

Revisiting the Classics: Online RL in the Programmable Dataplane

Kyle A. Simpson, Dimitrios P. Pezaros

✉ k.simpson.1@research.gla.ac.uk

🌐 FelixMcFelix 🌐 <https://mcfelix.me>

4th June, 2021

University of Glasgow



Data-driven networking: Automate control, optimisation, configuration of the network.

- Flow performance optimisation.
- Resource allocation.
- Adaptive response to load, intrusions, etc.
- Feedback loop-like.

Why programmable data-planes?

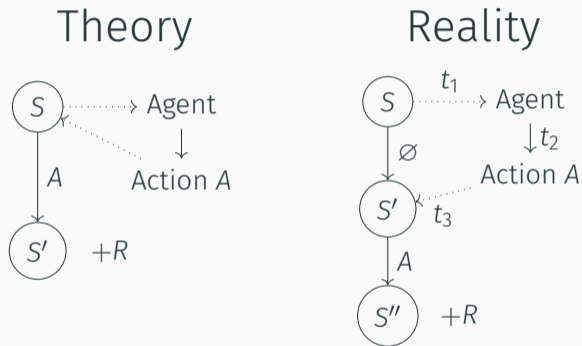
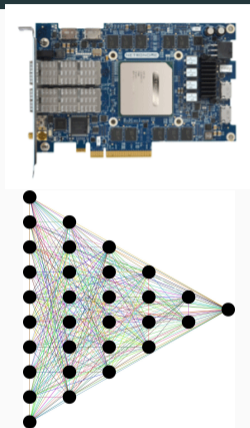


Figure 1: Asynchronous RL delays and state slippage (policy updates omitted).

- In data-driven, want to **minimise time to act**.
- RL assumes that action & policy update are zero-cost.
 - Not so in real deployments!
 - State drift, etc.
- Controller contact time, serialisation, ...
- In other ML, often need line rate inference.
- Programmable network hardware fills this niche. 3/23

Recent Programmable Trends in Data-Driven Networks

- ML acceleration, line-rate packet classification.
- How? Train model off-NIC, convert to **binary neural network**¹, or **decision tree**².
- Limits? No online training, cost of backprop algo (expensive!), vast data needs.
- **What if we need online learning?**



¹Siracusano *et al.*, 'Running Neural Networks on the NIC'.

²Xiong and Zilberman, 'Do Switches Dream of Machine Learning?: Toward In-Network Classification'.

Timing: Why not offload to the controller?

For SmartNICs, the attached host *is the (closest) controller*.

- PCIe access times $\mathcal{O}(\mu\text{s})^3$
- Crossing VMs/vNFS has $\sim 10 \times$ higher cost⁴
- MATs recompiled $\mathcal{O}(s)$. Many rules \implies batching.

Meanwhile core-to-core on NFP around 140 ns @ 1.2 GHz.

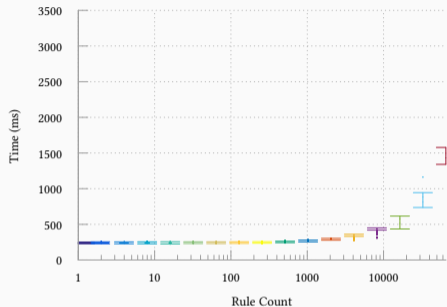


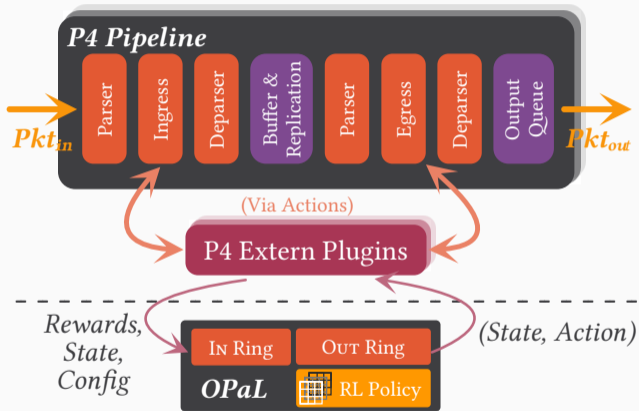
Figure 2: Netronome rule installation cost (1 table, 1–65 536 rules).

