NetDev 0x15

PTP optimization using genetic algorithm

Marta Plantykow, Milena Olech, Maciej Machnikowski

Notices & Disclaimers

Performance varies by use, configuration and other factors. Learn more at www.Intel.com/PerformanceIndex.

Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See backup for configuration details. No product or component can be absolutely secure.

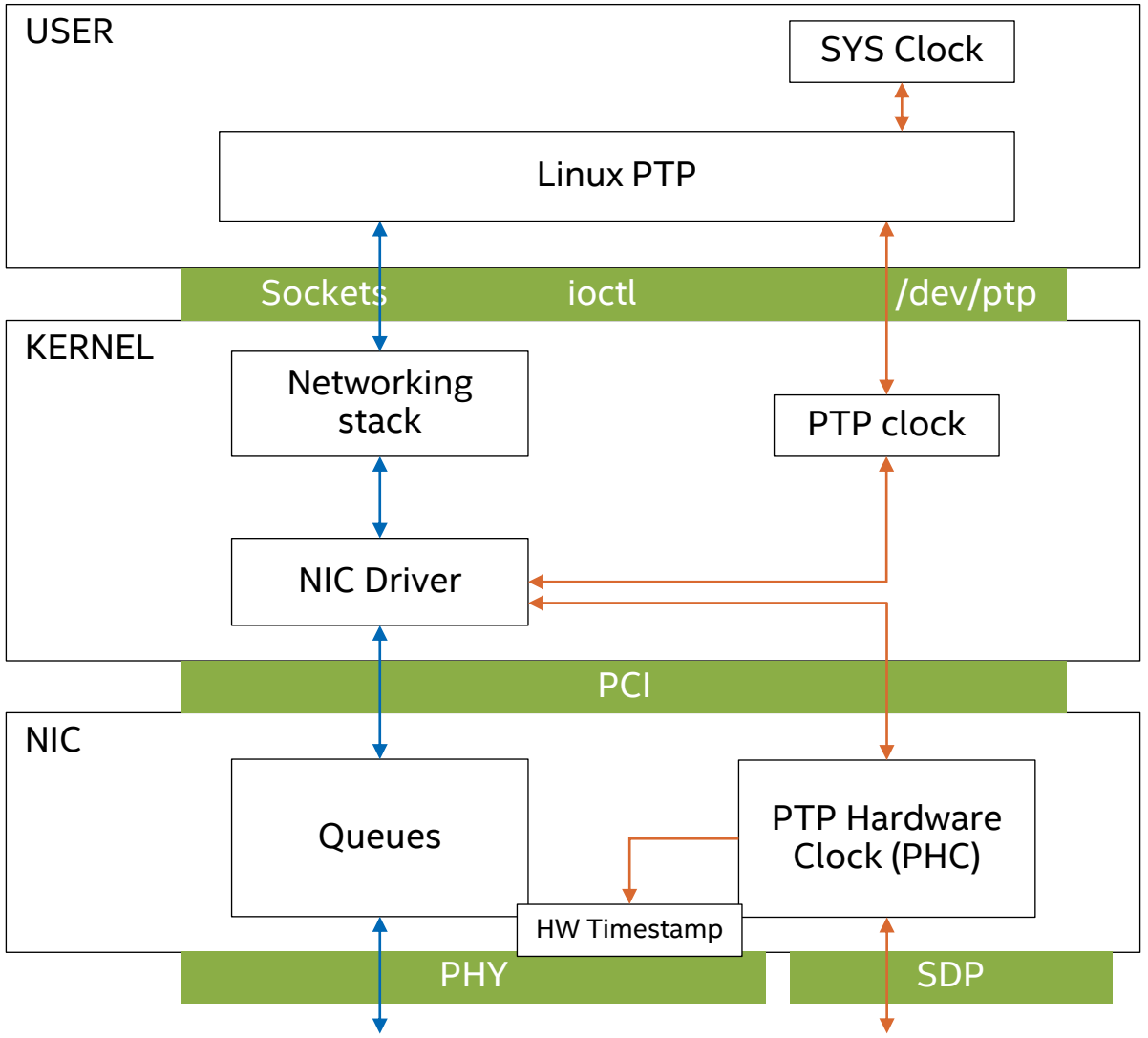
Your costs and results may vary.

Intel technologies may require enabled hardware, software or service activation.

