Clara: Performance Clarity for SmartNIC Offloading

Yiming Qiu
Rice University

Qiao Kang
Rice University

Ming Liu
University of Washington

Ang Chen
Rice University

ABSTRACT

The gap between CPU and networking speeds has motivated the development of SmartNICs for near-network processing. Recent work has shown that many network functions can benefit from SmartNIC offloading, but identifying the best porting strategy requires hand-tuning and workload-specific optimizations. The developer has no easy way to understand the ported performance beforehand.

We are developing a tool called Clara, whose goal is to provide performance clarity for SmartNIC offloading. Clara can analyze an *unported* NF in its original form, and *predict* its performance when ported to a SmartNIC target. This automated workflow enables the developer to easily customize offloading strategies, obtain performance insights, and identify suitable SmartNIC models for her workloads. Clara's key technical roadmap is to *emulate a compiler*, lowering an unported program to a SmartNIC target logically, without performing code generation. This results in a mapping from core NF logic to SmartNIC hardware resources, and Clara then plugs in NIC parameters to predict the performance for specific workloads. We describe our progress so far, and report initial validation results with Netronome hardware.

KEYWORDS

SmartNICs; Network functions; Performance prediction

ACM Reference Format:
Yiming Qiu, Qiao Kang, Ming Liu, and Ang Chen. 2020. Clara: Performance Clarity for SmartNIC Offloading. In *Proceedings of the 19th ACM Workshop on Hot Topics in Networks (HotNets '20), November 4–6, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3422604.3425929

1 INTRODUCTION

High-speed networks are the backbone of datacenters, and their data rates are consistently increasing over the years [9, 19]. As a result, server CPUs spend more and more cycles on packet processing, and the consumed resources are no longer available to revenue-generating tenant VMs. Worse, the gap between CPU and networking speeds will further widen in the post-Moore era [18, 22]. This has motivated the development of SmartNICs [1, 4, 5, 8] for *near-network processing.*

Unlike traditional NICs with hardwired offloading modules (e.g., TSO/LRO/checksum), SmartNICs have programmable hardware cores, specialized packet engines, and a wide variety of domain-specific accelerators. Researchers have leveraged SmartNICs as a general-purpose offloading platform, and developed key/value store applications [33, 43], microservices [35], and other types of network functions [27, 44]. As SmartNICs stay close to the packet datapath, offloading could bring latency reduction and improve cost efficiency. Their embedded cores are also more energy-efficient than server CPUs, driving down the operational cost of cloud datacenters. As such, we have witnessed a burgeoning SmartNIC ecosystem, with a range of vendors moving into the market (e.g., Netronome, Marvell, Nvidia, Intel).

However, as a computing substrate, SmartNICs represent a significant departure from the familiar programming model and performance characteristics of x86 servers. First off, SmartNICs have heterogeneous architectures that integrate general-purpose cores, accelerators, and a complex memory hierarchy. Program performance is predicated upon a developer's understanding of the architectural details. Furthermore, vendors have adopted drastically different SmartNIC designs (e.g., programmable ASICs vs. SoC vs. FPGAs) and programming models (e.g., C [1, 4, 8], P4 [8], or Verilog [5]). As each platform requires a host of new skills, developer experience does not easily transfer across platforms. The combined effect is that a developer needs to carefully reason through layers of complexity to understand SmartNIC performance. This performance opacity has led to a manual and cumbersome development process for NF offloading.

Indeed, existing work [35, 43] resorts to hand optimizations and workload-specific benchmarking to identify the best offloading strategy. The developer often does not know beforehand how well a particular NF would perform if ported to a target hardware. She needs to first rewrite the host server program against the SmartNIC-specific toolchain, perform hardware benchmarks on a representative workload, and re-optimize the ported program when necessary. If her assumptions about the workloads, architectural details, or the NFs are amiss, multiple rounds of hand-tuning would be required. Worse, after considerable investment, she might only find that the offloading benefits do not justify the rewriting efforts in the first place. Therefore, researchers and developers would significantly benefit from an automated workflow for understanding ported performance.

In this work, we take a first step toward performance clarity in SmartNIC offloading. Our goal is to assist developers in understanding ported NF performance on a target NIC, without actually requiring them to port the program first. Our tool, Clara, analyzes the original *unported* NF, and *predicts* its offloading performance. A developer can target this analysis to different SmartNIC backends and workloads, even before she has the SmartNIC hardware at hand. She could rely on Clara's outputs to determine whether or not to offload a particular NF, how to perform an effective port, or which SmartNIC models are best suited for their workloads. We envision

Qiu and Kang contributed to this work equally.

HotNets '20, November 4–6, 2020, Virtual Event, USA
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-8145-1/20/11…$15.00
https://doi.org/10.1145/3422604.3425929

that such automated support would considerably accelerate the development cycles when offloading NFs to SmartNICs.

The design of Clara raises a set of technical challenges, including capturing heterogeneous NIC platforms, handling unported NFs, and predicting offloading performance. As the highest-level challenge, Clara must perform predictive analysis before a SmartNIC program even exists. To address this, our insight is that the traditional roles of a compiler *come close* to our need—with the caveat that compilers do not predict performance and that compiling unported programs would inevitably fail. Our technical roadmap therefore is to *mimic a compiler*. Clara attempts to lower the core NF function to a logical SmartNIC model. But instead of performing code generation, it only produces a mapping from NF logic to hardware resources. Clara estimates the best mapping by encoding a set of ILP (integer linear programming) constraints that emulate hand-tuning and optimizations when porting a program. It then plugs in performance parameters that it has collected about the NIC hardware to predict NF performance on a particular workload.

