Meili: Towards SmartNIC as a Service

Qiang Su
City University of Hong Kong
Microsoft Research

Shaofeng Wu
CUHK

Zhixiong Niu
Microsoft Research

Ran Shu
Microsoft Research

Peng Cheng
Microsoft Research

Yongqiang Xiong
Microsoft Research

Chun Jason Xue
City University of Hong Kong

Zaoxing Liu
University of Maryland

Hong Xu
CUHK

CCS CONCEPTS

• **Networks** → **Programmable networks**; **In-network processing**; • **Hardware** → **Networking hardware**;

KEYWORDS

SmartNIC, Hardware heterogeneity

ACM Reference Format:
Qiang Su, Shaofeng Wu, Zhixiong Niu, Ran Shu, Peng Cheng, Yongqiang Xiong, Chun Jason Xue, Zaoxing Liu, and Hong Xu. 2023. Meili: Towards SmartNIC as a Service. In *ACM SIGCOMM 2023 Conference (ACM SIGCOMM '23), September 10, 2023, New York, NY, USA.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3603269.3610859

1 INTRODUCTION

The gap between the stagnation of CPU power and the increase in network bandwidth has promoted a shift towards placing more computation on network hardware [16, 17]. Therefore, SmartNICs have become prevalent in data centers to serve various cloud applications, from network functions [15, 17, 22] to high-level applications like distributed applications and storage [14, 16, 18–21, 23].

Along with the prosperity of the innovations of SmartNIC applications, the industry is facing two challenges of the current use in the cloud. First, SmartNICs always feature wimpy and limited onboard resources despite the diverse hardware architectures [1–4, 7–13]. Therefore individual SmartNIC cannot conform to the requirements of all kinds of applications especially when they are highly dynamic. To address this, hardware vendors are working on designing more powerful and resourceful SmartNICs, but the pace of hardware deployment lags behind the rapid evolution of applications. Second, the sharing of SmartNICs is inefficient, as they are owned by individual application teams and it requires coordination of resource usage and workload deployment on a case-by-case basis. This leads to redundant labor on SmartNIC management and may slow down the development of SmartNIC-accelerated applications in production. Moreover, because the cloud provider lacks the complete vision of SmartNIC clusters, it becomes difficult to perform typical management tasks, such as resource allocation, scaling, workload placement and failover.

ACM SIGCOMM '23, September 10, 2023, New York, NY, USA

ACM ISBN 979-8-4007-0236-5/23/09…$15.00
https://doi.org/10.1145/3603269.3610859

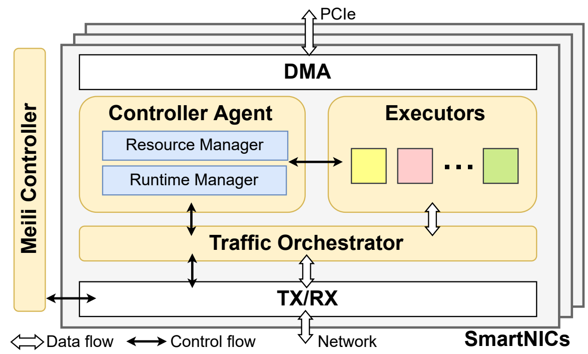

Figure 1: Overview of Meili's architecture.

In this poster, we propose a new paradigm of SmartNIC as a service to tackle these challenges. The central thesis of our approach is to organize the SmartNIC resources as a separate pool, thereby shifting the resource and workload management to the cloud provider. The workloads can be deployed without coordination among different application teams, and their owners also do not need to worry about resource limitations during development. Furthermore, cloud providers have a complete view of the SmartNIC cluster, allowing them to enforce a variety of management policies.

Following this approach, we present a novel system called Meili to efficiently develop and deploy workloads in heterogeneous SmartNIC clusters. We present the preliminary design in this poster, which aims at making SmartNIC cluster details transparent to developers and enabling flexible workload development. To do this, Meili introduces a new programming model that comprises customizable abstractions for common packet- and socket-based workloads, as well as heterogeneity-transparent functions, in which Meili implements and optimizes well-known SmartNIC-accelerated functions (*e.g.,* Crypto) using heterogeneous hardware architectures, and exposes hardware-independent APIs. With Meili's abstraction, a workload is composed with multiple finer-grained functions, which can be consolidated on heterogeneous SmartNICs. This leads to a new problem of finer-grained resource allocation and workload placement, and we leave this for future work.

2 INITIAL DESIGN

Architectural Overview. The architecture of Meili is depicted in Figure 1. In the control-plane, Meili runs a Controller Agent (CA) on each SmartNIC which also talks to a central Meili Controller. CA runs a Resource Manager and a Runtime Manager to monitor and schedule SmartNIC resource usage and workload status, synchronizes them with Meili Controller periodically, and configures the data-plane and resource management policies on each SmartNIC (*e.g.,* resource allocation). Meili Controller collects the resource and workload states from each SmartNIC, and performs global workload placement across the SmartNIC cluster to meet the performance requirements of multiple SmartNIC workloads. In the data-plane, each workload instance runs in a separate runtime, which is called an *executor* in Meili. The Traffic Orchestrator (TO) on each SmartNIC dynamically manages the traffic to/from the onboard workloads, utilizing the data-plane policies from CA (*e.g.,* load balancing).