Agenda

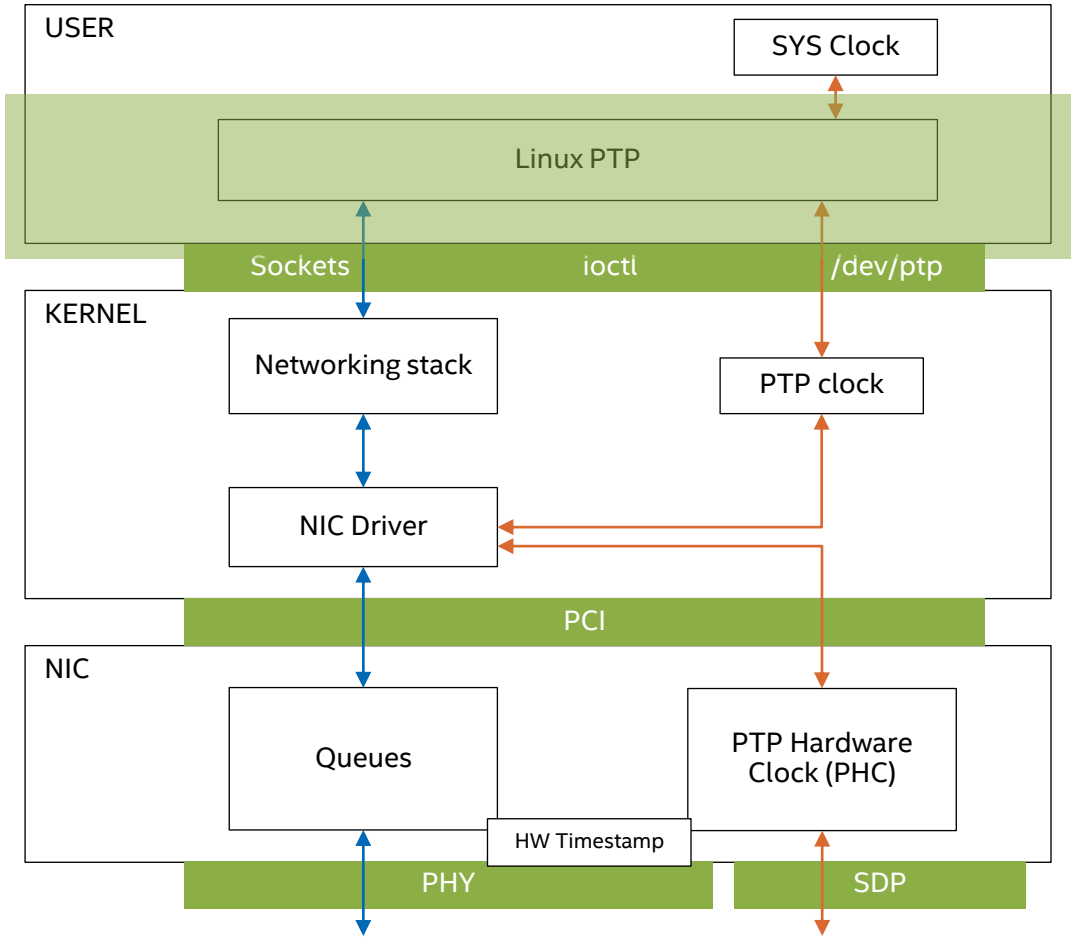
- PTP
- PID controller
  - Introduction
  - PID tuning
- Genetic Algorithms
- Data evaluation methods
- Developed framework details
- Results
- Future work

PTP on Linux

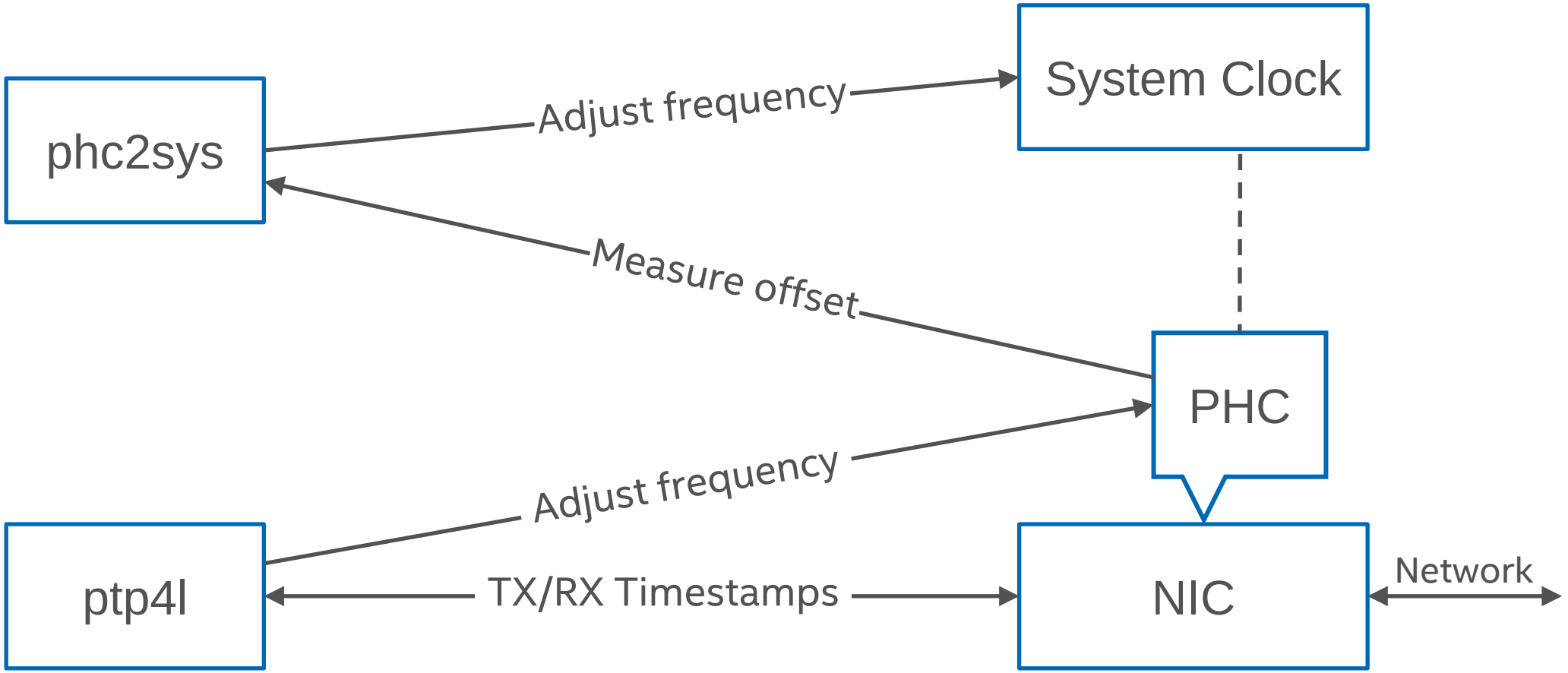

USER
SYS Clock
Linux PTP
Sockets
ioctl
/dev/ptp
KERNEL
Networking stack
PTP clock
NIC Driver
PCI
NIC
Queues
PTP Hardware Clock (PHC)
HW Timestamp
PHY
SDP