³Neugebauer *et al.*, 'Understanding PCIe performance for end host networking'.

⁴Cziva and Pezaros, 'Container Network Functions: Bringing NFV to the Network Edge'.

How do we bring online, in-NIC RL?

Device Cores/Area allocated to P4



Spare Device Cores/Area

- Classical RL built on tile-coding—**online**.
- **Fixed-point arithmetic**.
- Async wrt. datapath.
- Dynamic selection of last reward, trace info.
- **Runtime configurable** (policy, size, application) from data/control-plane. **Task independent**.

Background and Design

Single-step (classical) RL—Sarsa

A simple explanation:

How to select an action? Pick largest from list: $\hat{q}(S_t, \cdot, \mathbf{w}_t)$

How should we adjust the value of selected items?

$$\delta_t = \overbrace{R_{t+1} + \gamma \hat{q}(S_{t+1}, A_{t+1}, \mathbf{w}_t)}^{\text{New target value: Reward+some of the next action's value}} - \underbrace{\hat{q}(S_t, A_t, \mathbf{w}_t)}_{\text{Current value estimate}},$$

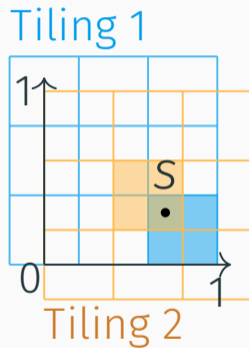
Policy parameter update:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \overbrace{\alpha \delta_t}_{\text{Move a little bit of } \delta_t \text{ along...}} \underbrace{\nabla \hat{q}(S_t, A_t, \mathbf{w}_t)}_{\text{...the policy's gradient}}.$$

Design implications?

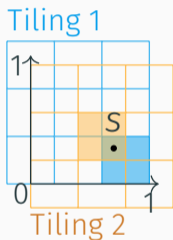
$$s = \begin{pmatrix} 0.7 \\ 0.3 \end{pmatrix}$$

$$x(s, \cdot) = \left\{ \begin{array}{l} T_{1,9}, \\ T_{2,5}, \\ T_{bias} \end{array} \right\}$$



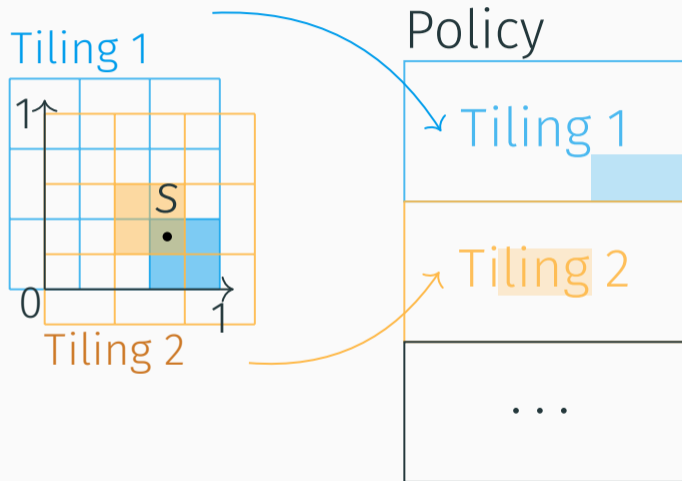
Tile Coding (ii)

$$s = \begin{pmatrix} 0.7 \\ 0.3 \end{pmatrix}$$
$$x(s, \cdot) = \left\{ \begin{array}{l} T_{1,9}, \\ T_{2,5}, \\ T_{bias} \end{array} \right\}$$