Clara is ongoing work, so we are continuing to develop a full solution to these challenges. The rest of this paper outlines the technical roadmap, and presents initial results as validation.

2 MOTIVATION

In this section, we further motivate our problem and outline the workflow of Clara.

2.1 Understanding performance is hard

Performance is a key goal in SmartNIC offloading, but understanding offloading performance before porting an NF is a daunting task. A myriad of factors come into play.

SmartNIC backends. SmartNICs come in great diversity [17]. Competing designs have endorsed ARM or MIPS SoC architectures [1, 4], FPGAs [5], and programmable ASICs [8, 28]. Moreover, their programming models range from P4 [7] to C variants [6] to Verilog/VHDL [20]. SmartNIC programs are developed and compiled using vendor-specific toolchains; and the eventual binary may run on general-purpose cores, packet processing engines, and specialized accelerators. Moreover, some SmartNICs directly sit on the datapath to process every single packet, and other off-path designs require explicit packet steering by a NIC switch. Program performance is highly dependent on SmartNIC architectures.

Offloading strategies. When porting a program to a SmartNIC, the developer needs to reorganize core NF logic and data structures, and reason about how NF components might map to the underlying hardware. Therefore, the best offloading strategy may not be obvious beforehand. Further, the ported programs would vary from developer to developer, and their performance depends on a developer's knowledge about NIC details. For instance, on Netronome Agilio, the latency of LPM (longest prefix match) functions could vary by orders of magnitude depending on whether the program uses the "flow cache"—a hardware accelerated SRAM table. Implementations that use the flow cache significantly outperform those that use software match/action processing in DRAM for cache hits. As another example, computing TCP header checksum for a 1000-byte packet at the ingress accelerator only takes 300 cycles, as packet data is immediately available. On the other hand, the same functionality at network processor cores would require 1700 extra cycles for accessing memory.

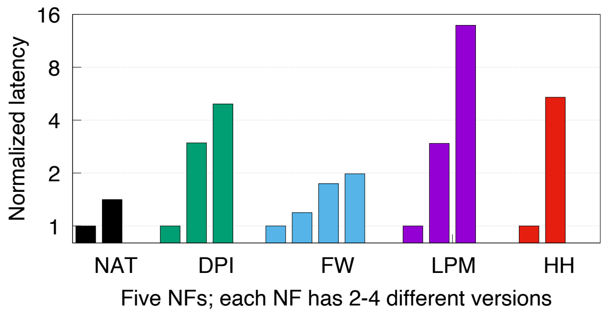


Figure 1: Performance variability of five network functions on a Netronome SmartNIC. For each NF, we benchmark two to four different versions with the same core logic. One network address translation (NAT) variant uses the checksum accelerator and the other does not. Deep packet inspection (DPI) variants handle different packet sizes. Firewall (FW) variants store flow state in different memory locations and have varying flow distributions. Longest prefix match (LPM) has different numbers of match/action rules and optionally uses the flow cache. Heavy hitter detection (HH) has varying packet rates. All NF latencies are normalized against the fastest version.

Traffic workloads. Offloading performance also depends on the NF workloads, such as the number of concurrent flows, packet types, sizes, and arrival rates. To start with, an NF may process packets differently depending on their header types. For instance, TCP SYN packets may require flow state setup, and TCP packets may incur extra L4 checksum operations than UDP. Moreover, the same operation may also take different amounts of time depending on the packet sizes—e.g., checksum, DPI. Flow distributions, on the other hand, could result in different working set sizes, which in turn cause different memory access patterns and cache behaviors. Existing work on SmartNIC offloading resorts to workload-specific benchmarking to identify the best strategies [35, 43].

Interference. Performance reasoning becomes even more challenging for co-resident NFs. Memory-intensive NFs (e.g., programs with large flow tables) may pollute the cache for other NFs. Compute-intensive NFs (e.g., DPI) may cause head-of-line blocking at accelerators. Furthermore, the degrees of parallelism, the queueing disciplines and capacities of interconnects, and the scheduling policies all contribute to the overall NF performance.

The combination of these factors often leads to unexpected performance behaviors when an NF is ported to a SmartNIC. As concrete evidence, Figure 1 shows a set of benchmarks on a Netronome SmartNIC for five NFs; we benchmark each NF with 2–4 different implementations of the same core logic, or using different workloads. Depending on the use of accelerators, memory locations, flow table sizes, packet sizes, and packet rates, the resulting NF performance can vary up to 13.8×.

