Online RL in the Programmable Dataplane with OPaL

Kyle A. Simpson, Dimitrios P. Pezaros
University of Glasgow, Scotland
k.simpson.1@research.gla.ac.uk

University of Glasgow — NETLAB Networked Systems Research Laboratory

Reinforcement learning (RL) is a key tool in data-driven networking for learning to control systems online. While recent research has shown how to offload machine learning tasks to the dataplane (reducing processing latency), online learning remains an open challenge unless the model is moved back to a host CPU, harming latency-sensitive applications. Our poster introduces OPaL—On Path Learning—the first work to bring online reinforcement learning to the dataplane. OPaL makes online learning possible in SmartNIC/NPU hardware by returning to classical RL techniques—avoiding neural networks. This simplifies update logic, enabling online learning, and benefits well from the parallelism common to SmartNICs. We show that our implementation on Netronome SmartNIC hardware offers concrete latency improvements over host execution.

Introduction

Accelerated ML inference is well-studied…

· DNN accelerators (e.g., *BrainWave*) allow line rate inference of *trained* models with 32x lower batching.

· *Cost: O(ms) latency, offline only.*

· Dataplane ML converts a *trained* model → NIC/Switch friendly data format (e.g., *BNN, MATs*) to act on per-packet or per-flow state.

The main research questions:

· Can we have online learning *in the dataplane*?

· Enable learn *online* control from NIC-only data without simulation?

· How do we take advantage of the design of SmartNIC hardware to do this?

· *I.e., many "wimpy" cores, no FPU.*

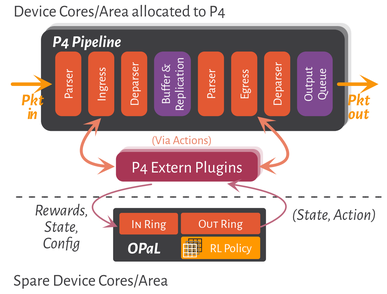


Fig. 1—Design & Interaction with P4

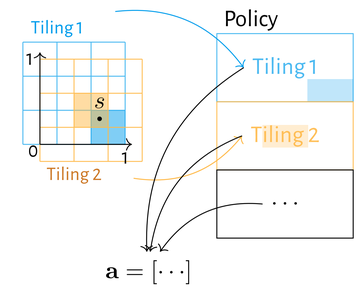


Fig. 2 (Algorithm)—Tile-coded RL: a) is map-reduce, b) is wait-free for fixed-point tiles, taking advantage of NPU parallelism.

Architecture

Design *(fig. 1)*: *Perform inference and training using extra device cores/area, interface with existing control plane via externs.*

Algorithm *(fig. 2)*: *Classical tile-coding + SARSA are O(µs) even online, enabled on this hardware by fixed-point arithmetic. This enables a novel, parallel, wait-free algorithm implementation.*

Implementation *(fig. 3)*: *Inference/update commands pushed with state data (1, 2), scattered to workers (3), gathered via shared atomic preference list (4), read out & installed (5–7).*

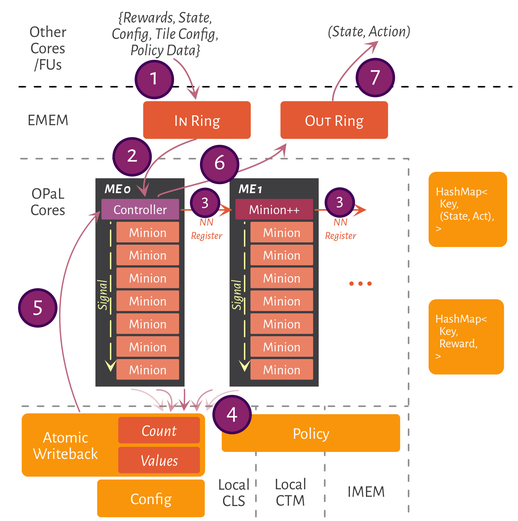
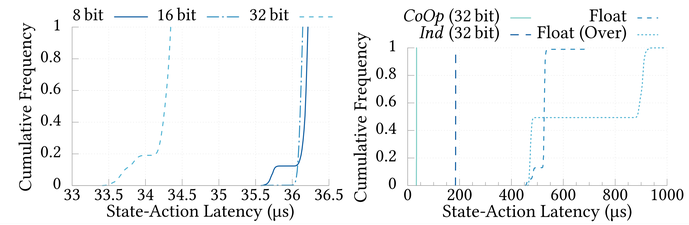


Fig. 3—Implementation on Netronome SmartNIC, signalling cores (MEs) to split inference/updates across hardware threads.



Datatype	Machine/FW	Workers	Throughput (k actions/s)		Throughput/core (k actions/s)	
			Offline	Online	Offline	Online
Float	Collector	4	7.673(49)	1.627(31)	1.918(12)	—
	MidServer	6	5.584(30)	0.791(12)	0.931(5)	—
Int32	OPaL-*Ind*	32	**172.875(229)**	4.333(5)	**5.402(7)**	—
	OPaL-*CoOp*	32	29.166(173)	**16.141(73)**	0.911(5)	**0.504(2)**

Analysis

From our implementation on Netronome SmartNICs (*vs numpy f32 on i7-6700K*), for both *online* (*CoOp*, live training) and *offline* (*Ind*, pre-trained) operation of low-latency RL control:

· Order of magnitude latency improvements at median and 99.99th %ile for both parallel strategies (*Ind*, *Coop*).

· Greater throughput (per-core) without reducing P4 processing resources.

· Small effect on cross-traffic processing.

Future Work

· Implementation of OPaL functional units in NetFPGA.

· End-to-end comparison of accuracy, convergence, latency in AQM and traffic engineering use cases.