Linux PTP project

- Set of userspace tools
- Implement PTP (IEEE 1588)
- Use standard kernel APIs to synchronize and manage the clocks
- Supports HW and SW time stamping
  - Best precision w/ HW time stamping
- http://linuxptp.sourceforge.net/



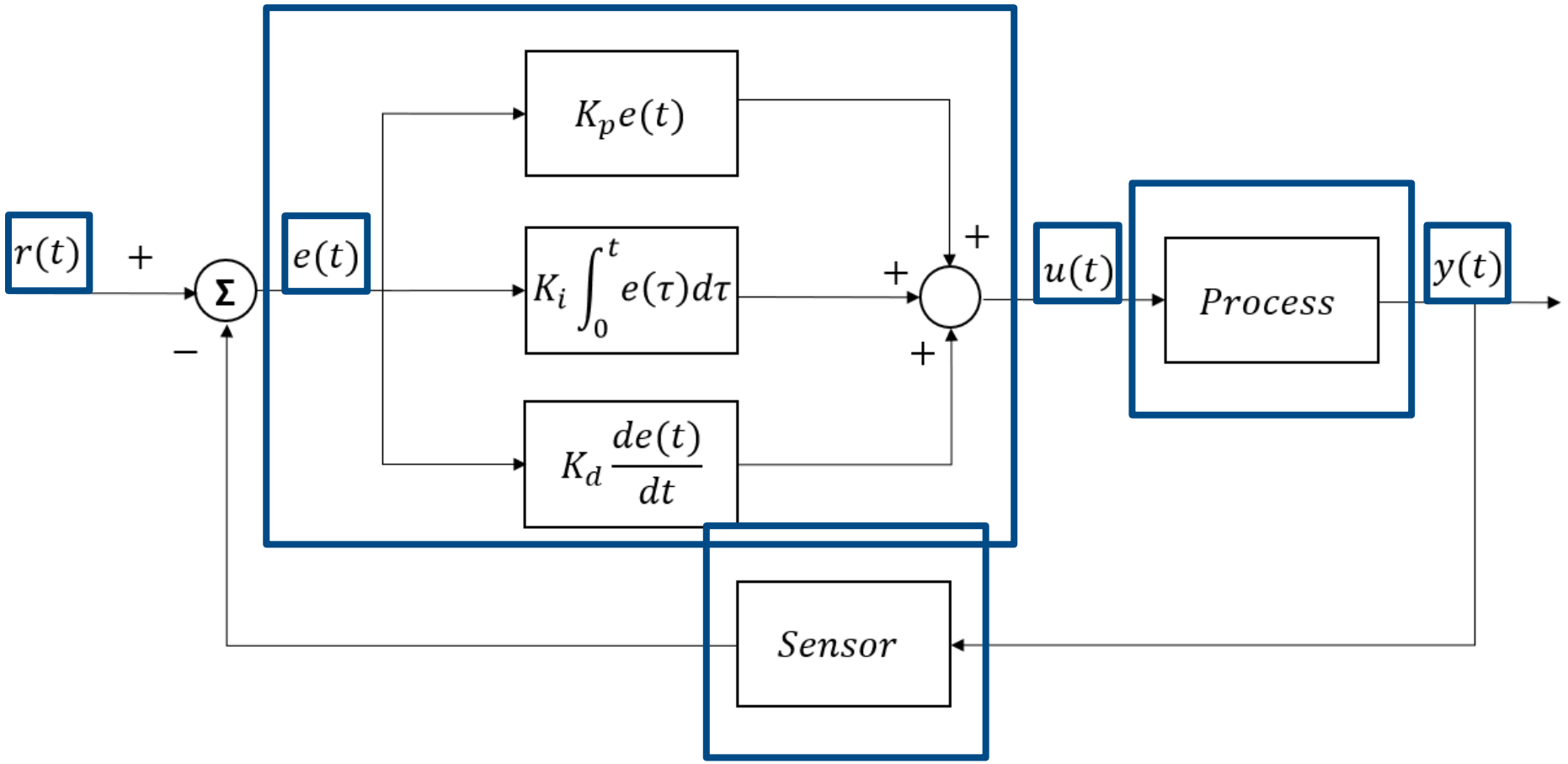
Linux PTP project



PID controller - Introduction

- Proportional-Integral-Derivative controller
- Commonly used to control feedback loops
- Uses error signal to control the object
- Three main parameters:
  - Proportional gain K_p
  - Integral gain K_i
  - Derive rate gain K_d