Programming abstraction. SmartNIC workloads are generally built on two fundamental data abstractions: *packet* and *socket*, which mainly corresponds to the operations on the data packets and user application buffers, respectively. Therefore, Meili defines its abstractions as *packet processing* and *socket processing*, whose behavior can also be described by user-customized functions (UCFs). A workload is composed of the abstractions chained via a directed graph.

Packet processing. Packet processing typically involves per-packet and per-connection operations. As a result, Meili defines two data structures: 1) *Meili_packet*, which contains the packet headers, the payload, and a reference to the per-packet metadata; 2) *Meili_flow*, which contains the connection descriptor (*e.g.,* 5-tuple) and the per-connection metadata. Additionally, UCFs are defined as callback functions that can access the whole structure and compute the corresponding metadata. For example, Meili provides the packet transformation abstraction `Meili.pkt_trans()`, which allows to access, compute, and modify the *Meili_packet* by a UCF, such as changing the payload size. Other abstractions include packet filter, flow extraction and flow transformation.

Socket pocessing. Meili's socket processing abstraction follows the `epoll` mechanism, and supports operators for socket registration and event processing correspondingly. Developers can register a socket to Meili after connection establishment, allowing Meili to manage the processing on that socket. Meanwhile, the event processing functionality (*e.g.,* `EPOLL_IN`) can be defined as a UCF. Note that the socket processing is only supported on SmartNICs with complete OS stacks.

```
// User-customized functions
Meili_packet decrease_TTL(Meili_packet pkt) {
    pkt.hdr.TTL = pkt.hdr.TTL - 1;
    return pkt; }
BOOL payload_check(Meili_packet pkt) {
    // Built-in function
    return Meili.regex(RULE, pkt.payload); }
BOOL dst_IP_check(Meili_packet pkt) {
    return ip_equal(DIP, pkt.hdr.dst_ip); }
// Meili packet processing abstractions
Meili.pkt_trans(decrease_TTL, pkt); // Compute
Meili.pkt_flt(dst_IP_check, pkt); // Filter
Meili.pkt_flt(payload_check, pkt); // Filter
```

Listing 1: The pseudocode that decreases the TTL and drops the packets with specific destination IP address or DPI rule violation.

Heterogeneity-Transparent Function. It is imperative that Meili conceals the heterogeneity of SmartNICs from developers in order to maintain programming flexibility. To do this, Meili implements and optimizes a core set of functions that feature well-known SmartNIC-accelerated semantics (*e.g.,* RegEx, Crypto), and each function may have multiple implementations on various heterogeneous SmartNICs. To provide transparency to developers, Meili also provides a set of unified hardware-independent APIs and redirects the requests to appropriate implementation based on the performance requirements. For instance, Meili exposes the `Meili.AES()` function, which may have different implementations on an FPGA SmartNIC, a Crypto engine of SoC SmartNICs (*e.g.,* BlueFiled [9]), or even the onboard CPU cores. Specifically, we allow the developer to configure the shared parameters, which are usually function-specific, while Meili takes care of configuring hardware-specific parameters.

Listing 1 presents a packet processing example.

Value of K	10	100	1000
Baseline	7.62	12.36	25.44
Meili	1.25	1.83	4.67

Table 1: The average latencies (ms) of top-K flow. Baseline and Meili runs over one SmartNIC and 8 SmartNICs, respectively.

	64 B	128 B	256 B	512 B	1500 B
Baseline	1445.36	367.75	92.62	11.59	5.79
Meili	21824	21120	16384	8192	2796

Table 2: The IPSec throughput (KPPS) using an FPGA-based AES accelerator.

3 PRELIMINARY RESULTS

We showcase Meili's application benefits using 8 NVIDIA BlueField-2 SmartNICs [9] and 1 Intel FPGA SmartNIC [6].

Top-k flow. We implement a top-k flow workload with 119 LoC, and the Meili version only needs 7 lines of code change — it encapsulates the top-k logic as a UCF and utilizes the *Flow Transformation* abstraction. To evaluate the performance benefits, we generate 10000 flows using DPDK-Pktgen based on the open-source trace [5] and run the workload on the onboard CPU cores. We measure the average end-to-end latencies of searching the top 10, 100, 1000 flows across the 8 SmartNICs. Table 1 presents the results. We observe that Meili reduces the latencies by 80% when utilizing 8 SmartNICs, this demonstrates that the application logic is able to leverage more onboard CPU cores across the SmartNICs to achieve lower completion time.

IPSec. We build an IPSec using the encryption accelerator on an FPGA SmartNIC [6, 17]. The Baseline IPSec uses `libssl` and runs on the onboard CPU core, while the Meili version calls `Meili.AES()` for encryption and Meili redirects the traffic to the accelerator. The encryption algorithm is AES-256. Table 2 shows the throughput when the packet size increases. Observe that Meili achieves ~19× and ~483× throughput improvement at 64 B and 1500 B, respectively, by using the encryption accelerator. The raw performance benefits are from the FPGA architectural advantages, and Meili makes it seamless for a SmartNIC workload to attain this gain.

