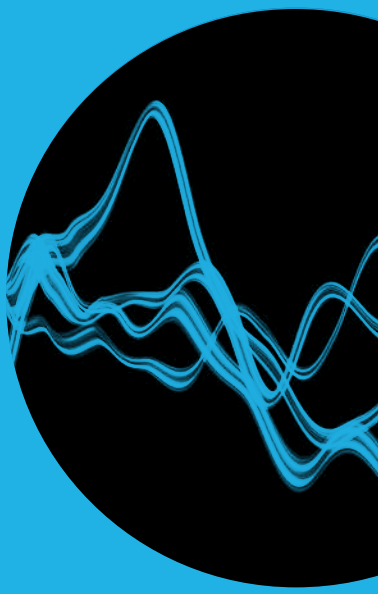
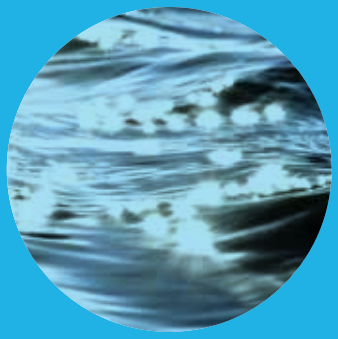
CHARTING THE FUTURE OF INNOVATION | #7·2022

Ericsson Technology Review



ENERGY-EFFICIENT PACKET PROCESSING IN 5G MOBILE SYSTEMS



ERICSSON

Energy-efficient packet processing in 5G mobile systems

Communication service providers around the world are looking for new opportunities to reduce energy consumption in their networks. As a complement to other approaches, we have evaluated a promising new method: applying micro-sleeps in packet processing nodes. Micro-sleeps can be opportunistically applied in all idle periods, thereby saving energy across a wide range of traffic conditions.

LEIF JOHANSSON,
PER HOLMBERG,
ROBERT SKOG

There are several ways to improve the energy efficiency of mobile communication systems and reduce the amount of energy it takes to provide mobile broadband services.

■ As part of its efforts to reduce network energy consumption, a communication service provider (CSP) may choose to upgrade to new hardware (HW) that consumes less energy than the previous generation. It is also possible to cut energy consumption with the help of an advanced scale-in/scale-out mechanism that reduces the amount of HW in use when traffic is low, such as during the night. In these conditions, some servers can be turned off, which results in energy savings.

Regardless of the energy-saving mechanism, however, it is essential to ensure there is no negative impact on real-time characteristics such as jitter and packet latency. It is also important to note that the correlation between Moore's law and power consumption savings for each new HW generation has decreased.

Our latest research indicates that CSPs could save additional energy in their data centers by applying micro-sleeps in packet processing nodes. This approach has both lower overhead and lower latency than existing methods, and makes it possible to reduce power consumption even at high load, without degrading application performance. Because power management is implemented in communications libraries, it is easy to integrate into existing applications.

Packet processing in communication networks

Packet processing in communication networks involves the application of different functions and algorithms to ensure that each packet can run through the network efficiently. It includes steps such as packet identification, inspection and manipulation.

The network packet is a fundamental component in a packet-switched network. It consists of three main parts: the header, the payload and the trailer. The header contains information such as the sender address, the destination address, the length of the packet and the packet sequence number. The payload contains the transferred data – that is, one part of a video, e-mail or other type of content. The trailer marks the end of the packet and can also include error detection and correction information.

The different packet processing functions and algorithms range from fairly simple to more complex ones. A basic routing function is a good example of simple packet processing. More complex packet processing functions involve the application of different policies, charging and manipulation of the packets.

User plane packet processing nodes

While micro-sleeps can be applied to all types of packet processing, our research indicates that they are particularly impactful when used in the user plane function (UPF).

According to its definition in 3GPP, the UPF is a user plane packet processing node in mobile systems that links the RAN to the internet (or similar data networks) [4]. Simply put, the main purpose of the UPF is to forward packets to and from the internet. This is often done in combination with different traffic optimization functions that range from pure Transmission Control Protocol (TCP) optimization to more advanced video optimization in combination with intelligent traffic congestion logic.