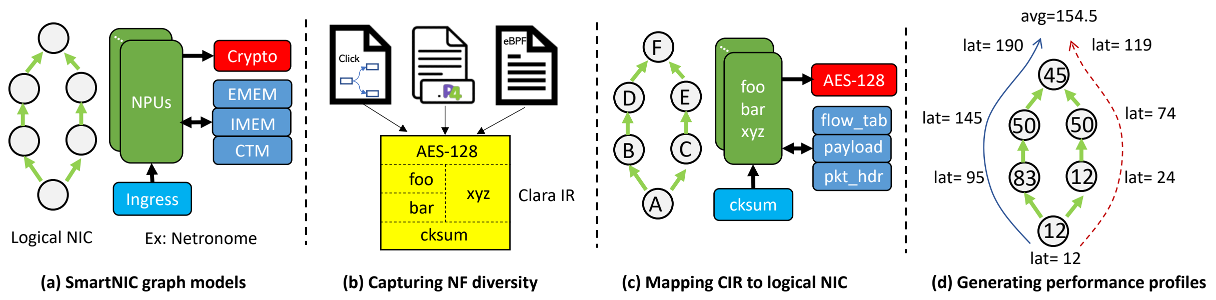


Figure 2: The technical roadmap of Clara and the key challenges. (a) Clara constructs a logical SmartNIC model that can be parameterized to capture a range of diverse NIC backends. (b) Clara transforms an input NF into an Intermediate Representation (CIR). (c) Clara maps the CIR to the logical SmartNIC, while optimizing for an performance objective. (d) Clara outputs the performance profile for the input NF on a particular workload.

2.2 State of the art & Limitations

To understand offloading performance, today we have no better approach than first porting the NF to a SmartNIC and then performing hardware benchmarks [27, 33, 35, 43, 44].

Benchmarking is the de-facto approach to understanding program performance, but in the case of NF offloading, the SmartNIC program does not exist beforehand. The original NF may be written in general-purpose C, and it might rely on different NF framework libraries (e.g., Click or eBPF APIs). The developer has to first port the NF, redesign the logic and data structures, and explore a range of offloading strategies until the best plan emerges. This is a burdensome process, and may be full of surprises if assumptions about hardware, workloads, or NF programs are slightly off.

We envision a fundamentally different approach to this problem. If we can develop support for predicting offloading performance, even if imperfectly, the developer could still gain valuable insights prior to porting that she otherwise does not have. She could rely on the performance predictions to decide whether or not to perform offloading, identify a promising porting strategy, or to choose a suitable SmartNIC model for her target workloads.

2.3 Clara: Automated performance clarity

The Clara project aims to achieve the above goal and provide automated performance clarity. The conceptual challenge that Clara needs to address is to *predict* the performance of an *unported* NF on a *hypothetical* NIC target. As we discussed, Clara's technical roadmap is to *mimic a compiler*, and Figure 2 shows the workflow.

First, Clara develops a *logical NIC* model to capture the architectural diversity of SmartNICs. Next, Clara builds a performance profile for each SmartNIC using hardware microbenchmarks, including memory latency, accelerator throughput, and other performance metrics. In order to handle NF diversity, Clara transforms an unported NF into an Intermediate Representation (IR) using LLVM [32], and then analyzes the IR to identify code blocks that may be mapped to the NIC as a whole. Clara estimates the best mapping by encoding a set of constraints from the logical NIC model, performance parameters, and the NF code blocks. Solving these constraints would produce a mapping that maximizes ported performance. Finally, Clara takes in a workload description (e.g., a pcap trace) and analyzes how packets would traverse the NF mapping. This results in latency predictions for the workload, or idealized throughput estimations.

3 SOLUTION SKETCH

In this section, we sketch the technical roadmap that we are following in developing Clara.

3.1 The logical SmartNIC model

Clara uses a *logical SmartNIC model* (LNIC) to capture heterogeneous SmartNIC architectures in a logically uniform representation. Clara models an LNIC as a graph $\langle \mathbf{V}, \mathbf{E} \rangle$, where $\mathbf{V}$ is the set of nodes and $\mathbf{E}$ the set of edges. A node could be a *compute unit*, a *memory region*, or *switching hub*. Compute units are typed—some may be header processing units, others may be domain-specific accelerators, and yet others may be general-purpose cores. Memory regions have different sizes and access latencies. Memory regions could be shared by multiple compute units, and the access latency varies depending on where the access is issued. Switching hubs include embedded NIC switches and traffic managers.

Edges represent memory buses and on-chip interconnects. An edge from a compute unit to a memory region $c \leftrightarrow m$ denotes memory accesses, and it is weighted to capture NUMA (non-uniform memory access) effects. An edge $m \leftrightarrow M$, on the other hand, represents the memory hierarchy and data eviction/fetch directions. Compute-to-compute edges $c_1 \longrightarrow c_2$ are unidirectional: they describe the staged or pipelined execution for incoming packets. Edges from and to a switching hub may involve packet queues, which are parameterized by queueing capacities and disciplines.

This graph-based model captures the majority of existing NIC architectures, although FPGA-based SmartNICs [5, 20] could in principle operate at a finer granularity. FPGAs are programmable at the gate level, and the classic approach is to use high-level synthesis (HLS) techniques to "place and route" bitstreams on the hardware. Existing work has studied FPGA performance prediction and its respective challenges [49]. That said, when FPGAs are specifically used for SmartNIC functionalities, the workflow may be different. Existing projects first develop coarser-grained accelerators on the

FPGAs (e.g., parsers) [34], and then program against these "cores". Our LNIC model reflects the latter development workflow.

Example: Netronome Agilio, for instance, has network processing units (NPUs), checksum units, and crypto accelerators. NPUs form islands, and each island has Cluster Target Memory (CTM) that is shared by all enclosed NPUs. Outside the islands are Internal and External Memory units (IMEM and EMEM), which have higher capacities and access latencies. Each NPU also has local register files and private memory. NPUs, accelerators, and IMEM/EMEM are interconnected by a distributed switch fabric.