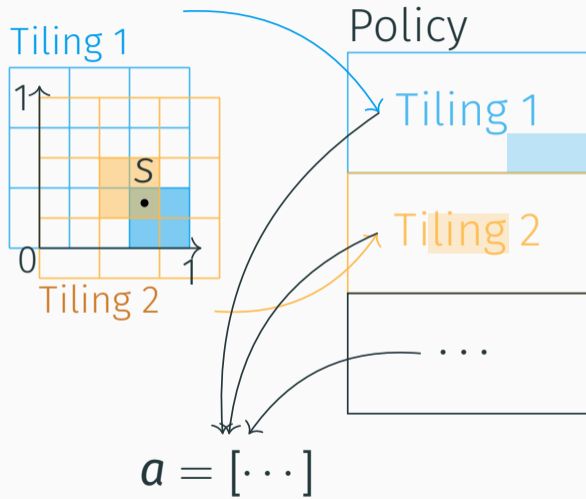


- Operations? $+$, $-$, \times , \div
 - PoT tile width? \div replaced w/ shift.
- Tiling set: identical dimensions, different shifts
- Gradient for RL?
 - List of hit tiles.
- Can be more complex...

Tile Coding: Parallelism



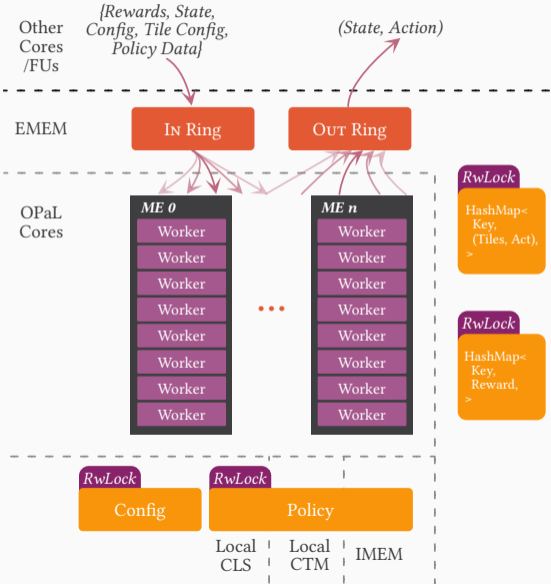
Tile Coding: Parallelism (ii)



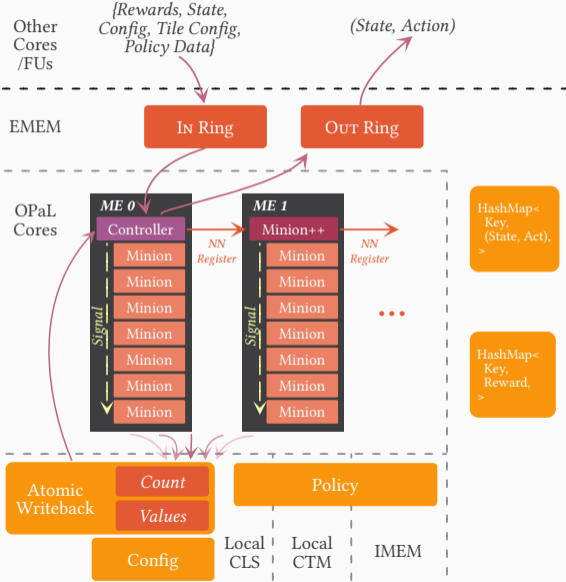
Designs: How to exploit on-NIC parallelism?

- Netronome SmartNICs *very* multi-core (NFP-6480).
 - NetFPGAs also allow creating separate, effectively async functional units.
- Two ways to take advantage:
 - Available threads process *Independently*.
 - Available threads *CoOperate* on each inference or learning task.
- Basic algorithm:
 - (Parallel) action compute.
 - Output action.
 - Check for trace in progress for this input.
 - If found: compute δ , do (parallel) policy update.

Designs: *Ind* (on NFP)



Designs: CoOp (on NFP)



Evaluation

Versus commodity hosts...

- State-Action/Update latency
- Online/Offline throughput
- Impact on cross-traffic
- Device resource use

On large-ish policies (20D state, 10 actions, bias+(7×1D, 8×2D, 1×4D)).

