NICA: OS Support for Near-data Network Application Accelerators

Haggai Eran
Technion
Haifa, Israel
haggaie@tx.technion.ac.il

Lior Zeno
Technion
Haifa, Israel
liorz@tx.technion.ac.il

Gabi Malka
Technion
Haifa, Israel
gabima@technion.ac.il

Mark Silberstein
Technion
Haifa, Israel
mark@ee.technion.ac.il

ACM Reference format:
Haggai Eran, Lior Zeno, Gabi Malka, and Mark Silberstein. 2017. NICA: OS Support for Near-data Network Application Accelerators. In *Proceedings of Workshop on Multi-core and Rack Scale Systems, Belgrade, Serbia, April 2017 (MaRS 2017),* 3 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

Today's developers are facing tremendous challenges to keep up with skyrocketing network I/O rates. As CPU performance and memory bandwidth scaling stagnate, the compute capacity headroom for running network application logic is shrinking, leaving a few hundreds cycles for processing at line speed. To scale, future applications will have to break out of CPU-centric designs, instead processing data in flight on the NICs during the transfer.

Emerging NIC architectures are turning this vision into reality, incorporating increasingly programmable compute elements. For example, recent Mellanox-Innova NICs host a large Xilinx FPGA which can perform any function or serve as a network endpoint itself. Cavium LiquidIO, EZChip TileMX100, Netronome Agilio SmartNIC and Mellanox Bluefield NICs (will) allow executing arbitrary custom processing functions implemented in C. Kaufmann et al. present a programmable NIC based on match and action pipelines [3].

These modern *intelligent NICs*, or *iNICs*, are being integrated into modern data-centers [2], a process mostly driven by their ability to offload network functions such as virtualization bridging, network address translation, firewalls, network cryptography acceleration, and other network and transport layer applications. Yet since iNICs are programmable, they have the potential to accelerate higher level applications. Given a machine that is already equipped with an iNIC it would be worthwhile to utilize them for the benefit of other applications.

Consider, for example, a network service for processing temperature measurements from multiple sensors. An iNIC may extract the application data and execute a simple data analysis on each message, passing to the host only the messages with abnormal measurements. Beyond such filtering scenarios, an iNIC may dynamically steer the received data directly to another peripheral or accelerator device, e.g., NVMe SSD or a GPU while removing the CPU from the data path. The key enabler for such an accelerator-centric design is the iNIC's ability to execute transformations on network data to match the format of the target device; today such transformations must be done on the CPU.

MaRS 2017, Belgrade, Serbia
2017. 978-x-xxxx-xxxx-x/YY/MM...$15.00
DOI: 10.1145/nnnnnnn.nnnnnnn

```
// 1. NIC-side init.
k = ikernel(THERMOMETER_ID);

// 2. Communicate iKernel parameters
ik_set(k, "critical_temp", 55);

// 3. Host-side socket init.
s = socket();
bind(s);

// 4. Set-up socket on the NIC
ik_attach(k, s);

// 5. Read iKernel-specific state
ik_get(k, "average_temp");
```

Figure 1: ikernel software integration example

Unfortunately, moving application logic to the iNIC is a substantial challenge. The traditional network processing paradigm, like the one used in Click [4–6], DPDK, or P4 [1] is a poor match for developing application-level code we consider in this work. These systems expose raw Ethernet packets directly to user code on the iNIC, bypassing the network abstraction layers, network data protection and I/O performance isolation mechanisms in the OS. Moreover, the iNIC itself provides no isolation for its tasks, each being able to monopolize the NIC resources. Finally, the programmer faces the daunting task of implementing low-level network processing code, developing application-specific interfaces to interact with the iNIC from the CPU, and ends up with a highly hardware-specific, non-portable code.

In this work we develop NICA, a set of new OS abstractions and APIs that facilitate application development for both the iNIC and the CPU, and add support for efficient and safe execution of user code on iNICs. NICA offers a simple extension to the socket interface in spirit of Berkeley Packet Filters (eBPF): an application dynamically *attaches* a processing kernel, *ikernel*, to one or more network sockets, and uses a standard socket API to send and receive data (Figure 1). Underneath, the ikernel receives the transport layer packets (we currently support UDP) destined to or originated from the application, but also may create, drop or modify packets by itself. The ikernel maintains its state in local iNIC memory, which can also be accessed by the CPU. In our temperature monitoring example we store the total packet count, classifier parameters in that memory, and also use it for error reporting and debugging.

The guiding principles of our design are as follows: **(1) Ease-of-development** OS abstractions for efficient data path (send/recv), iNIC execution control (ikernel execution management and access to its state) and iNIC-CPU interaction. **(2) Compatibility with network stack** Seamless integration of iNIC processing with higher levels of the network stack (e.g., coordination of routing and ARP tables to enable sending packets from iNIC). **(3) Network I/O Isolation and protection** Support for multiple ikernels on iNIC: iNIC resource arbitration and network data protection between ikernels of different CPU processes; low overhead for network flows which do not have ikernels attached. **(4) Portability** Support for the CPU fallback to allow portability with regular NICs, simplified development and debugging. **(5) Interoperability with regular network I/O** Compatibility with other network endpoints unaware of iNIC acceleration.

Our prototype uses a Mellanox Innova NIC, which combines a NIC and an FPGA in a bump-on-the-wire design. We develop ikernels in High-Level Synthesis (HLS) inside the vendor-provided sandbox having Ethernet MACs, NIC memory access, control registers and reliable command channel. We extend it with the UDP stack functionality which includes packet steering, header and data split and checksumming. Packets arriving at the NIC are associated with a flow using a steering unit and are passed to the application logic together with the application defined context.