3.2 Parameterizing the LNIC

The LNIC model describes the "skeleton" of the hardware architecture, and our next step is to annotate it with a set of hardware-specific parameters. This includes a) *architectural* parameters, such as memory sizes, degrees of parallelism, queue capacities, and b) *performance* parameters, such as memory access latencies, compute unit speeds (e.g., number of cycles for an instruction), and accelerator throughput. These annotations are NIC-dependent, so Clara needs to obtain them from hardware specifications or microbenchmarking, as a one-time effort for each SmartNIC. These benchmarking efforts are shielded from Clara users, and the obtained parameters for a NIC are reusable across NFs.

Most (though not all) SmartNIC databooks include architectural parameters. To obtain missing parameters for a NIC component, one possible approach is to perform latency/throughput tests [40], gradually increasing the offered load to locate the "knee" of the latency curve. For instance, we might observe that memory accesses to <2 kB regions have near constant latency, but it dramatically increases beyond that as memory is spilled to the next level of hierarchy. Estimating performance parameters, on the other hand, would require us to develop a set of "unit-test" benchmark programs that are NF-independent. These benchmarks should comprehensively cover compute cores, all levels of memory hierarchy, and the switching hubs of a NIC backend.

Outside the context of SmartNICs, the architecture and HPC communities have studied instruction or code block benchmarking extensively [10, 15, 47, 53, 54]. More recently, researchers have also applied machine learning to extract performance models from training programs [36]; similar techniques have been demonstrated to work well for GPU program performance [13]. Clara should be able to borrow from both lines of work.

Example: For instance, for Netronome Agilio, each NPU has 4 kB local memory, which takes 1-3 cycles to access. The CTM on each island has 256 kB memory, and takes 50 cycles to access. The IMEM and EMEM are 4 MB and 8 GB, and their access latencies are up to 250 and 500 cycles, respectively; the EMEM also has a 3MB cache. Packets smaller than 1 kB will reside in the CTM entirely, but the tails of larger packets will spill to the EMEM. In terms of compute units, each NPU core has 8 threads, and an incoming packet is always bound to a single thread. Our microbenchmarks further show that, in terms of compute performance, metadata modifications typically take 2-5 cycles on the NPU, and parsing packet headers takes around 150 cycles because header data needs to be copied from CTM to local memory.

3.3 Transforming NFs to dataflow graphs

So far, we have focused on developing SmartNIC models and identifying their performance parameters. Next, we describe how Clara handles NF diversity by generating an abstract representation. Although most network functions are written in general-purpose C, recent work has also considered alternatives such as eBPF [2] and P4 [7]; depending on the NF framework, the programs may also rely on framework-specific APIs, e.g., Click [29] or eBPF [2] calls. To handle such diversity, Clara abstracts away these details by analyzing the NFs at a lower level of representation. It transforms the input NF using LLVM [32] into a Clara Intermediate Representation (CIR). The CIR contains hardware-independent bytecode instructions, and framework-specific API calls can be easily recognized from the bytecode. Clara substitutes these calls with a set of "virtual" calls, and binds them to the SmartNIC backend later in the analysis.

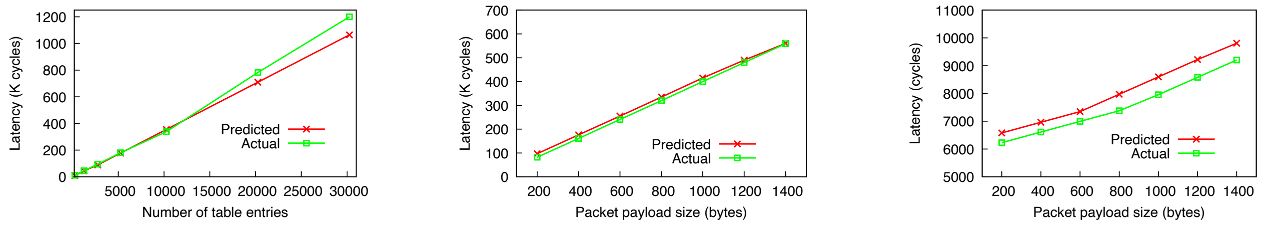
Next, Clara converts the CIR into a dataflow graph, where nodes represent program code blocks, and edges represent the traffic direction. LLVM can natively identify *basic blocks*, which is a sequence of bytecode instructions without branches or jumps—they are always executed as a whole. However, sometimes semantic information may be better captured at a coarser granularity—e.g., header parsing might require multiple branches. The LLVM infrastructure has support for various types of bytecode "pattern matching" [45], which is originally developed for bytecode rewriting and optimizations. One possible solution in Clara is to develop specialized pattern matching algorithms for SmartNIC components, e.g., header parsing or accelerator invocation.

Example: Consider NFs that are written in the Click framework. Clara would rely on API calls to 'network_header' to pattern match header parsing code blocks. It will substitute this call to 'vcall_get_hdr' in the CIR, which will later be mapped to the match/action engine in the SmartNIC.

3.4 Mapping dataflow graphs to LNIC

Next, Clara mimics the role of a compiler and attempts to lower the CIR dataflow graph to the parameterized LNIC. It estimates the best mapping by emulating a set of optimizations to capture different porting strategies and hand-tuning. Specifically, Clara identifies such a mapping by encoding a set of ILP (integer linear programming) constraints, and invoking a solver to find an optimal solution that maximizes performance (e.g., minimizing latency).