PID controller - Introduction



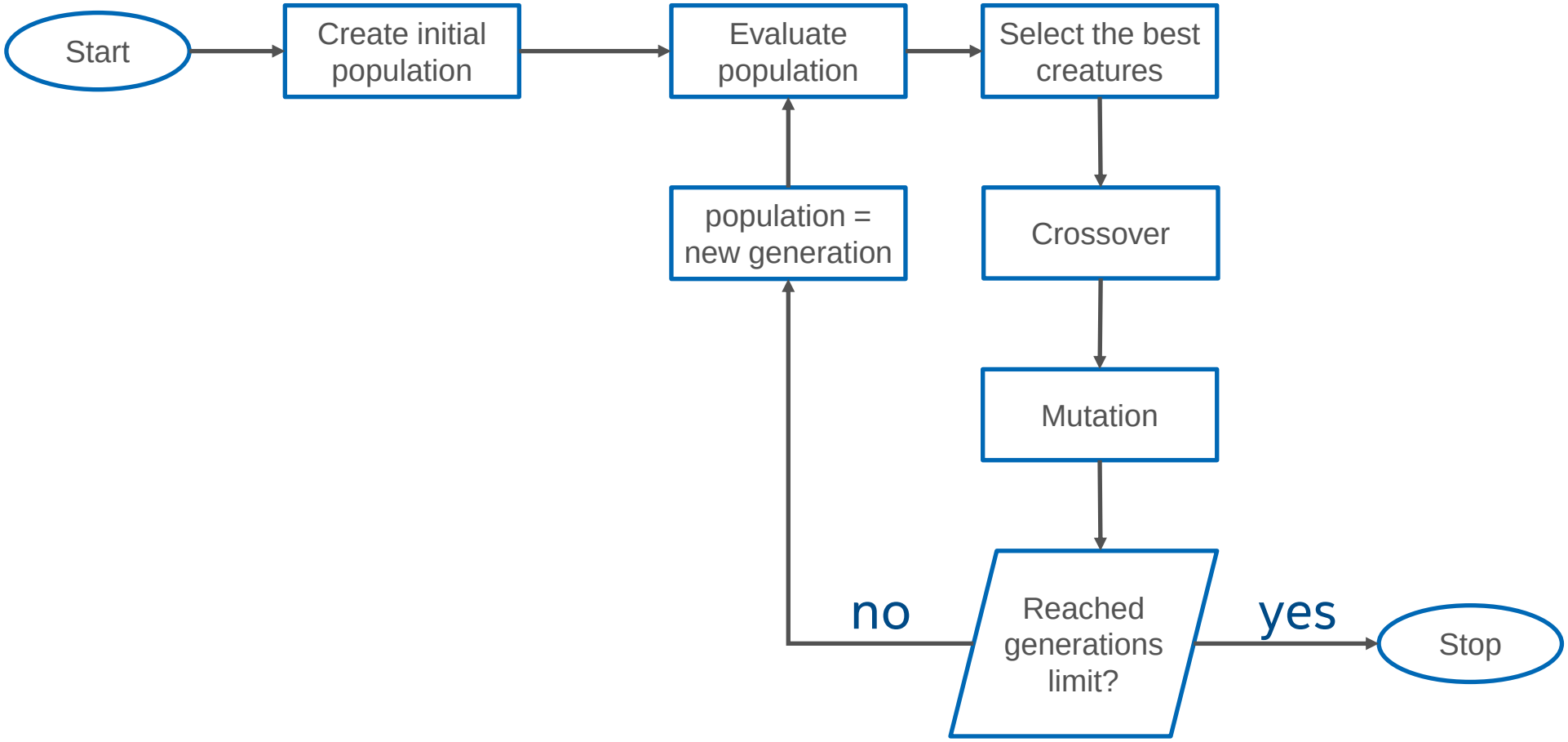
PID Controller – Tuning methods

- K_p K_i and K_d need to be carefully selected
- Behavior of the controller depends on parameters
- Incorrect parameters may destabilize the object
- Usually require manual tuning
- Difficult to find the right parameters

Genetic Algorithms

- Random search method that mimics natural evolution
- Depends on responses from the environment
- Reaches the best solution by:
  - Reproduction
  - Crossover
  - Mutation

Genetic Algorithms



Data evaluation methods

- Compare
  - expected values ($P_i, i = 1, 2, \ldots, n$)
  - observations ($O_i, i = 1, 2, \ldots, n$)
- Errors defined as:

$$e_i = P_i - O_i$$

- Model performance is based on sum of $\mathbf{e_i}$ $(i = 1,2, \ldots, n)$

Data evaluation methods

- Generic formula of average error can be represented by:

$$e_{\gamma}^{-} = [\sum_{i=1}^{n} w_i |e_i|^{\gamma} / \sum_{i=1}^{n} w_i]^{\frac{1}{\gamma}}$$

Data evaluation methods - RMSE

- $\gamma = 2$: Root Mean Square Error
- Most widely used to evaluate model performance
- Bigger errors have more significant impact on the result

$$RMSE = [n^{-1} \sum_{i=1}^{n} |e_i|^2]^{\frac{1}{2}}$$

Data evaluation methods - MAE

- $\gamma = 1$: Mean Absolute Error

$$MAE = n^{-1} \sum_{i=1}^{n} |e_i|$$

Test framework

Framework

1. Create initial random population (from a preconfigured ranges)
2. For each creature in population:
    1. Run tool under test with each creature's parameters
    2. Evaluate results
3. Select candidates for new generation
4. Crossover + mutation
5. Go back to #2 and repeat for each epoch

Framework – Data evaluation

- Match follower system's time to the leader time as closely as possible.
- Ideal scenario requires leader offset to be 0
- Calculate an error between expected values and observations

Framework – phc2sys

1. Set a fixed initial time offset
2. Synchronize system time to the PHC time
3. Use 20 reads/correction to minimize impact of bus contention
4. Use different P and I parameters for controller parameters
5. Run the tool for a fixed time
   - 80 s for short runs
   - 300 s for long runs
6. Parse tool output