Latency

Datatype	Machine/FW	State-Action Latency (μs)			State-Update Time (μs)		
		Median	99 th	99.99 th	Median	99 th	99.99 th
Float	Collector	515.94	606.06	725.03	606.06	636.82	833.99
	MidServer	1 069.07	1 125.1	1 508.0	1 260.04	1 605.99	1 719.864
Int32	OPaL-Ind	185.133	185.533	186.213	230.840	231.347	232.227
	OPaL-CoOp	34.347	34.520	34.573	62.000	62.440	63.120

Lower latency with **just one (slower) core**.

One NFP island \implies **15 \times median S \rightarrow A speedup**.

Far tighter tail than host offload!

Latency—Bit Depth

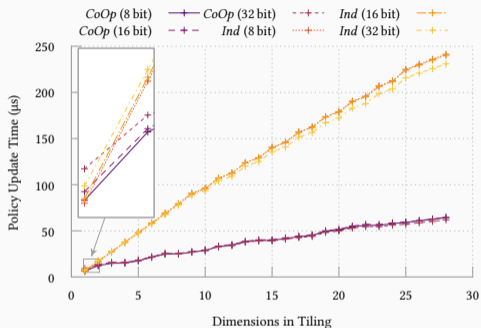
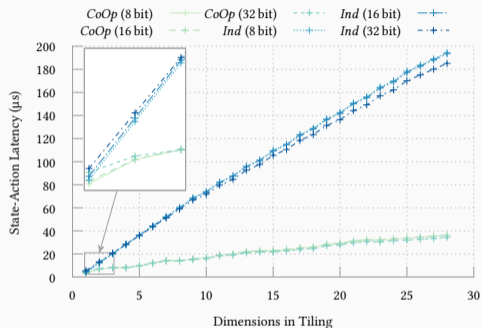


Figure 3: OPaL action and update latencies based on work size (crossover at 3-dim, 10-dim).

Latency—Core Count

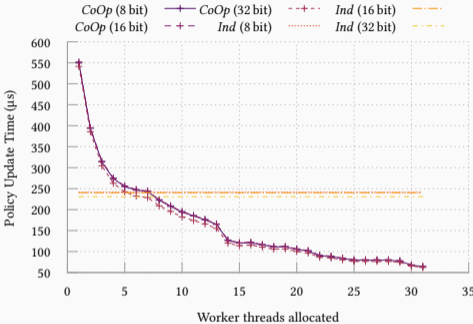
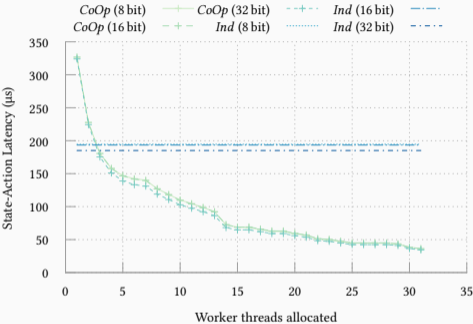
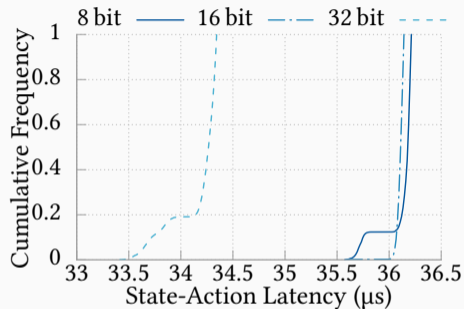
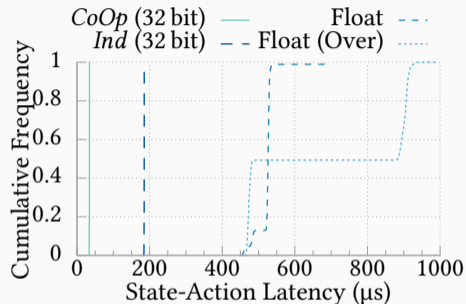


Figure 4: OPaL action and update latencies based on worker count (crossover at 3-core, 8-core).

Examining Tail Latencies



(a) CoOp latency CDFs



(b) OPaL and host latency CDFs

Figure 5: Cumulative state-action latency plots for OPaL and host-based execution.