Clara first encodes a set of *compute* constraints to capture the mapping Π between dataflow nodes and LNIC compute units. For instance, 0/1 variable x_{ij} encodes whether the i-th dataflow node is mapped to the j-th compute unit, and the constraint is that every dataflow graph must be mapped to exactly one compute unit: $\forall i, \sum x_{ij} = 1$. If there exists a directed edge from one dataflow node t to another node k, the same direction must be preserved in the pipelined execution: $\Pi[k] \leq \Pi[t]$, assuming pipelined compute units are numbered in ascending order. The cost of a particular mapping depends on the performance of code blocks on their assigned compute units, e.g., by summing up the cost of every instruction in a code block based on the NIC performance parameters. Clara then encodes a set of *memory* constraints Γ for memory allocation strategies. This captures the fact that NF state could be placed at



(a) LPM **(b) VNF** **(c) NAT**

Figure 3: Initial validation on the prediction accuracy of Clara on four NFs.

different levels of memory hierarchy, and a placement strategy will incur a specific cost in terms of access latencies. For *switching hubs*, Clara formulates another set of constraints Θ that encode their queueing capacities and latencies. Solving Π, Γ, and Θ with a solver will yield a mapping that emulates a compilation process.

Additional considerations may arise if the SmartNIC datapath lacks support for certain x86 instructions. For instance, some SmartNIC cores do not have floating-point units (FPUs) [35], so such operations need to be emulated in software. Clara accounts for such mappings by capturing these instructions and emulation performance as part of the LNIC model.

Example: For instance, the ILP solver may map header processing logic to Netronome's NPUs, checksum functions to an accelerator, and place the flow table in the IMEM if it is under 3 MB.

3.5 Predicting performance

Now, Clara is almost ready to predict NF performance using the computed mapping, with another hurdle that different network packets may exercise different parts of the NF, and their performance characteristics will necessarily vary. Moreover, some NFs may exhibit different processing behaviors based on the packet history—e.g., a stateful firewall. Therefore, the user may provide a "workload profile" to describe the target traffic—e.g., a pcap trace or a more abstraction profile such as "80% TCP vs. 20% UDP" or "10 k concurrent TCP flows with 300-byte average packet size". Clara can then simulate the execution for the set of packets, and identify how a packet traverses the parameterized LNIC. Alternatively, Clara could leverage symbolic execution [14, 26] to comprehensively enumerate all NF behaviors, and identify the packet types that would exercise each behavior. This would enable Clara to generate a set of performance predictions per packet type.

Developers may be interested in different types of performance predictions. Our first step for Clara is to analyze NF latency, similar as existing work [26]. Extending the prediction for throughput analysis would require additional enhancements [36], e.g., to model scheduling, parallelism and packet queueing behaviors. Extending Clara for energy analysis would require modeling energy consumption [35].

An interesting class of challenges arises from *interference*. For instance, if multiple NFs share the same SmartNIC substrate, there may be resource contention. As a starting point, Clara could slice the LNIC to model, for instance, "half" of the NIC. In the case where LNIC "halves" leave footprints in each other's resources (e.g., cache contention), Clara should be able to leverage advances in the real-time systems community [42, 50] for worst-case execution time (WCET) analysis, which takes resource contention into account.

Example: For instance, Clara might output a profile stating that, for a particular NF, TCP and UDP packets would incur different amounts of processing cycles. It may also state that, TCP SYN packets experience higher latency, but the following packets will hit the flow cache and therefore have lower latency.

4 PRELIMINARY VALIDATION

Clara is still ongoing work, but we have already developed an early prototype and performed a set of initial experiments. The Clara prototype now has LNIC support for Netronome Agilio CX 40Gbps SmartNICs. We have obtained the performance parameters using a set of microbenchmark programs that we have developed. They measure the performance of 1) packet parsers, 2) checksum units, 3) flow cache, 4) header and metadata modifications, 5) atomic and bulk memory loads and stores, as well as 6) a subset of general-purpose compute instructions. We found that the NPU cores do not perform out-of-order execution, so they have stable performance parameters; moreover, the performance parameters for some components are functions over the data size or type—e.g., the latency for longest prefix match grows with the number of table entries. Clara uses LLVM to transform C-based NF programs into CIR.

Our experiments are conducted in a Ubuntu 18.04 server with six Intel Xeon E5-2643 Quad-core 3.40 GHz CPUs, 128 GB RAM, 1 TB hard disk, and a Netronome SmartNIC. We have evaluated three NFs: a) LPM, which performs longest prefix matches on incoming packets; b) VNF, a function chain that includes DPI, metering, header modifications, and flow statistics; and c) NAT, a network address translator that maintains a per-flow table and performs table lookups for header translation for each received packet. These NFs were originally written in C using DPDK libraries; we invoke Clara on the core packet handler functions—the DPDK processing framework itself is not included in this analysis, as the SmartNIC implementation does not have this overhead. We have also manually ported these programs to Netronome using its development toolkits. These programs are used as the baselines for understanding the prediction accuracy of Clara.

Figure 3 shows the latency prediction results for the unported NFs, and compares them with the actual latency after manually porting them to the SmartNIC. We have used 60k packets per second as the traffic rate, and computed the average latency across 1 million packets. For LPM, VNF, and NAT, we have observed a prediction inaccuracy of 12%, 3%, and 7%, respectively. This initial set of results seems encouraging, as the prediction is reasonably close to the benchmarked performance. As future work, we plan to study a wider range of NFs, SmartNIC models, and traffic profiles. With more complex programs and traffic profiles, Clara's accuracy might decrease, but even imperfect predictions could still serve as useful performance hints.