> WHILE MICRO-SLEEPS CAN BE APPLIED TO ALL TYPES OF PACKET PROCESSING, THEY ARE PARTICULARLY IMPACTFUL WHEN USED IN THE UPF

The UPF includes several complex processing functions such as GPRS Tunneling Protocol User Plane encapsulation and decapsulation. Other UPF functions include access control, bearer lookup, QoS mapping and marking. It also follows rules for guaranteed bitrate and maximum bitrate. Packets passing through the UPF are also subject to online/offline charging – that is, the application of different charging policies.

Instructions about how to process packets for different user equipment come from the session management function/policy control function. The processing of packets from different users occurs independently for the most part. All the packet processing functions and algorithms must meet the system requirement of real-time characteristics such as jitter and packet latency.

Kernel packet processing

The built-in, native networking support in computers is implemented as part of the operating system (OS) kernel and uses the POSIX (Portable Operating

Terms and abbreviations

AMD – Advanced Micro Devices | **API** – Application Programming Interface | **CPU** – Central Processing Unit | **CSP** – Communication Service Provider | **DPDK** – Data Plane Development Kit | **HW** – Hardware | **NIC** – Network Interface Card | **OS** – Operating System | **RTD** – Round-Trip Delay | **SW** – Software | **UPF** – User Plane Function

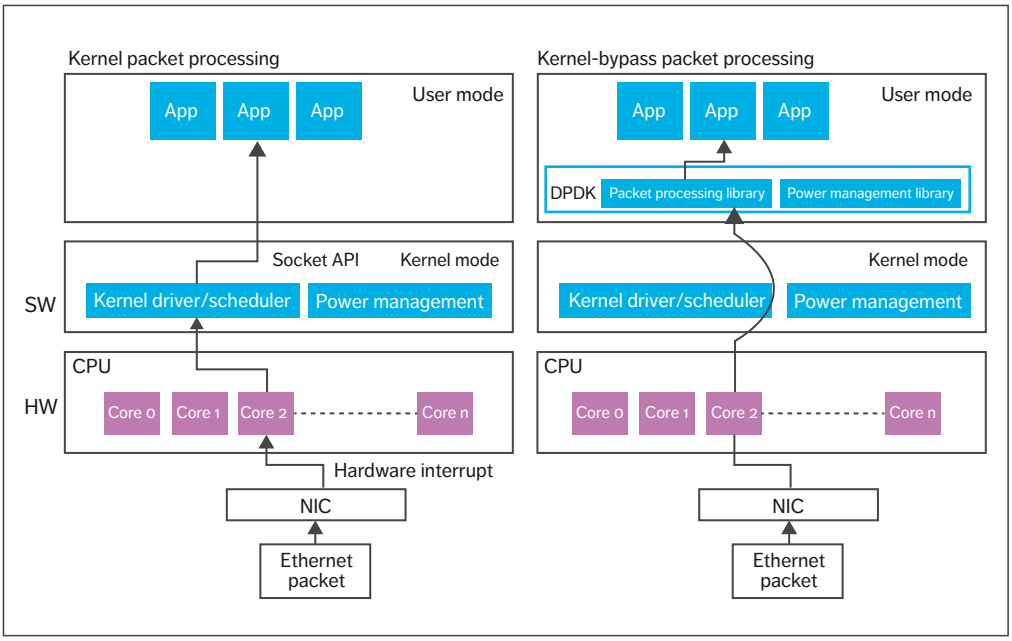


Figure 1 Kernel packet processing versus kernel-bypass packet processing

System Interface) socket as its standard application programming interface (API).

In kernel packet processing, user-mode application programs use the POSIX socket API to send and receive packets, while the kernel driver/scheduler handles the interaction with the network interface card (NIC). If the network is not ready (no packet has arrived) the kernel can decide to block that application while waiting.

The left side of *Figure 1* illustrates how kernel packet processing works, with "App" representing user-mode applications. In this model, the OS is aware of the idle time, and OS power management can work to save power.

Kernel packet processing has multiple disadvantages. Most importantly, the high overhead of the OS calls and the copying of packets to/from the kernel space makes it hard to scale to high networking speeds and high packet rates. For example, the time between 1,534-byte packets on a 200Gbit Ethernet is 61ns. This corresponds to the execution time for performing a system call, which does not leave sufficient time to perform the packet processing itself. The speed of today's network cards makes kernel-packet processing challenging at best, and impossible at worst.

