On the Analysis of Computational Delays in Reinforcement Learning-based Rate Adaptation

Ricardo Trancoso, João Pinto, Rúben Queirós, Hélder Fontes, Rui Campos

INESCTEC

U.PORTO
FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

EAI SIMUTools 2023
Seville, Spain

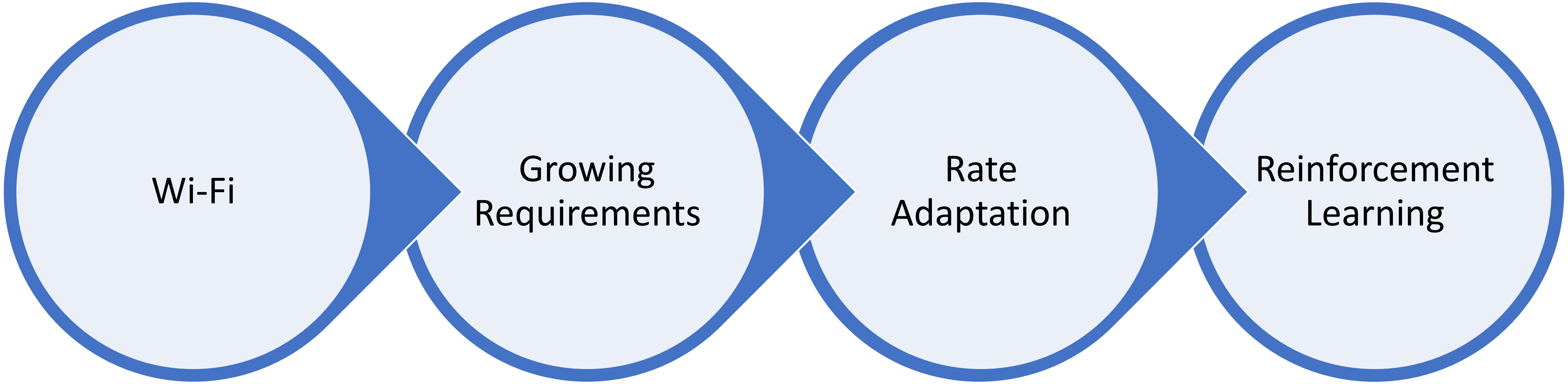
Introduction

Methodology and Implementation

Results

Conclusions

Context


Wi-Fi
Growing Requirements
Rate Adaptation
Reinforcement Learning

Context

Computational Delays affect performance

Authors do not provide implementational details

Possible gap in the literature

Contributions

Sensitize	Reduce	Evaluate
Raise awareness of the execution time problem	Describe methods to reduce delays	Create a framework to simulate these delays