5 RELATED WORK

Performance profiling. Researchers have developed a line of work on program performance profiling. These tools can analyze program performance for GPUs [24, 25, 46], FPGAs [49], mobile SoCs [23], and specialized hardware accelerators [12, 48]. Clara is particularly related to program profilers that predict cross-platform performance: Yang et al. [51] predict overall program performance by measuring partial executions on different platforms. PHANTOM [52] measures sequential execution time on a single node to predict parallel performance at a larger scale. CERE [15] extracts representative codelets from LLVM code for performance prediction in different high-performance computing architectures. GROPHECY [37] predicts GPU performance by manually extracting program skeletons of the CPU program and predicting the performance of each component. XAPP [13] uses machine learning techniques to predict GPU program performance. Compared to these work, Clara considers an emerging class of hardware, SmartNICs, and specifically focuses on NF performance.

NF performance. High-performance packet processing is an important goal for network functions. Clara is particularly related to two recent projects: BOLT [26] and CASTAN [41]. BOLT predicts the latency profile of DPDK-based network functions written for x86 platforms. CASTAN builds a high-accuracy CPU cache model, and uses symbolic execution to enumerate program behaviors for worst-case execution time analysis. In comparison, Clara focuses on predicting *ported* performance for SmartNIC offloading. Recent NF offloading projects have highlighted the differences between SmartNIC and x86 platforms [27, 33, 35, 43, 44]; and the opacity of ported performance motivates the need for a tool like Clara.

Heterogeneous architectures. Hardware heterogeneity has received extensive attention from the architecture community, as the post-Moore era ushers in many domain-specific architectures and accelerators [22]. The use of ILP to map network functions in Clara is inspired by compute scheduling on spatial architectures [38] and multicore platforms [31]. Another project, HPVM [30], develops a parallel programming model and compiler that rely on LLVM to generate code for GPUs, CPUs, and vector hardware. Compared to HPVM, Clara only mimics a compiler, but it does not perform code generation; rather, it relies on an estimated mapping to predict performance.

Network processors. Researchers and practitioners have developed various types of network processor architectures in the past. MGR [39] designs a 50Gbps IP router using Alpha processors instead of ASIC as forwarding engines for programmability. Commercial offerings in the past such as Intel IXP [3] and IBM PowerNP [11] were also specialized for network flow processing. Nova [21] and Shangri-La [16] develop programming languages and compilers for Intel IXP network processors. The recent resurgence of SmartNICs is closely related to these earlier efforts in network processor architectures.

6 SUMMARY AND FUTURE WORK

SmartNICs have become a popular offloading platform for network functions (NFs). Existing work has demonstrated performance benefits for NF offloading, but the performance characteristics of offloaded programs are opaque prior to porting. In order to understand performance behaviors of an NF, the developer needs to first port it to the target SmartNIC and then perform workload-specific hardware benchmarks. Clara is a first step toward automated performance clarity for SmartNIC offloading. It aims to analyze an unported NF program in its original form, and predict its performance on a NIC backend with a particular workload. The resulting performance hints can guide developers to decide whether to offload a particular NF, how to perform an effective port, and which SmartNIC model may be best for the workload.

Many interesting research problems remain open as we develop a full solution to Clara. First, developing prediction techniques for *NF throughput* or *energy consumption* would require extending Clara to capture core parallelism, queueing capacity and discipline, head-of-line blocking, and energy models [35]. Second, another useful task is to understand the performance of *partial offloading*, where the NF is partitioned into two components—one resident in the SmartNIC and another in server CPUs. Capturing partial offloading performance requires reasoning about the host/NIC interconnect (e.g., PCIe), and cache coherence protocols (or the lack thereof). The architecture community has developed related work [12] that Clara should be able to borrow from. Moreover, the current results we have obtained are based on a Netronome SmartNIC; SoC-based SmartNICs have general-purpose cores therefore a closer programming model to x86 CPUs, whereas FPGA-based SmartNICs are just the opposite. Related, some SmartNICs only support run-to-completion packet processing, whereas others can additionally support pipelined processing. It would be interesting to consider a wider range of SmartNICs in Clara. Last but not least, predicting NF performance is only a starting point; developers can benefit even further if Clara can generate concrete porting strategies for different NF components as offloading hints.

7 ACKNOWLEDGMENTS

We thank our shepherd, Craig Partridge, and the anonymous reviewers for their valuable feedback. We also thank James Bornholt, Kuo-Feng Hsu, Katie Lim, Ratul Mahajan, Srinivas Narayana, Mangpo Phothilimthana, and Jiarong Xing for their insightful comments. This work was partially supported by NSF grants CNS-1801884 and CNS-2016727.

REFERENCES

[1] BlueField SmartNIC Ethernet. https://www.mellanox.com/products/BlueField-SmartNIC-Ethernet.
[2] eBPF Introduction. https://www.netronome.com/technology/ebpf/.
[3] Intel IXP processors. https://www.intel.com/content/dam/www/public/us/en/documents/specification-updates/ixp4xx-product-line-network-processors-spec-update.pdf.
[4] LiquidIOII Smart NICs. https://www.marvell.com/products/ethernet-adapters-and-controllers/liquidio-smart-nics.html.

[5] Mellanox Innova-2 Flex Open Programmable SmartNIC. https://www.mellanox.com/products/smartnics/innova-2-flex/.

[6] Netronome datapath programming tools. https://www.netronome.com/products/datapath-programming-tools/.