4 ACKNOWLEDGMENT

This work is supported in part by funding from the Research Grants Council of Hong Kong (11209520).

REFERENCES

[1] AMD Alveo Series. https://www.avnet.com/wps/portal/ebv/products/new-products/npi/2018/xilinx-u200-alveo-card-a/.
[2] Asterfusion Helium SmartNIC. https://cloudswit.ch/product/marvell-cn9670-smartnic/.
[3] Azure Catapult. https://www.microsoft.com/en-us/research/project/project-catapult/.
[4] Broadcom Stingray SmartNIC. https://docs.broadcom.com/doc/PS250-PB.
[5] CAIDA traces. https://www.caida.org/.
[6] Intel Arria 10 product table. https://www.intel.co.id/content/dam/www/programmable/us/en/pdfs/literature/pt/arria-10-product-table.pdf.
[7] Intel IPU SmartNIC. https://www.intel.com/content/www/us/en/products/details/network-io/ipu.html.
[8] Marvel OCTEON 10 SmartNIC. https://www.marvell.com/content/dam/marvell/en/public-collateral/embedded-processors/marvell-octeon-10-dpu-platform-product-brief.pdf.
[9] Mellanox BlueField-2 DPU. https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/documents/datasheet-nvidia-bluefield-2-dpu.pdf.
[10] Napatech SmartNIC. https://www.napatech.com/products/.
[11] Netronome Agilio SmartNIC. https://www.netronome.com/media/documents/PB_Agilio_CX_1x40GbE-7-20.pdf.
[12] Pensando Distributed Services Architecture SmartNIC. https://www.servethehome.com/pensando-distributed-services-architecture-smartnic/.
[13] Silicom FPGA SmartNIC. https://www.silicom-usa.com/pr/4g-5g-products/4g-5g-adapters/silicom-fpga-smartnic-n5010_series/.
[14] Jongyul Kim, Insu Jang, Waleed Reda, Jaeseong Im, Marco Canini, Dejan Kostić, Youngjin Kwon, Simon Peter, and Emmett Witchel. LineFS: Efficient SmartNIC Offload of a Distributed File System with Pipeline Parallelism. In *Proc. ACM SOSP*, 2021.
[15] Yanfang Le, Hyunseok Chang, Sarit Mukherjee, Limin Wang, Aditya Akella, Michael M. Swift, and T. V. Lakshman. UNO: Uniflying host and Smart NIC offload for flexible packet processing. In *Proc. ACM SoCC*, 2017.
[16] Bojie Li, Zhenyuan Ruan, Wencong Xiao, Yuanwei Lu, Yongqiang Xiong, Andrew Putnam, Enhong Chen, and Lintao Zhang. KV-Direct: High-Performance In-Memory Key-Value Store with Programmable NIC. In *Proc. ACM SOSP*, 2017.
[17] Bojie Li, Kun Tan, Layong (Larry) Luo, Yanqing Peng, Renqian Luo, Ningyi Xu, Yongqiang Xiong, Peng Cheng, and Enhong Chen. ClickNP: Highly Flexible and High Performance Network Processing with Reconfigurable Hardware. In *Proc. ACM SIGCOMM*, 2016.
[18] Ming Liu, Tianyi Cui, Henry Schuh, Arvind Krishnamurthy, Simon Peter, and Karan Gupta. Offloading Distributed Applications onto SmartNICs Using IPipe. In *Proc. ACM SIGCOMM*, 2019.
[19] Ming Liu, Simon Peter, Arvind Krishnamurthy, and Phitchaya Mangpo Phothilimthana. E3: Energy-Efficient Microservices on SmartNIC-Accelerated Servers. In *Proc. USENIX ATC*, 2019.
[20] Jaehong Min, Ming Liu, Tapan Chugh, Chenxingyu Zhao, Andrew Wei, In Hwan Doh, and Arvind Krishnamurthy. Gimbal: Enabling Multi-Tenant Storage Disaggregation on SmartNIC JBOFs. In *Proc. ACM SIGCOMM*, 2021.
[21] Phitchaya Mangpo Phothilimthana, Ming Liu, Antoine Kaufmann, Simon Peter, Rastislav Bodik, and Thomas Anderson. Floem: A Programming System for NIC-Accelerated Network Applications. In *Proc. USENIX OSDI*, 2018.
[22] Yiming Qiu, Jiarong Xing, Kuo-Feng Hsu, Qiao Kang, Ming Liu, Srinivas Narayana, and Ang Chen. Automated SmartNIC Offloading Insights for Network Functions. In *Proc. ACM SOSP*, 2021.
[23] Henry N. Schuh, Weihao Liang, Ming Liu, Jacob Nelson, and Arvind Krishnamurthy. Xenic: SmartNIC-Accelerated Distributed Transactions. In *Proc. ACM SOSP*, 2021.