is also a Hardware offload method that allows you to transfer processing of processor threads to the NIC. Accordingly, the processor core will be freed up and more CPU cores will have the opportunity to switch to microsleep. HW offload also offers other potential benefits with improved throughput and latency. Laboratory experiments have shown that the processor's power consumption has decreased from 190W to 61W and at maximum load from 190W to 145W. Even under the worst conditions, power-saving methods remain effective. The utilization of the technologies described above has reduced energy consumption by 68%. And with the maximum traffic load, power consumption was reduced by 24%. [12]

4. Conclusion

In this article, various approaches to improving DPDK efficiency were discussed, and special attention was paid to methods for reducing DPDK energy consumption. In particular, mechanisms such as zero copy, busy polling and batch processing were considered, which can significantly increase the speed of packet processing. We have considered methods such as DVFS, LPI and specifically C-states, adaptive polling, as well as the thread sleep method. We have proven that all these methods have had a significant effect on reducing energy consumption. For example, adaptive polling can increase application efficiency by up to 60% and reduce power consumption by up to 6.5 watts, while data processing efficiency can be increased by up to 50%. LPI provides energy savings of 15-30% using special C-states. It was also experimentally found out that the thread sleep method can reduce the load by more than 80% while slightly losing performance. In addition, the service cores method was considered, which allows you to redistribute the load and thereby reduce energy consumption. Also, we reviewed the application of DPDK in various fields such as 5G networks and NFV, and approaches for efficient energy consumption in these areas. The main contribution of this work is a detailed overview of the mechanisms for improving DPDK efficiency.

References

[1] Q. C. Mingjie Wu and J. Wang, "Toward low CPU usage and efficient DPDK communication in a cluster," 2021.

[2] *Improving the flexibility of DPDK Service Cores*, https://doc.dpdk.org/guides/prog_guide/env_abstraction_layer.html.

[3] M. J. Denis Blazevic, "Improving the flexibility of dpdk service cores," 2019.

[4] S. K. Palaniappan and P. B. Nagaraja, "Efficient data transfer through zero copy," 2008.

[5] *power-man*, https://doc.dpdk.org/guides-19.02/prog_guide/power_man.html.

[6] P. Okelmann, F. G. Leonardo Linguaglossa, and G. C. Paul Emmerich, "Adaptive batching for fast packet processing in software routers using machine learning," 2021.

[7] X. Li, T. Z. Wenxue Cheng, and B. Y. Fengyuan Ren, "Towards power efficient high performance packet i/o," 2020.

[8] R. B. Dongsheng Ma, "Enabling power-efficient dvfs operations on silicon," 2010.

[9] "Low-power idle states," https://edc.intel.com/content/www/us/en/design/ipla/software-development-platforms/client/platforms/alder-lake-desktop/12th-generation-intel-core-processors-datasheet-volume-1-of-2/009/low-power-idle-states/.

[10] *Intel® 64 and IA-32 Architectures Software Developer's Manual*, https://cdrdv2.intel.com/v1/dl/getContent/671110.

[11] D. Hunt, R. Pattan, P. Shah, R. Sexton, and C. MacNamara, "Power Management – User Wait Instructions Power Saving for DPDK PMD Polling Workloads," 2023.

[12] P. H. Leif Johansson and R. Skog, "Energy-efficient packet processing in 5g mobile systems," 2022.

[13] H. G. Trifonov, "Traffic-aware adaptive polling mechanism for high performance packet processing," 2017.

[14] *Service cores*, https://doc.dpdk.org/guides/prog_guide/service_cores.html.

[15] Z. Xu, F. Liu, T. Wang, and H. Xu, "Demystifying the Energy Efficiency of Network Function Virtualization," 2016.