[7] The P4 language repositories. https://github.com/p4lang.

[8] SmartNIC Overview - Netronome. https://www.netronome.com/products/smartnic/overview/.

[9] IEEE P802.3bs 400 GbE Task Force. Adopted Timeline. http://www.ieee802.org/3/bs/, 2018.

[10] A. Abel and J. Reineke. Uops. info: Characterizing latency, throughput, and port usage of instructions on intel microarchitectures. In *Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019.

[11] J. R. Allen, B. M. Bass, C. Basso, R. H. Boivie, J. L. Calvignac, G. T. Davis, L. Frelechoux, M. Heddes, A. Herkersdorf, A. Kind, J. F. Logan, M. Peyravian, M. A. Rinaldi, R. K. Sabhikhi, M. S. Siegel, and M. Waldvogel. IBM PowerNP network processor: Hardware, software, and applications. *IBM Journal of Research and Development*, 47(2.3):177–193, 2003.

[12] M. S. B. Altaf and D. A. Wood. LogCA: A performance model for hardware accelerators. *IEEE Computer Architecture Letters*, 14(2):132–135, 2014.

[13] N. Ardalani, C. Lestourgeon, K. Sankaralingam, and X. Zhu. Cross-architecture performance prediction (XAPP) using CPU code to predict GPU performance. In *Proceedings of the 48th International Symposium on Microarchitecture*, 2015.

[14] C. Cadar, D. Dunbar, D. R. Engler, et al. Klee: Unassisted and automatic generation of high-coverage tests for complex systems programs. In *8th USENIX Symposium on Operating Systems Design and Implementation*, 2008.

[15] P. D. O. Castro, C. Akel, E. Petit, M. Popov, and W. Jalby. CERE: LLVM-based codelet extractor and replayer for piecewise benchmarking and optimization. *ACM Transactions on Architecture and Code Optimization*, 12(1), 2015.

[16] M. K. Chen, X. F. Li, R. Lian, J. H. Lin, L. Liu, T. Liu, and R. Ju. Shangri-La: Achieving high performance from compiled network applications while enabling ease of programming. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2005.

[17] S. Choi, M. Shahbaz, B. Prabhakar, and M. Rosenblum. λ-nic: Interactive serverless compute on programmable smartnics. In *IEEE International Conference on Distributed Computing Systems*, 2020.

[18] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger. Dark silicon and the end of multicore scaling. In *Proceedings of the 38th Annual International Symposium on Computer Architecture*, 2011.

[19] D. Firestone, A. Putnam, S. Mundkur, D. Chiou, A. Dabagh, M. Andrewartha, H. Angepat, V. Bhanu, A. Caulfield, E. Chung, H. K. Chandrappa, S. Chaturmohta, M. Humphrey, J. Lavier, N. Lam, F. Liu, K. Ovtcharov, J. Padhye, G. Popuri, S. Raindel, T. Sapre, M. Shaw, G. Silva, M. Sivakumar, N. Srivastava, A. Verma, Q. Zuhair, D. Bansal, D. Burger, K. Vaid, D. A. Maltz, and A. Greenberg. Azure Accelerated Networking: SmartNICs in the Public Cloud. In *15th USENIX Symposium on Networked Systems Design and Implementation*, 2018.

[20] D. Firestone, A. Putnam, S. Mundkur, D. Chiou, A. Dabagh, M. Andrewartha, H. Angepat, V. Bhanu, A. Caulfield, E. Chung, et al. Azure accelerated networking: Smartnics in the public cloud. In *15th USENIX Symposium on Networked Systems Design and Implementation*, 2018.

[21] L. George and M. Blume. Taming the IXP network processor. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2003.

[22] J. L. Hennessy and D. A. Patterson. A new golden age for computer architecture. *Communications of the ACM*, 62(2):48–60, 2019.

[23] M. Hill and V. J. Reddi. Gables: A roofline model for mobile SoCs. In *Proceedings of the IEEE International Symposium on High Performance Computer Architecture*. IEEE, 2019.

[24] S. Hong and H. Kim. An analytical model for a GPU architecture with memory-level and thread-level parallelism awareness. In *Proceedings of the 36th Annual International Symposium on Computer Architecture*, 2009.

[25] S. Hong and H. Kim. An integrated GPU power and performance model. In *Proceedings of the 37th Annual International Symposium on Computer Architecture*, 2010.

[26] R. Iyer, L. Pedrosa, A. Zaostrovnykh, S. Pirelli, K. Argyraki, and G. Candea. Performance contracts for software network functions. In *16th USENIX Symposium on Networked Systems Design and Implementation*, 2019.

[27] G. P. Katsikas, T. Barbette, D. Kostic, R. Steinert, and G. Q. Maguire Jr. Metron:NFV service chains at the true speed of the underlying hardware. In *15th USENIX Symposium on Networked Systems Design and Implementation*, 2018.

[28] A. Kaufmann, S. Peter, N. K. Sharma, T. Anderson, and A. Krishnamurthy. High performance packet processing with FlexNIC. In *Proceedings of the 21st International Conference on Architectural Support for Programming Languages and Operating Systems*, 2016.

[29] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The Click modular router. *ACM Trans. Comput. Syst.*, 18(3):263âĂŞ297, 2000.

[30] M. Kotsifakou, P. Srivastava, M. D. Sinclair, R. Komuravelli, V. Adve, and S. Adve. HPVM: Heterogeneous parallel virtual machine. In *Proceedings of ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 2017.