Introduction

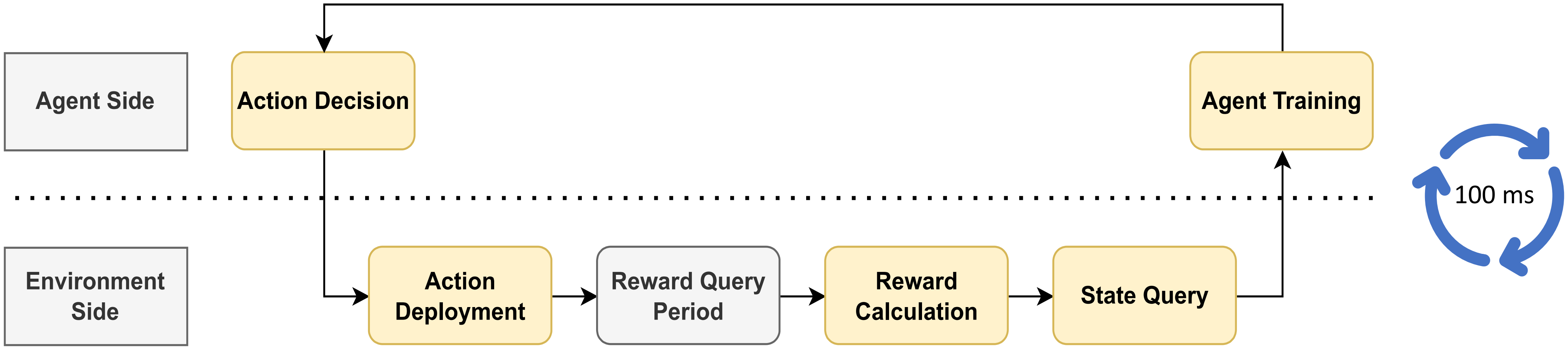
Methodology and Implementation

Results

Conclusions

DARA Overview

DARA **D**ata-driven **A**lgorithm for **R**ate **A**daptation



Preliminary Experiment

- In simulation, DARA performed satisfactorily
- Implementation in a real environment (Base DARA) was not trivial
- Example of overlooking the effect of computational delays
- Average execution time of one loop was 528.8 ms!

Goals

Minimize delays

Keep conceptual design

Keep hardware

Improvements

- Low-level Information Access
- Information Collection
- Information Parsing

Low-level Information Access

- Reward information took 100 ms to update.
- Solution: Modify mac80211 Linux kernel module
  - Provides up-to-date data directly from the kernel

- However, waiting time after each action is needed
  - Preliminary: reduce period to 50 ms
  - Final: file read assynchronously, rest of the algorithm can proceed during wait

Information Collection

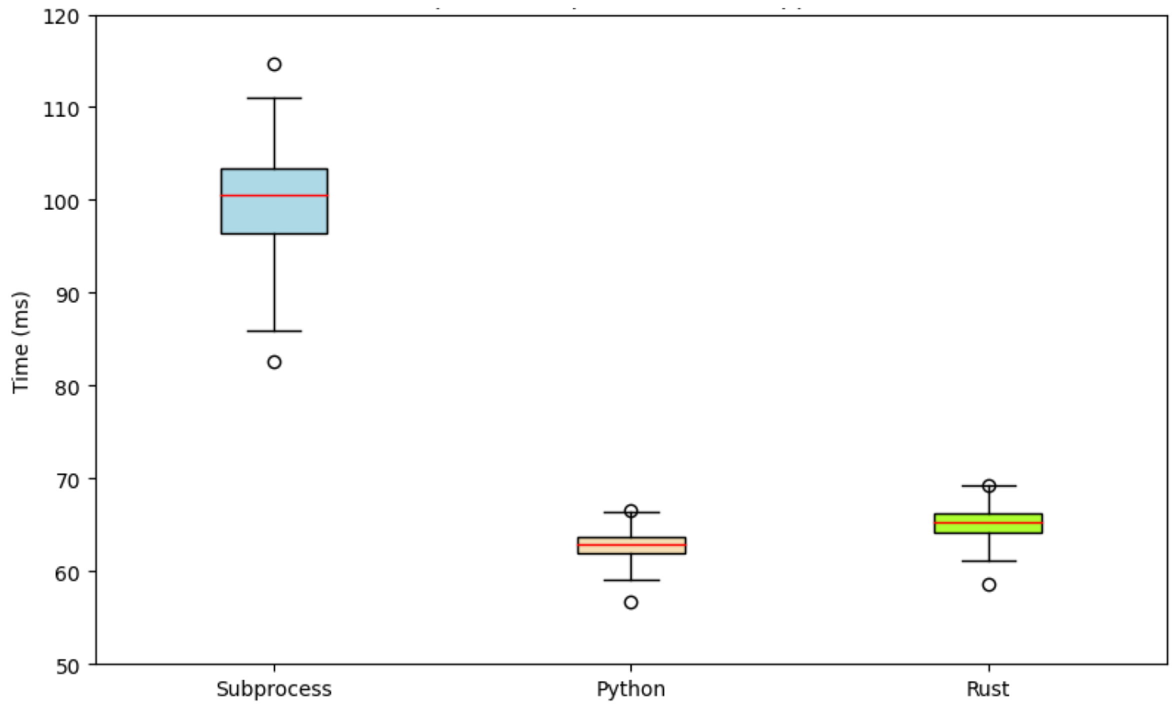
3 Alternatives

- **Subprocess**
  - Allows use of simple but flexible bash commands
- **Python**
  - Part of the algorithm is already in Python
- **Rust**
  - Attempt to leverage compiled language speed