Power management in the kernel can be problematic at high networking speeds. The OS kernel monitors and saves energy based on core utilization. This is a much slower mechanism that does not have the ability to follow the rapid changes in high-speed network traffic, resulting in queue buildups, delays and even packet drops.

Kernel-bypass packet processing

Kernel-bypass packet processing eliminates the kernel execution overhead by moving packet processing directly to the user-space, as shown on the right side of Figure 1. In this scenario, the OS can dedicate a network interface to an application such

as the Data Plane Development Kit (DPDK), which can program the HW from the user-space.

When DPDK is used, packets are received directly in user-space memory without kernel interaction, and all network-related interrupts are disabled. It is up to the application to make sure that packet queues and ring buffers to the NIC are checked frequently for new packets.

To avoid packet drops and reduce packet latency, the DPDK-based application is designed to check the packet ring buffer in busy-waiting mode, where the complete core is assigned to the application thread. This enables packet processing without any context switching or HW interruptions and minimal cache pollution, as the application thread is the only user of the core. Measurements using DPDK as a kernel bypass for packet processing show that millions of packets can be received in a single core and pipelined to other cores for further packet processing.

Kernel-bypass packet processing solves the issue of how to handle packet processing at speeds of 200 Gigabit Ethernet and beyond, but the busy-waiting technique of packet reception comes at a cost. No energy savings will be achieved if all the packet processing cores are 100 percent utilized in busy-waiting mode.

In kernel-bypass packet processing, Ethernet packets are transferred directly to the user-mode application memory. Power management needs to be performed in the user-space within the DPDK library. It is important to note that DPDK interacts with the kernel power management when changing core frequency [1].

> “WHEN THE EVENT OCCURS (OR THE MAXIMUM WAIT TIME EXPIRES) THE PROGRAM SIMPLY WAKES UP AND CONTINUES EXECUTING”

Energy-efficient packet processing using DPDK

Higher data rates in packet processing have made it necessary to change packet processing to use DPDK with dedicated cores, user-mode execution and poll-mode drivers to remove the high overhead from context switches and system calls. Unfortunately, though, this solution is not as energy efficient as modern HW allows it to be. There are three main issues that need to be addressed:

1. The busy-waiting mode consumes more power than necessary when waiting for work.
2. OS power management is slow and cannot utilize idle time between packets and bursts of packets for high-speed interfaces.
3. The busy-waiting mode in DPDK always makes the core appear to be 100 percent utilized, with no information upon which to base power management decisions such as scaling down at periods of low traffic.

To solve these issues, power management actions must be fast, controlled directly from the packet processing application and executed in user mode. Two developments have made this possible. Firstly, server processors now have support for entering power-saving sleep states directly from applications in user mode. Secondly, the latest release of the DPDK library includes an option to utilize these sleep states while waiting for new networking events, making it possible to avoid the problems associated with the busy-waiting loop [1].

User-mode sleep states

Processors have specific instructions for entering sleep states, waiting for events and waking up. Previously these instructions were only available in the OS in supervisor mode, but new versions of these instructions have become available that are safe to use for implementing user-mode micro-sleep states. They are much faster than the old instructions and can be implemented without OS support.

The latest instructions allow a user-mode program to set up an event to monitor (that is, specify the update in the ring buffer that the program will