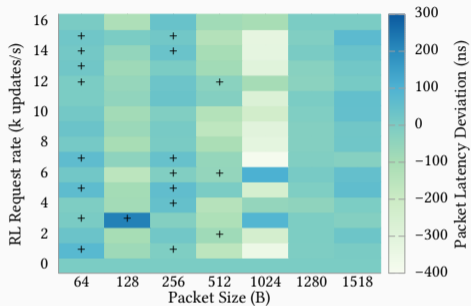
Throughput

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)

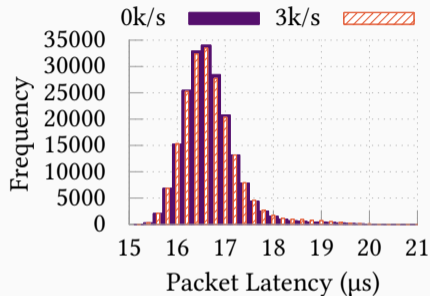
In-NIC and quantised offers **higher throughput per core**.

Parallel Sarsa key to maximising **online** throughput.

Impact on cross-traffic



(a) Deviations in 99th percentile RTTs.



(b) 128 B outlier (99th +222 ns)

Figure 6: Effects on tail latency of cross-traffic—typically sub-78 ns.

Resource Use

Firmware	EMEM		EMEM Cache		IMEM		i5.CLS		i5.CTM	
	MiB	%	KiB	%	KiB	%	KiB	%	KiB	%
Base P4	6 776.67	88.24	268.52	2.91	858.28	10.48	0.00	0.00	0.00	0.00
<i>Ind</i> (1)	6 780.21	88.28	2 541.08	27.57	1 263.28	15.42	24.75	38.67	94.25	36.82
<i>Ind</i> (4)	6 780.22	88.28	2 545.33	27.62	1 263.28	15.42	51.18	79.97	107.00	41.80
<i>CoOp</i> (1)	6 779.12	88.27	1 773.59	19.24	1 263.28	15.42	22.41	35.01	90.00	35.16
<i>CoOp</i> (4)	6 779.12	88.27	1 769.84	19.20	1 263.28	15.42	52.16	81.49	90.00	35.16

- *Ind*: 27 μ s
- *CoOp*: 54–238 μ s
- **New policy data?** Just memcopies.
- Only design, bit depth, max policy sizes need recompile.
- Can mix and match agent types in the network, export learned policy over control plane.

Takeaways:

Online in-NIC RL is possible!

Order-of-magnitude latency improvement over offloading, higher online throughput.

Platform-specific, but similar design for SmartNIC hardware class.

Future work: use-cases (AQM, DDoS prevention & accuracy), NetFPGA, transfer learning.

Questions?

Scheduler performance

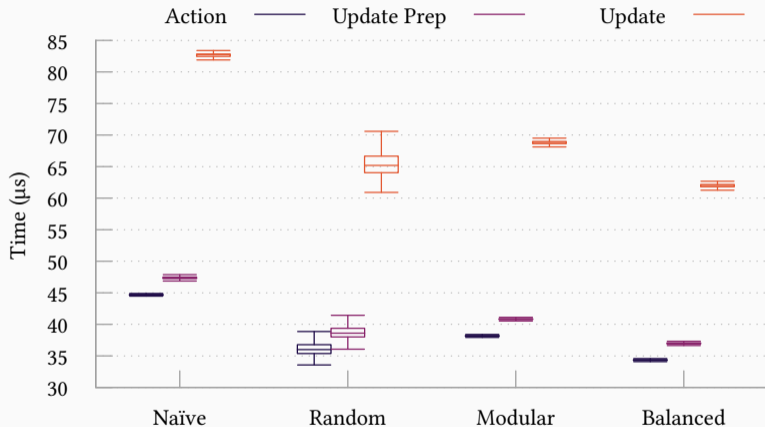


Figure 7: Action/update compute times in a 32 bit CoOp agent under different work schedulers.