Information Parsing

- Files contain unnecessary information, requiring parsing
- Two different scenarios:
  - State file – Complex
  - Reward file – Simple
- 3 Alternatives
  - **Subprocess**
  - **Python**
  - **Regex**

Information Collection and Parsing



Information Collection	Subprocess	Python	Rust
Average (ms)	49.637	12.805	15.107
Standard Deviation (ms)	± 4.990	± 1.385	± 1.546
Information Parsing	**Subprocess**	**Python**	**Regex**
State scenario (ms)	5.0318	0.0017	0.0014
Reward scenario (ms)	9.9792	0.0012	0.0018

- Fastest approaches were used to enhance DARA
- Biggest fault was due to Subprocess module

Average total time of each step:

- Base DARA 528.8 ms
- E-DARA 34.8 ms (≈94% decrease)

Simulation Methodology

Measure Experimentally → Normal Distribution of Delays → Simulated delay before action

Agent Mobile Node

SNR

Introduction

Methodology and Implementation

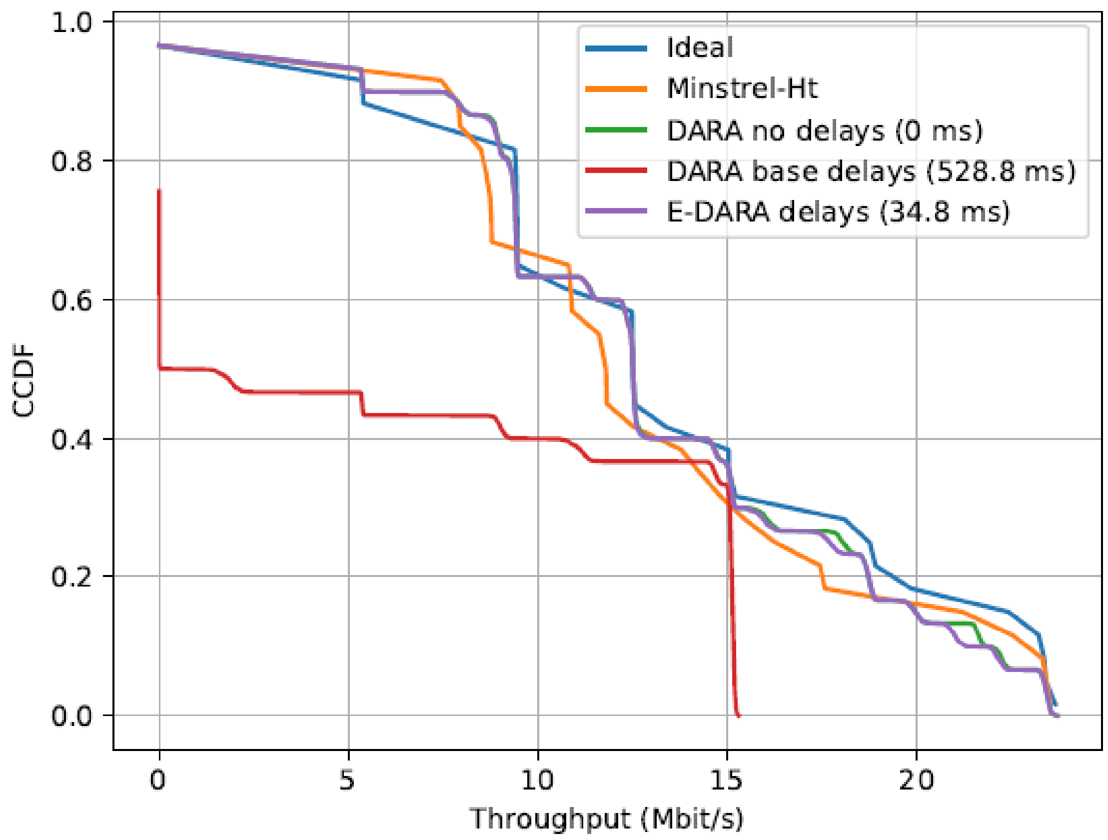
Results

Conclusions

DARA Comparisons

Perfect	Base	Enhanced	Simulation Trained
• No delays	• 528.8 ms delays	• 34.8 ms delays	• 34.8 ms delays (exploitation only)