Test results

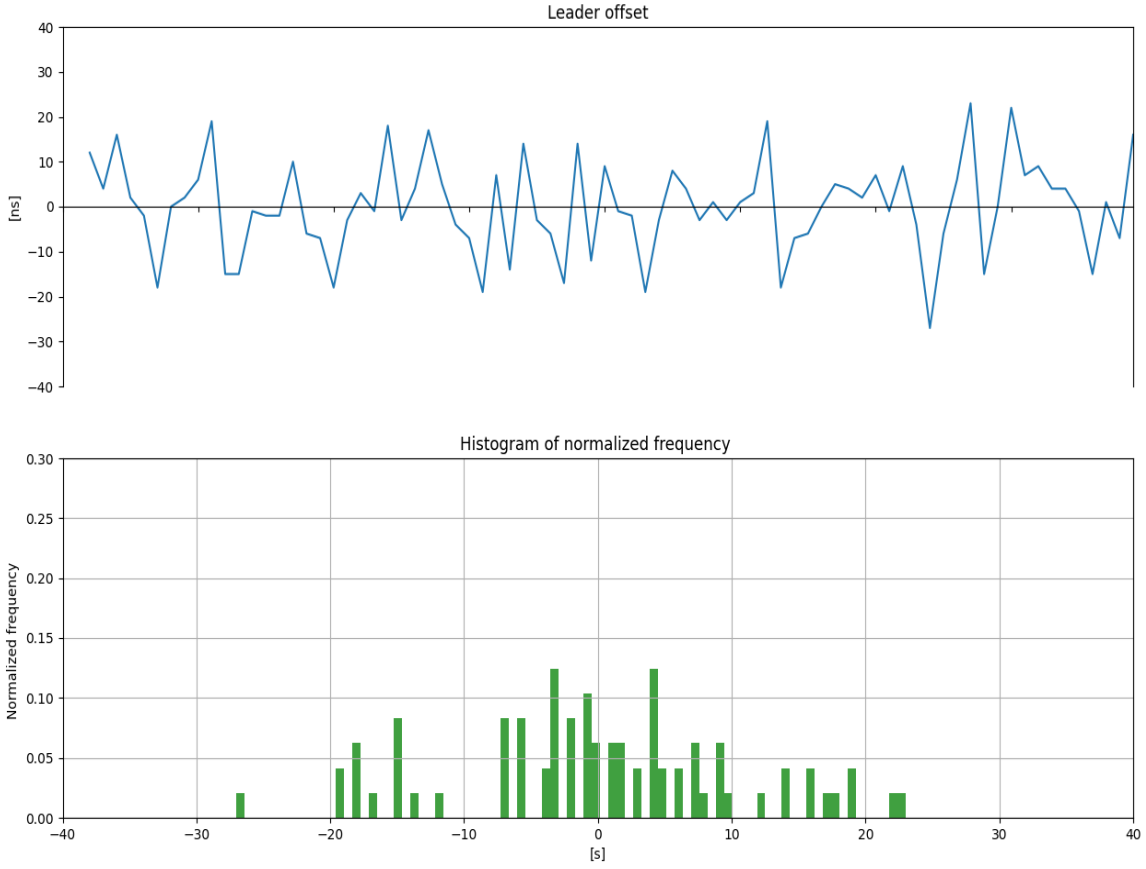
Results

1. Evaluate creatures using different metrics
   - MSE (1)
   - RMSE (2)
   - MAE (3)
2. Run algorithms with different number of epochs
   - 5
   - 10
   - 15