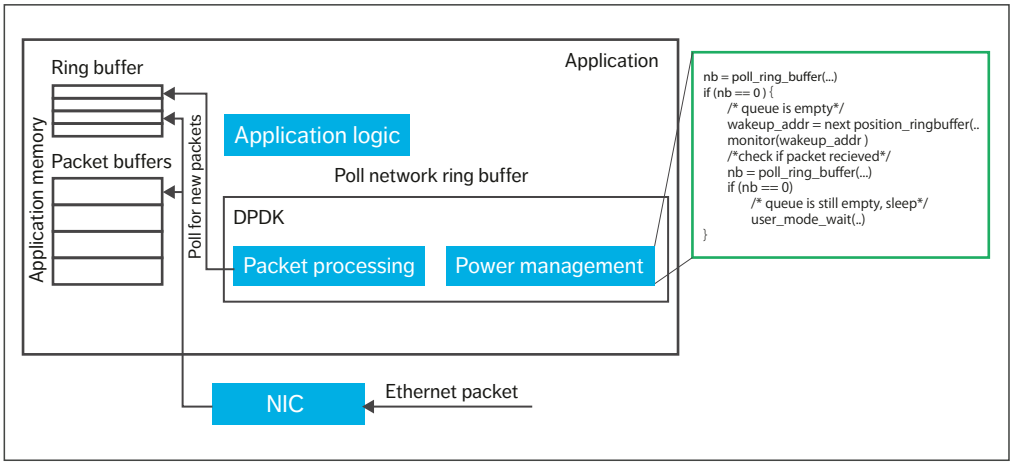


Figure 2 Power management integrated into the DPDK

wait for), define the maximum allowed wait time and then enter sleep state. When the event occurs (or the maximum wait time expires) the program simply wakes up and continues executing. There is no overhead to enter OS kernel, to do context switches and so on.

Examples of such user-mode instructions for setting up an event to monitor and for entering sleep state are UMONITOR and UMWAIT in Intel processors [2] and MONITORX and MWAITX in AMD processors [3]. These instructions make it possible to save energy for much shorter idle periods than was previously possible. They can also be efficient at higher communication speeds.

DPDK integration

Support for the new user-mode sleep-state mechanism has recently been added to the DPDK library, which makes it possible to enter energy-saving sleep states directly in the poll loop doing busy-waiting of DPDK descriptor ring or rings.

If the polls of the descriptor ring (or rings) are empty, an event monitor is set up and a final poll is done before user-mode sleep state begins. Relevant updates – such as an NIC or other central processing unit (CPU) core updates related to the monitored position in the ring – will be detected by processor HW, which will then wake up the CPU core to restart execution of the next instruction, continuing the poll. The poll loop will then detect the update.

Figure 2 illustrates how power management is integrated into the DPDK. In this setup, the network device allocates the ring buffers and packet buffers in the application user-mode memory and updates the ring buffer with the new packet buffer point at packet reception. User-mode sleep state is entered when the ring buffer is empty (that is, the queue to the NIC is empty). Wake-up is triggered when the NIC updates the ring buffer. It is important to note that the proposed power management does not impact the application logic.

Observability

Processors include a measurement functionality for time spent executing and time spent in different sleep states. User-mode sleep states make the CPU thread utilization visible for this measurement functionality, and the CPU thread utilization can also be observed for the DPDK application. CPU thread utilization can also serve as input to the control loops for other types of power management (such as frequency scaling) that may be used as a complement to user-mode sleep states, potentially leading to further improvements in energy efficiency.

Hardware offload

HW offload is a technique that transfers the handling of selected flows from the processor (the slow path) to the NIC (the fast path). This approach frees up the CPU core and makes micro-sleep technology more effective because a larger number of CPU cores have the opportunity to micro-sleep. HW offload also improves the efficiency of the Peripheral Component Interconnect bus and offers other potential benefits with regard to improving throughput, efficiency and latency in the system.

Processor core voltage/frequency scaling

Scaling down CPU core voltage and frequency is a complementary method for achieving the best efficiency over longer periods of lower processing. There is a trade-off with user-mode sleep state, as scaling down also means shorter idle times.

Since power consumption increases proportionally to the square of the voltage, it is favorable to scale down the voltage and frequency when the packet rate drops for a period of time. A fast and reliable monitoring function is needed to detect increased packet rates and, most importantly, packet bursts. The faster and more reliable the detection is, the more aggressively the voltage and frequency can be scaled down without causing negative effects and degrading KPI due to queue buildups that increase latencies, timing jitter and the associated risk of packet drops.

DPDK provides interfaces that enable fast control of CPU core voltage/frequency scaling. In our implementation, we use a fast packet-burst detection in DPDK together with a control loop that maintains a performance margin that handles even the worst-case burst scenarios. When there is a margin, the user-mode wait states offer an alternative method of energy saving, and we can avoid any negative impact on packet processing without compromising energy efficiency.