[31] M. Kudlur and S. Mahlke. Orchestrating the Execution of Stream Programs on Multicore Platforms. In *Proceedings of the 29th ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2008.

[32] C. Lattner and V. Adve. LLVM: A compilation framework for lifelong program analysis & transformation. In *Proceedings of CGO*, 2004.

[33] B. Li, Z. Ruan, W. Xiao, Y. Lu, Y. Xiong, A. Putnam, E. Chen, and L. Zhang. Kv-direct: High-performance in-memory key-value store with programmable nic. In *Proceedings of the 26th Symposium on Operating Systems Principles*, 2017.

[34] B. Li, K. Tan, L. Luo, Y. Peng, R. Luo, N. Xu, Y. Xiong, P. Cheng, and E. Chen. ClickNP: Highly flexible and high performance network processing with reconfigurable hardware. In *Proceedings of the 2016 ACM SIGCOMM Conference*, 2016.

[35] M. Liu, S. Peter, A. Krishnamurthy, and P. M. Phothilimthana. E3: Energy-efficient microservices on smartnic-accelerated servers. In *Proceedings of ther 2019 USENIX Annual Technical Conference*, 2019.

[36] C. Mendis, A. Renda, S. P. Amarasinghe, and M. Carbin. IIthemal: Accurate, portable and fast basic block throughput estimation using deep neural networks. In *Proceedings of International Conference on Machine Learning*, 2019.

[37] J. Meng, V. A. Morozov, K. Kumaran, V. Vishwanath, and T. D. Uram. Grophecy: GPU performance projection from cpu code skeletons. In *Proceedings of International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2011.

[38] T. Nowatzki, M. Sartin-Tarm, L. De Carli, K. Sankaralingam, C. Estan, and B. Robatmili. A General Constraint-Centric Scheduling Framework for Spatial Architectures. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2013.

[39] C. Partridge, P. P. Carvey, E. Burgess, I. Castineyra, T. Clarke, L. Graham, M. Hathaway, P. Herman, A. King, S. Kohalmi, et al. A 50-Gb/s IP router. *IEEE/ACM Transactions on Networking*, 6(3):237–248, 1998.

[40] N. M. Patel. Half-latency rule for finding the knee of the latency curve. *ACM Performance Evaluation Review*, 43:28–29, 2014.

[41] L. Pedrosa, R. Iyer, A. Zaostrovnykh, J. Fietz, and K. Argyraki. Automated synthesis of adversarial workloads for network functions. In *Proceedings of the 2018 ACM SIGCOMM Conference*, 2018.

[42] L. T. X. Phan, M. Xu, and I. Lee. Cache-aware interfaces for compositional real-time systems. In *Proceedings of Workshop on Compositional Theory and Technology for Real-Time Embedded Systems*, 2015.

[43] P. M. Phothilimthana, M. Liu, A. Kaufmann, S. Peter, R. Bodik, and T. Anderson. Floem: A programming system for NIC-accelerated network applications. In *13th USENIX Symposium on Operating Systems Design and Implementation*, 2018.

[44] S. Pontarelli, R. Bifulco, M. Bonola, C. Cascone, M. Spaziani, V. Bruschi, D. Sanvito, G. Siracusano, A. Capone, M. Honda, et al. Flowblaze: Stateful packet processing in hardware. In *16th USENIX Symposium on Networked Systems Design and Implementation*, 2019.

[45] S. Sarda and M. Pandey. *LLVM Essentials*. O'Reilly, 2015.

[46] J. Sim, A. Dasgupta, H. Kim, and R. Vuduc. A performance analysis framework for identifying potential benefits in GPU applications. In *Proceedings of the 17th ACM SIGPLAN symposium on Principles and Practice of Parallel Programming*, 2012.

[47] A. Snavely, L. Carrington, N. Wolter, and J. Labarta. A framework for performance modeling and prediction. In *Proceedings of the ACM/IEEE Conference on Supercomputing*, 2002.

[48] A. Sriraman and A. Dhanotia. Accelerometer: Understanding acceleration opportunities for data center overheads at hyperscale. In *Proceedings of the 25th International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020.

[49] Z. Wang, B. He, W. Zhang, and S. Jiang. A performance analysis framework for optimizing opencl applications on fpgas. In *Proceedings of the IEEE International Symposium on High Performance Computer Architecture*. IEEE, 2016.

[50] M. Xu, L. T. X. Phan, I. Lee, O. Sokolsky, S. Xi, C. Lu, and C. Gill. Cache-aware compositional analysis of real-time multicore virtualization platforms. In *Proceedings of IEEE Real-Time Systems Symposium*, 2013.

[51] L. T. Yang, X. Ma, and F. Mueller. Cross-platform performance prediction of parallel applications using partial execution. In *Proceedings of the ACM/IEEE Conference on Supercomputing*. IEEE, 2005.

[52] J. Zhai, W. Chen, and W. Zheng. Phantom: Predicting performance of parallel applications on large-scale parallel machines using a single node. *ACM Sigplan Notices*, 45(5):305–314, 2010.

[53] R. Zhang, Z. Budimlić, and K. Kennedy. Performance modeling and prediction for scientific Java applications. In *Proceedings of IEEE International Symposium on Performance Analysis of Systems and Software*, 2006.

[54] W. Zhang, M. Hao, and M. Snir. Predicting HPC parallel program performance based on LLVM compiler. *Cluster Computing*, 20, 2017.