Results



Algorithm	Average Throughput (Mbit/s)	Average frames lost
Ideal	13.27	—
Minstrel-HT	12.74	—
DARA no delays	13.04	1128.5
DARA base delays	6.44	4661.8
DARA enhanced delays	13.00	1189.0

- E-DARA achieves 102% higher throughput than base DARA
- E-DARA close to perfect version
- Computational delays severely affected performance

Simulation Training

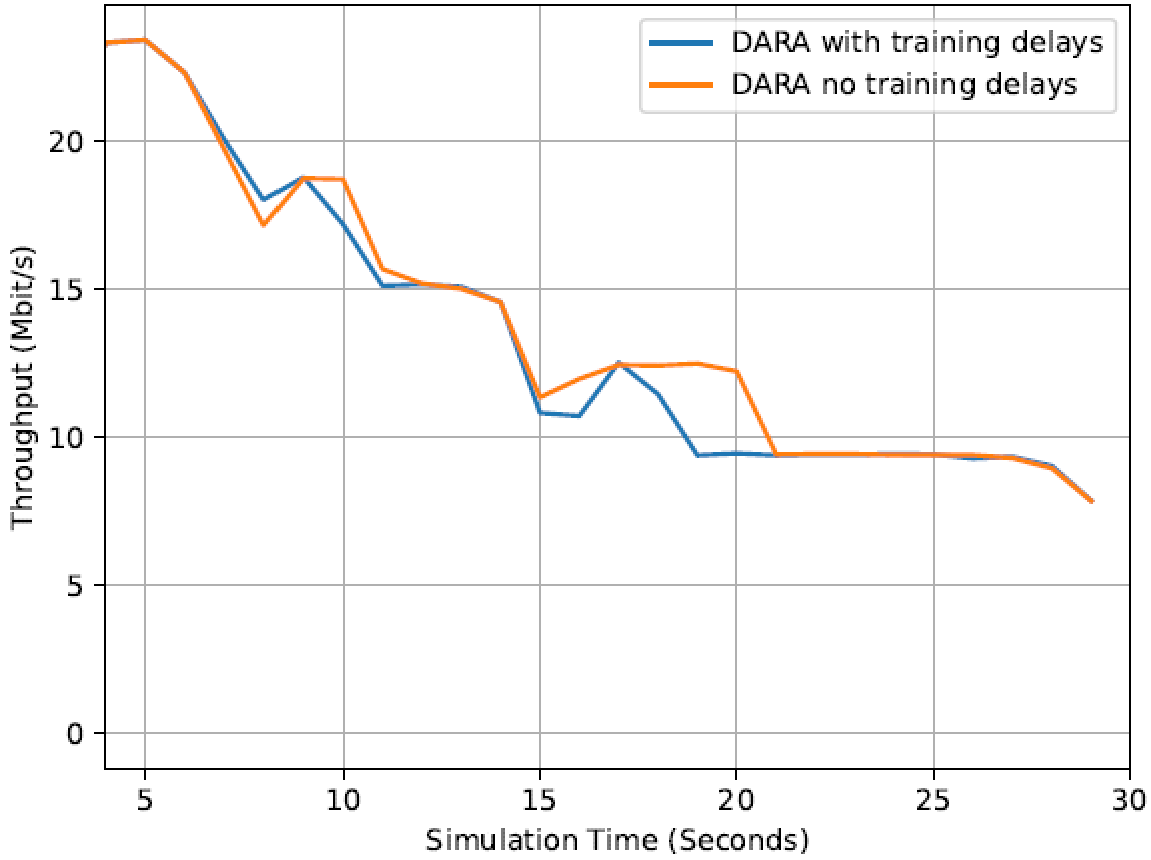
Simulation Training:

- Training stage performed in perfect conditions (no delays)
- Exploitation still remains with delays (34.8 ms).

Average throughput of E-DARA:

- Regular: 13.47 Mbit/s
- Simulation-Training: 13.83 Mbit/s (2.7% increase)

- May improve performance by reducing delay impact during training



Introduction

Methodology and Implementation

Results

Conclusions

Conclusions

- Computational delays are underdiscussed
- Simulations should consider delays
- Impact of delays can be significant although not apparent
- Future work can be done on analyzing different metrics, scenarios and algorithms

The End

Any Questions?

?