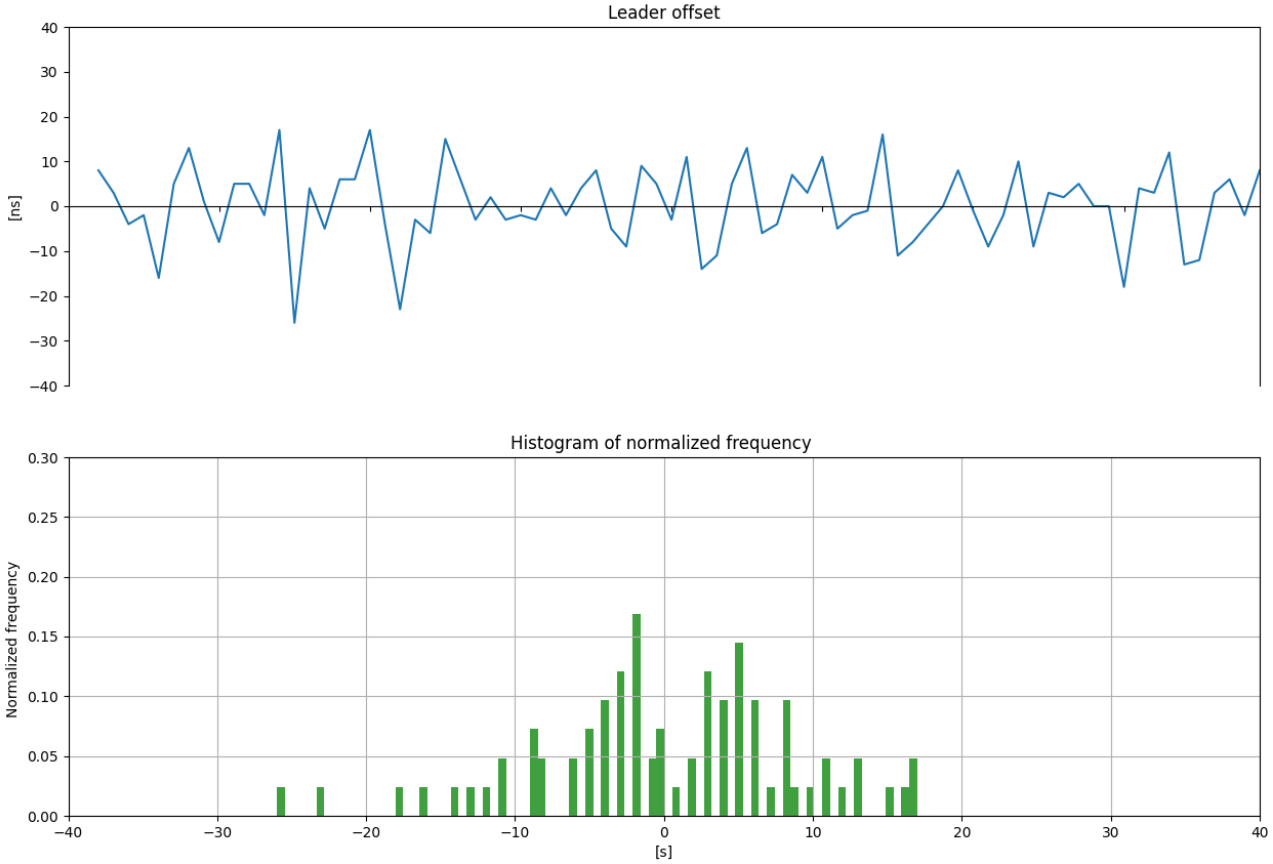
MSE

	Kp	Ki	Result
Default	0.7	0.3	110.26
Elite	0.59	0.93	75.08
		Improvement	32%

Default



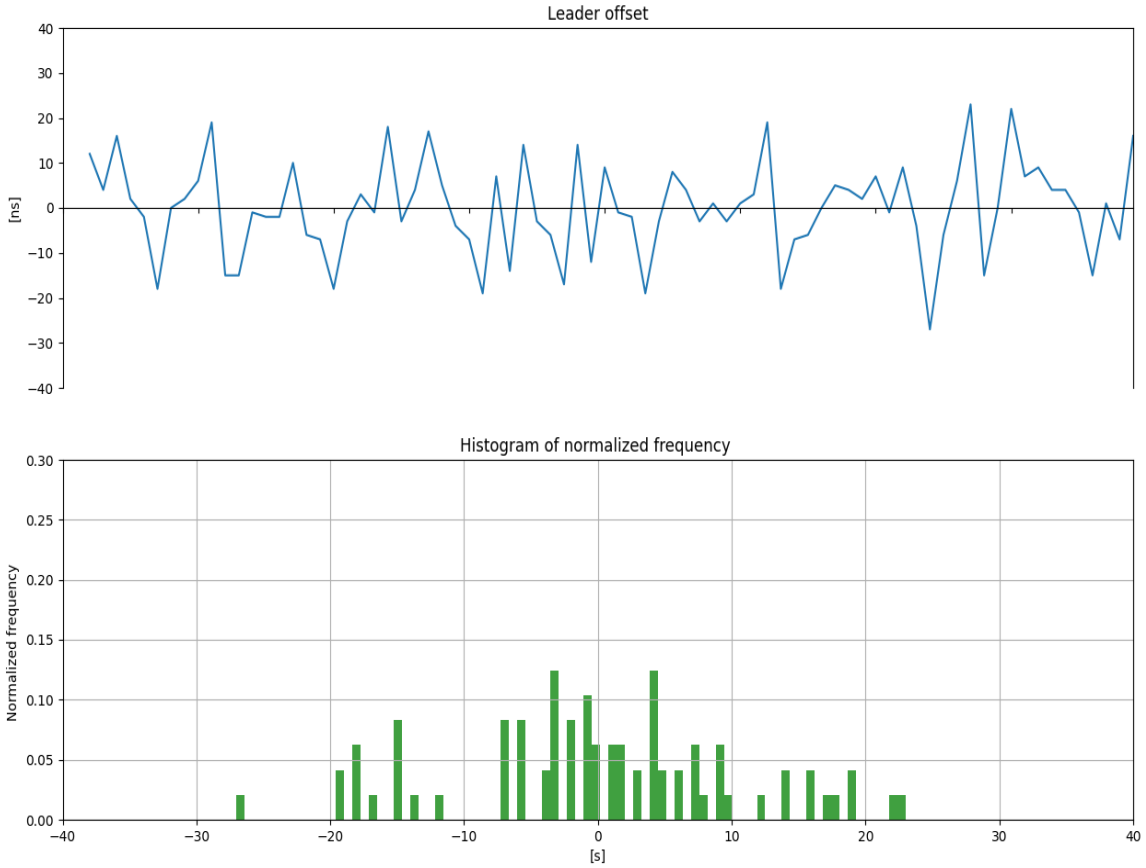