The ikernel hardware interfaces are based on AXI4-Stream allowing high speed data transfers to and from the ikernel. For each port (host and network) there are ingress and egress packet streams that pass UDP payload data. In addition, each packet stream is accompanied by a context stream, containing per-packet metadata. In order to control the ikernel's operation, an AXI4-Lite interface is used to expose a custom register space.

Providing isolation between ikernels is important for multitenancy and the stability of the system. We identify 4 areas where contended resource need isolation or arbitration. **(1) Network ports** Generating packets towards the network and to the host requires access to the shared network ports and arbitration is necessary. The arbiter can implement

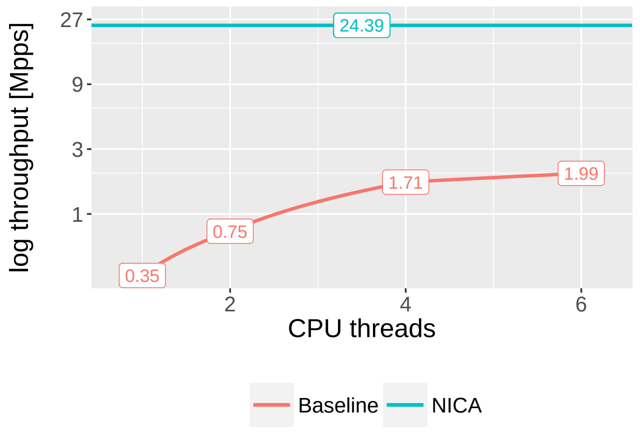

Figure 2: Throughput in millions of packets per seconds of CPU vs. NICA

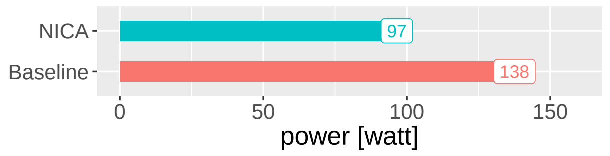

Figure 3: Power usage of CPU vs. NICA

one of the common packet scheduling algorithms or perhaps a programmable scheduler [7] can be used. **(2) DRAM memory** Local DRAM memory is limited in capacity and in bandwidth. In order for the ikernels to share the DRAM interface, a memory management unit (MMU) must be used. **(3) Power management** One may wish to limit the power used by an ikernel, by pausing its operation or with frequency scaling. **(4) FPGA resources** While FPGA resources such as LUT and BRAM units are limited, their distribution is done in advance as part of the FPGA bitstream synthesis.

We implement the thermal monitoring service which counts the number of application messages and calculates the temperature statistics (min,max,average). We use a standard unmodified UDP remote client to generate the load. Our preliminary performance results show that the iNIC can process 24.4 millions of packet/s, which is 12.3× higher than the host CPU using 6 cores and running standard Linux network stack (Figure 2), while using 30% less power (Figure 3).

REFERENCES

[1] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. 2014. P4: Programming Protocol-independent Packet Processors. *SIGCOMM Comput. Commun. Rev.* 44, 3 (July 2014), 87–95. DOI:http://dx.doi.org/10.1145/2656877.2656890

[2] Adrian Caulfield, Eric Chung, Andrew Putnam, Hari Angepat, Jeremy Fowers, Michael Haselman, Stephen Heil, Matt Humphrey,

Puneet Kaur, Joo-Young Kim, Daniel Lo, Todd Massengill, Kalin Ovtcharov, Michael Papamichael, Lisa Woods, Sitaram Lanka, Derek Chiou, and Doug Burger. 2016. A Cloud-Scale Acceleration Architecture, In Proceedings of the 49th Annual IEEE/ACM International Symposium on Microarchitecture. (October 2016). https://www.microsoft.com/en-us/research/publication/configurable-cloud-acceleration/

[3] Antoine Kaufmann, Simon Peter, Naveen Kr. Sharma, Thomas Anderson, and Arvind Krishnamurthy. 2016. High Performance Packet Processing with FlexNIC. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '16)*. ACM, New York, NY, USA, 67–81. DOI:http://dx.doi.org/10.1145/2872362.2872367

[4] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M. Frans Kaashoek. 2000. The Click Modular Router. *ACM Trans. Comput. Syst.* 18, 3 (Aug. 2000), 263–297. DOI:http://dx.doi.org/10.1145/354871.354874

[5] Bojie Li, Kun Tan, Larry Luo, Renqian Luo, Yanqing Peng, Ningyi Xu, Yongqiang Xiong, Peng Chen, Yongqiang Xiong, and Peng Cheng. 2016. ClickNP: Highly Flexible and High-performance Network Processing with Reconfigurable Hardware. (July 2016). https://www.microsoft.com/en-us/research/publication/clicknp-highly-flexible-high-performance-network-processing-reconfigurable-hardware/

[6] Teemu Rinta-aho, Mika Karlstedt, and Madhav P. Desai. 2012. The Click2NetFPGA Toolchain. In *Presented as part of the 2012 USENIX Annual Technical Conference (USENIX ATC 12)*. USENIX, Boston, MA, 77–88. https://www.usenix.org/conference/atc12/technical-sessions/presentation/rinta-aho

[7] Anirudh Sivaraman, Suvinay Subramanian, Mohammad Alizadeh, Sharad Chole, Shang-Tse Chuang, Anurag Agrawal, Hari Balakrishnan, Tom Edsall, Sachin Katti, and Nick McKeown. 2016. Programmable Packet Scheduling at Line Rate. In *Proceedings of the 2016 ACM SIGCOMM Conference (SIGCOMM '16)*. ACM, New York, NY, USA, 44–57. DOI:http://dx.doi.org/10.1145/2934872.2934899