Processor uncore voltage/frequency scaling

Uncore refers to the parts of the processor outside the actual cores – that is, the interconnection between the CPU cores (and other functions) and the interfaces on the processor that are shared and used by all CPUs. Scaling down the voltage and frequency of the uncore is a complement to energy-saving mechanisms implemented in the core. As in the core, it is favorable to scale down the voltage and frequency in the uncore when the packet rate drops for a period of time.

As the uncore is shared by all CPU cores, frequency needs must be coordinated between all cores and applications, including those outside DPDK, making uncore changes a slower mechanism than core frequency changes.

> “THE RESULTS OF THESE EXPERIMENTS INDICATE THAT THE POWER-SAVING METHODS ARE STABLE AND EFFICIENT”

Experimental validation and results

We ran two sets of lab experiments to validate the user-plane micro-sleep design. In the first set, we generated a workload that we consider to be realistic and applied it to a UPF application. The UPF measurements showed that processor energy consumption decreased from 190W to 61W at low traffic load and from 190W to 145W at maximum traffic load. Beyond quantifying the potential energy savings in a realistic use case, our results verified that power-saving methods work well with existing communication applications.

Our second set of experiments used synthetic workloads that were designed to simulate extreme conditions. The results of these experiments indicate that the power-saving methods are stable and efficient, with negligible impact even in worst-case conditions.

Energy-efficient packet processing applied on a user plane function

In the first set of experiments, we applied the DPDK-based power management with micro-sleep and

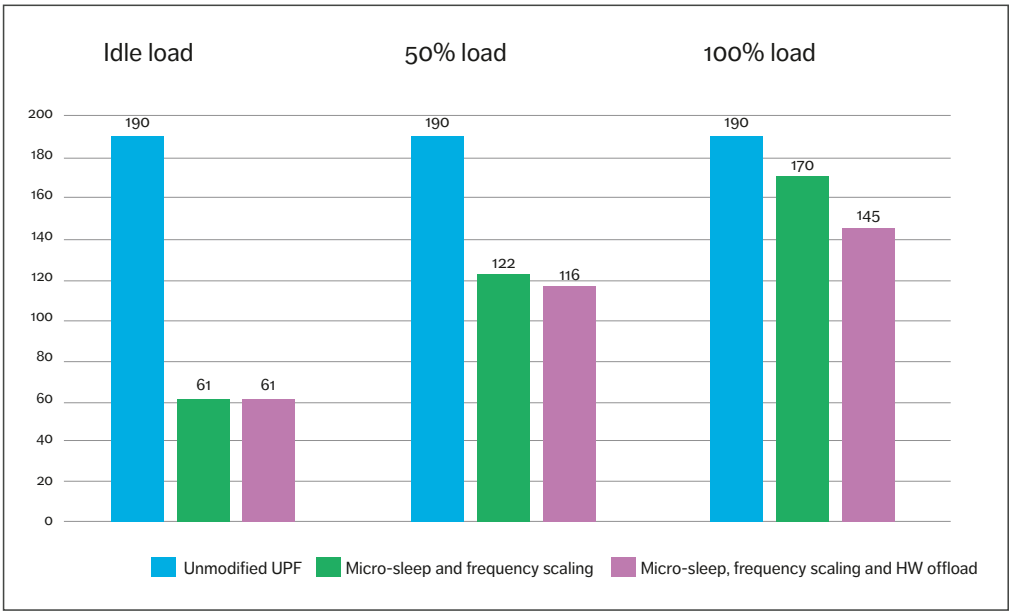


Figure 3 CPU power consumption at idle load, at 50 percent maximum traffic load and at 100 percent maximum traffic load

frequency scaling on an Ericsson UPF application. To measure the energy gain when using HW offload, the UPF is designed to off-load packet flows with the highest bandwidth to the network device.

As a traffic model, we used a traffic mix from our internal stability test. We generated the traffic using an external traffic generator that simulates the surrounding network elements. *Figure 3* presents the results. The blue bar shows CPU power consumption when running the DPDK-based UPF unmodified at idle load, at 50 percent of the maximum traffic load and at 100 percent traffic load. The green and purple bars show the energy consumption when running the same traffic load with power management without HW offload, and with both power management and HW offload.

The measurements show a 68 percent processor power reduction at idle load when using power management (the green bar on the far left in Figure 3). The bars for 50 percent load and 100 percent load with enabled power management show the reduction in processor power even as traffic load increases.