Elite



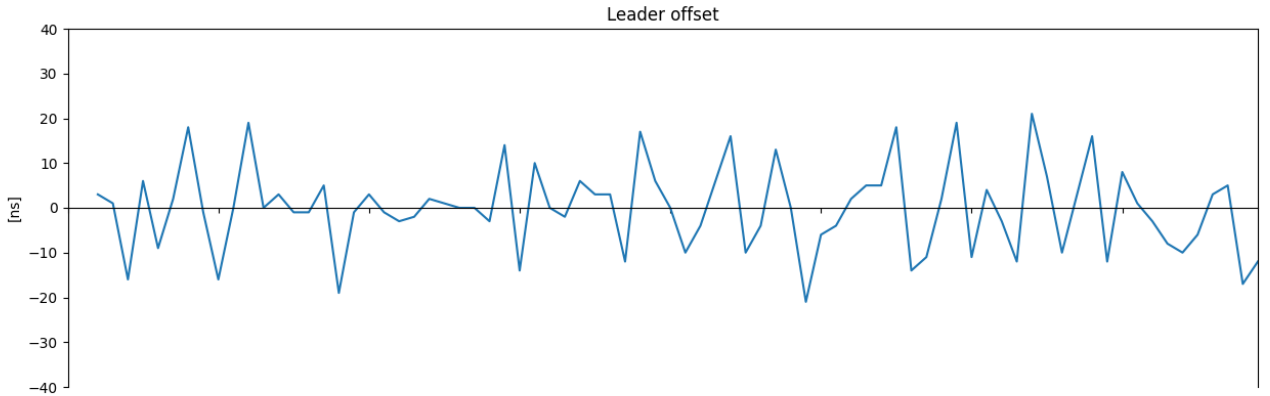
RMSE

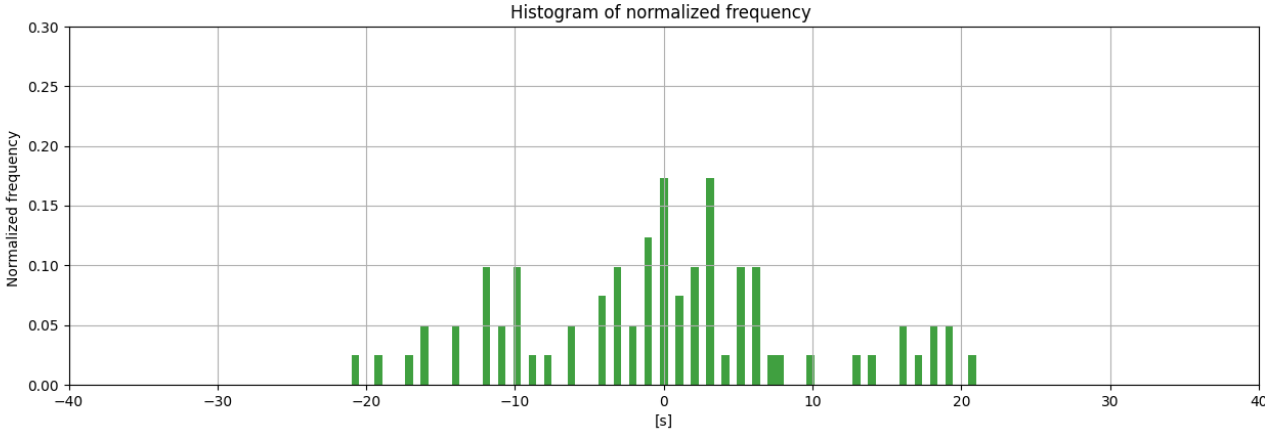
	Kp	Ki	Result
Default	0.7	0.3	11.68
Elite	0.09	0	8.98
		Improvement	23%