Per-worker throughput

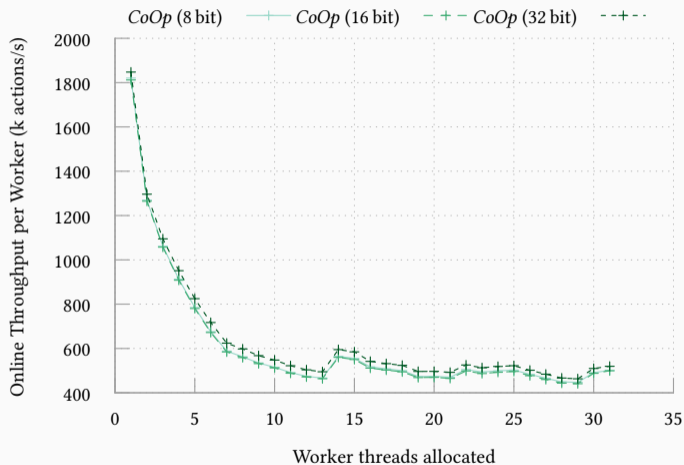


Figure 8: Throughput per added worker in a CoOp agent.



Latency—All OPaL

Datatype	Machine/FW	State-Action Latency (μs)			State-Update Time (μs)		
		Median	99 th	99.99 th	Median	99 th	99.99 th
Float	Collector	515.94	606.06	725.03	606.06	636.82	833.99
	MidServer	1 069.07	1 125.1	1 508.0	1 260.04	1 605.99	1 719.864
Int32	OPaL-Ind	185.133	185.533	186.213	230.840	231.347	232.227
	OPaL-CoOp	34.347	34.520	34.573	62.000	62.440	63.120
Int16	OPaL-Ind	193.427	193.787	194.587	240.333	240.840	241.560
	OPaL-CoOp	36.147	36.240	36.280	64.667	65.080	65.973
Int8	OPaL-Ind	194.520	194.840	195.240	241.173	241.707	242.760
	OPaL-CoOp	36.227	36.307	36.347	64.333	64.867	65.693

Throughput—All OPaL

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)
Int16	OPaL-Ind	32	165.437(118)	4.161(4)	5.170(4)	—
	OPaL-CoOp	32	27.664(36)	15.471(54)	0.865(1)	0.483(2)
Int8	OPaL-Ind	32	164.524(142)	4.147(5)	5.141(4)	—
	OPaL-CoOp	32	27.631(101)	15.552(68)	0.863(3)	0.486(2)

References i

-  Cziva, Richard and Dimitrios P. Pezaros. 'Container Network Functions: Bringing NFV to the Network Edge'. In: *IEEE Commun. Mag.* 55.6 (2017), pp. 24–31. DOI: [10.1109/MCOM.2017.1601039](https://doi.org/10.1109/MCOM.2017.1601039). URL: <https://doi.org/10.1109/MCOM.2017.1601039>.
-  Neugebauer, Rolf, Gianni Antichi, José Fernando Zazo, Yury Audzevich, Sergio López-Buedo and Andrew W. Moore. 'Understanding PCIe performance for end host networking'. In: *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication, SIGCOMM 2018, Budapest, Hungary, August 20-25, 2018*. Ed. by Sergey Gorinsky and János Tapolcai. ACM, 2018, pp. 327–341. DOI: [10.1145/3230543.3230560](https://doi.org/10.1145/3230543.3230560). URL: <https://doi.org/10.1145/3230543.3230560>.

References ii

-  Siracusano, Giuseppe, Salvator Galea, Davide Sanvito, Mohammad Malekzadeh, Hamed Haddadi, Gianni Antichi and Roberto Bifulco. 'Running Neural Networks on the NIC'. In: *CoRR abs/2009.02353* (2020). arXiv: 2009.02353. URL: <https://arxiv.org/abs/2009.02353>.
-  Xiong, Zhaoqi and Noa Zilberman. 'Do Switches Dream of Machine Learning?: Toward In-Network Classification'. In: *Proceedings of the 18th ACM Workshop on Hot Topics in Networks, HotNets 2019, Princeton, NJ, USA, November 13-15, 2019*. ACM, 2019, pp. 25–33. ISBN: 978-1-4503-7020-2. DOI: 10.1145/3365609.3365864. URL: <https://doi.org/10.1145/3365609.3365864>.