It is difficult to perfectly balance the processor load and thereby enable the cores that are not fully loaded to sleep and save energy. In our case, 10 percent of the energy could be saved at maximum traffic load (the green bar on the far right in Figure 3). This result will vary depending on the application and traffic mix.

NIC HW offload technology enables further opportunities for micro-sleep, leading to an overall 24 percent reduction at maximum load (the purple bar on the far right in Figure 3). The amount of power that can be saved using HW offload will depend on the ability of each application to utilize it and thereby save processor resources.

Synthetic benchmarks

Our proposed power management solution using micro-sleep and frequency scaling targets packet processing with an extremely low impact on packet

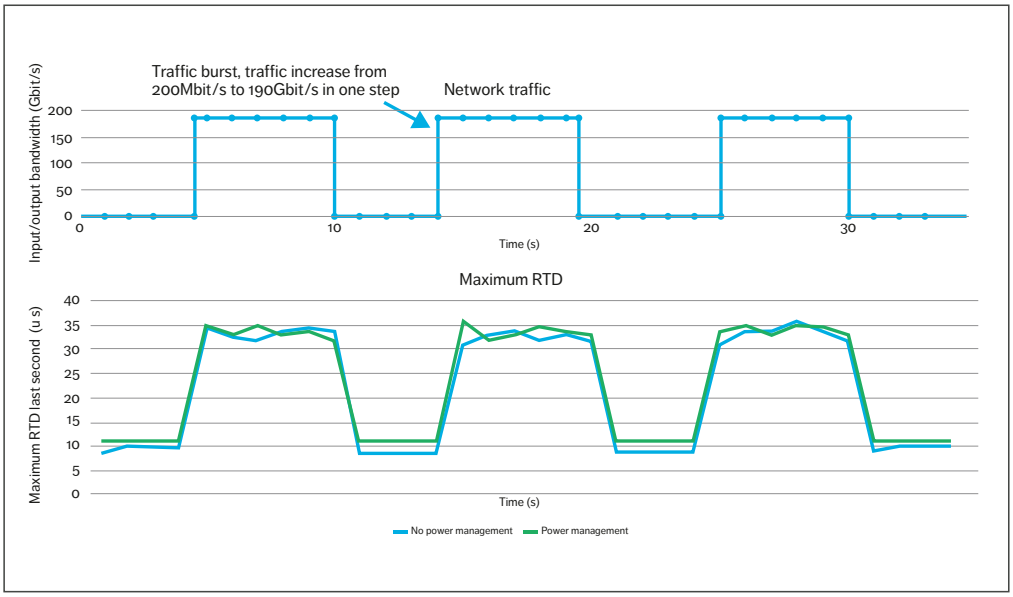


Figure 4 Measured maximum RTD when applying extreme traffic bursts, showing no increased delay when using power management

latency and a fast reaction to traffic bursts. For example, in low-latency power management, traffic bursts must not cause any packet drops. The new user-mode sleep states enable the core to wake up within a few 100ns and should have minimal impact on the total packet latency.

To measure the packet latency and the speed at which the power management functionality can react to traffic bursts, we used a synthetic benchmark where the application's impact on the packet latency is reduced as much as possible when receiving network traffic. This enabled us to measure the packet latency impact when using user-mode sleep states and frequency scaling. The synthetic benchmark is also used to measure how fast the system can scale up the performance when receiving traffic bursts.

The synthetic benchmark program fetches Ethernet packets using DPDK, swaps the media access control address and sends the packet back to the original source. The source machine measures the time from the sending of the packet to the receiving of the packet (known as the round-trip delay (RTD)) including packet drops. User-mode sleep is used when queues to the network device are empty and frequency scaling is triggered when network traffic increases/decreases.

Figure 4 shows the maximum measured RTD down to the last second, measured on the external traffic generator. At low traffic, the RTD is slightly higher when using power management (the green line in the bottom half of Figure 4) due to reduced core/uncore frequency. When traffic increased from 27Mbit/s to 190Gbit/s, the core/uncore frequency changed to maximum speed using the implemented DPDK burst detector. The power management impact of the packet delay is negligible at maximum load (190Gbit/s).

Our results indicate that the measured worst-case RTD or maximum packet jitter is not impacted, even if the RTD value is slightly higher at low traffic load. No packets were dropped during the measurements.