Default



Elite

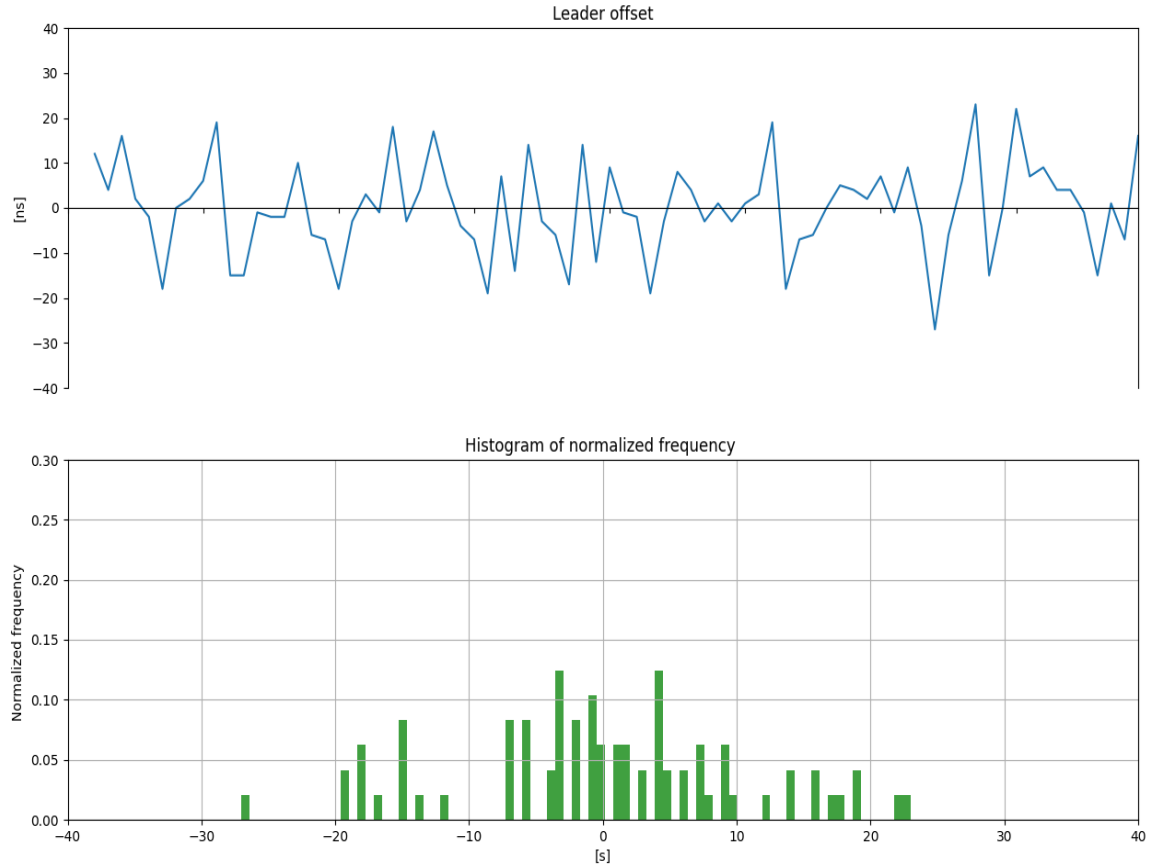




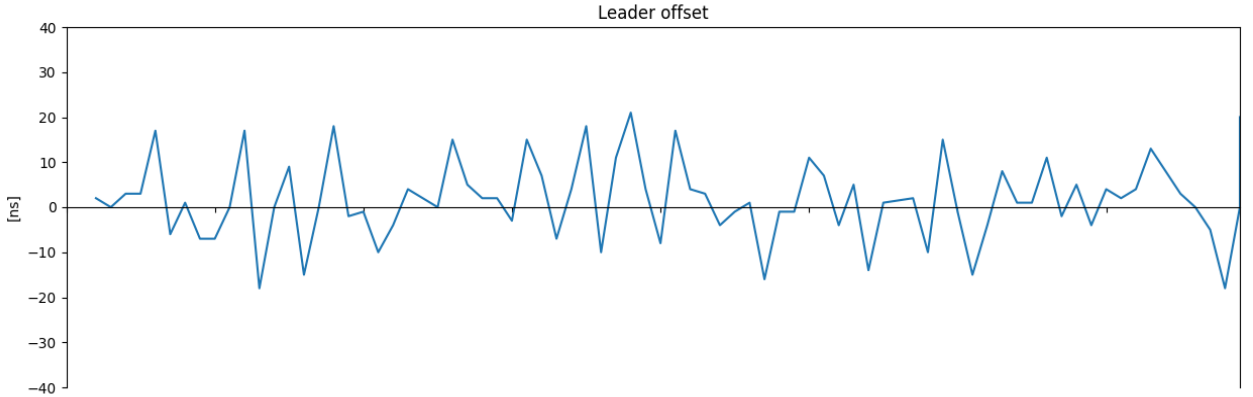
MAE

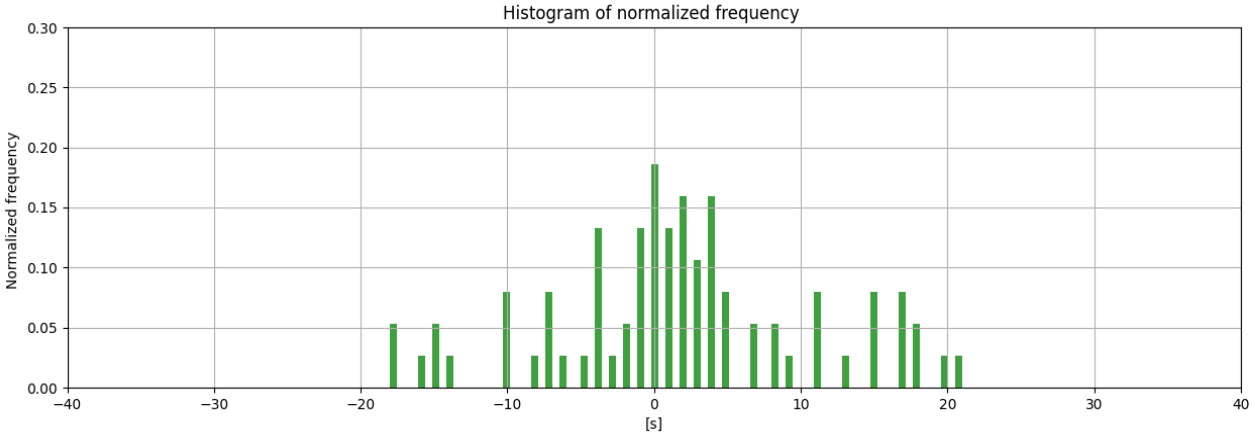
	Kp	Ki	Result
Default	0.7	0.3	8.99
Elite	0	2.51	7.32
		Improvement	19%

Default



Elite





Results (per generation)

Generation	elite Kp	elite Ki	MSE	% improvement over default
1	0.70	0.30	122.65	0%
2	10.68	0.00	112.60	8%
3	9.28	0.00	106.36	13%
4	1.30	0.00	111.71	9%
5	8.97	0.00	116.31	5%
6	9.64	0.00	107.27	13%
7	8.53	0.00	107.94	12%
8	1.99	0.00	107.34	12%
9	9.10	0.00	114.53	7%
10	9.28	0.00	106.36	13%
11	9.28	0.00	106.36	13%
12	6.89	0.00	110.94	10%
13	1.78	0.00	113.03	8%
14	1.40	0.00	105.45	14%
15	1.94	0.00	118.15	4%

Future work

Future work

- Coarse validation of stability of input parameters
  - Remove unstable configurations and replace with a different creature
- Test IEEE 1588 using ptp4l
- Longer test runs
- Find optimal number of epochs
- Find optimal elite size

Key takeaways

- PTP is crucial to meet strict Telco timing requirements
- LinuxPTP use PI controller to minimize errors
- Synchronization quality depend on PI parameters
- GA can optimize PI parameters
- Different goals can be reached using different metrics
- Tuned parameters result in 13-32% smaller errors default ones


intel