Conclusion

Our research clearly demonstrates that is possible to reduce energy consumption while simultaneously meeting the requirements of latency-sensitive applications. The latest generation of server processors enables fast power-saving transitions by entering sleep states directly from applications in user mode, with no impact on important KPIs like jitter and packet latency. With no overhead to enter the OS kernel and no context switches, the central processing unit core simply wakes up and continues executing when packets arrive during user-mode sleep state. Because power management is part of the user-mode packet processing library, energy savings are easy to integrate and validate on existing Data Plane Development Kit-based applications.

The combination of micro-sleep, hardware offload and frequency scaling has the potential to lead to significant energy savings. Our experiments on an Ericsson user plane function show that processor energy consumption decreased from 190W to 61W at low traffic and from 190W to 145W at maximum traffic. We also carried out experiments using a synthetic benchmark, which demonstrate that our proposed power management solution has no measurable impact on worst-case round-trip delay and worst-case packet latency.

References

1. **Data Plane Development Kit, Programmer's Guide, available at:** *https://doc.dpdk.org/guides-22.03/prog_guide/index.html*
2. **Intel, Intel® 64 and IA-32 Architectures Software Developer's Manual, available at:** *https://cdrdv2.intel.com/v1/dl/getContent/671110*
3. **AMD, AMD64 Technology, AMD64 Architecture Programmer's Manual, available at:** *https://www.amd.com/system/files/TechDocs/24594.pdf*
4. **3GPP specifications, available at:** *https://www.3gpp.org/specifications*

Further reading

- **Ericsson, How to break the energy curve, available at:** *https://www.ericsson.com/en/about-us/sustainability-and-corporate-responsibility/environment/product-energy-performance*
- **Ericsson blog, 5G energy consumption: what's the impact of 5G NR in real networks?, available at:** *https://www.ericsson.com/en/blog/2021/10/5g-energy-consumption-impact-5g-nr*
- **Ericsson, Network architecture domains, available at:** *https://www.ericsson.com/en/future-technologies/architecture/network-architecture-domains*
- **Ericsson, Network intelligence and services, available at:** *https://www.ericsson.com/en/networks*
- **Global5G.org, 5G and Energy Efficiency, available at:** *https://global5g.5g-ppp.eu/sites/default/files/BookletA4_EnergyEfficiency.pdf*

THE AUTHORS

Leif Johansson

◆ is an expert in the characteristics of open systems. He joined Ericsson in 1996 after receiving an M.Sc. in physics from Uppsala University, Sweden. Johansson has more than 20 years of experience evaluating and applying new technology in Ericsson's product portfolio.

Per Holmberg

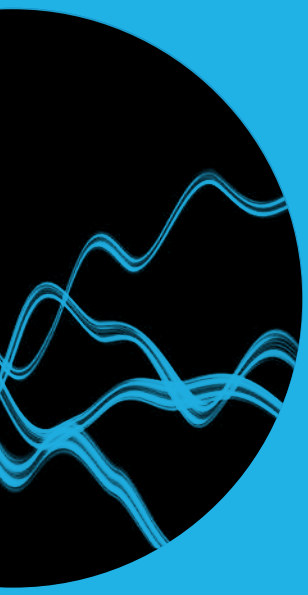
◆ is a computer architect. He holds an M.Sc. in electrical engineering from KTH Royal Institute of Technology in Stockholm, Sweden, and has worked at Ericsson since 1985 designing processing solutions primarily for communication equipment. Holmberg has held

specialist and expert positions in computer architecture and processing systems, been lead architect for several processor and computer designs and is responsible for more than 50 inventions.

Robert Skog

◆ is a senior expert in the field of service architecture. After earning an M.Sc. in electrical engineering from KTH Royal Institute of Technology in 1989, he joined Ericsson's two-year trainee program for system engineers. Since then, he has mainly worked on end-to-end solutions and traffic

optimization for everything from the first WAP solutions to today's advanced user plane solutions. In 2005, Skog won Ericsson's prestigious Inventor of the Year award.



ISSN 0014-0171
284 23-3379 | Uen

Ericsson
SE-164 83 Stockholm, Sweden
Phone: +46 10 719 0000