# Khronos: Middleware for Simplified Time Management in Cyber Physical Systems

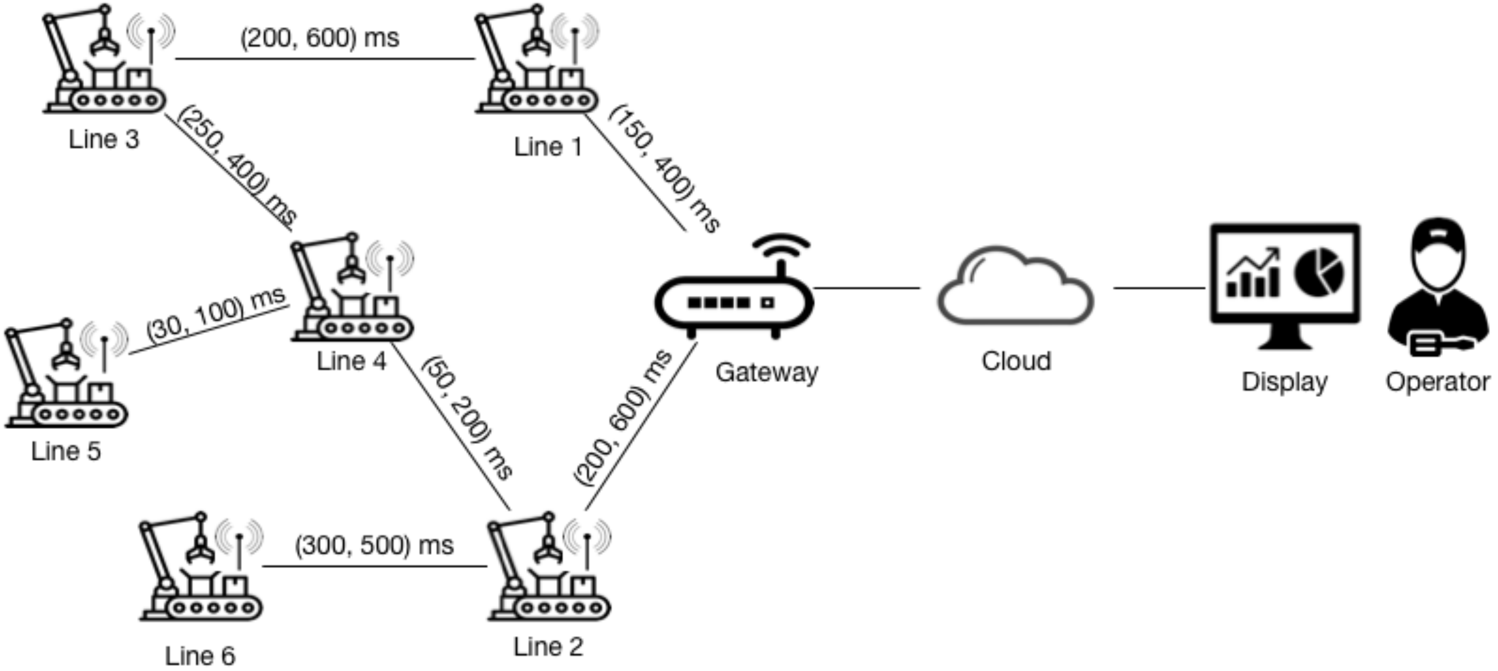
**Stefanos Peros**, Stéphane Delbruel, Sam Michiels, Wouter Joosen and Danny Hughes

KU LEUVEN DistriNet

DEBS' 19

# Industrial Use Case

› Fast-moving consumer goods company:






KU LEUVEN DistriNet

# Challenge

- Managing event arrival-time boundaries in CPS
  - **varying** network **latency**
    - wireless medium
    - packets propagate across different paths
  - **varying** packet **inter-generation delay**
    - clock drift

# State-of-the-Art

- Rely on **application developer**
  - static timeouts @ compile time
  - e.g. leased signals[1]

```
@lease(4000)
class FleetData extends Signal{
 //Definition of the FleetData signal omitted for brevity
}
```

# Problem Description

- Predicting time-boundaries at compile time
  - **impractical** (if not impossible)
    - CPS application developer != infrastructure expert
    - non-deterministic event arrival times

# Problem Description

- Predicting time-boundaries at compile time
  - impractical (if not impossible)
  - **inefficient**
    - waiting too long can fail to produce useful result
    - not waiting long enough may lead to faults
      - incomplete information

# Problem Description

- Application developers do **not know**
  - how long to wait for sensor packet arrivals

# Problem Description

- Application developers do **not know**
  - how long to wait for sensor packet arrivals
- But **do know**
  - how important it is to wait for sensor packet arrivals
    - before proceeding with complex event computation
    - % **completeness** constraint

# Timeliness vs Completeness

- **Trade-off**
  - Higher **completeness** constraint
    - larger timeouts
      - slower (re)actions (**timeliness**)
  - Lower completeness constraint
    - smaller timeouts
      - faster (re)actions

# Related Work

- ProbSlack[2]
  - adds **dynamic** offset to user-defined timeout
    - delay model
    - user tolerance $\delta$ for missed events (~ completeness)

# ProbSlack[2]

- Relies on **developer** to specify **@ compile time**
  - **timeout** (query frequency)
    - e.g. sampling periods can change at runtime
  - additional **configuration**
    - refresh period T for delay model(s)

# Research Problem

- State-of-the-art **time management** solutions for **CPS** rely heavily on the **application developer**
  - timeout specification @ compile time
  - user-defined parameter configuration

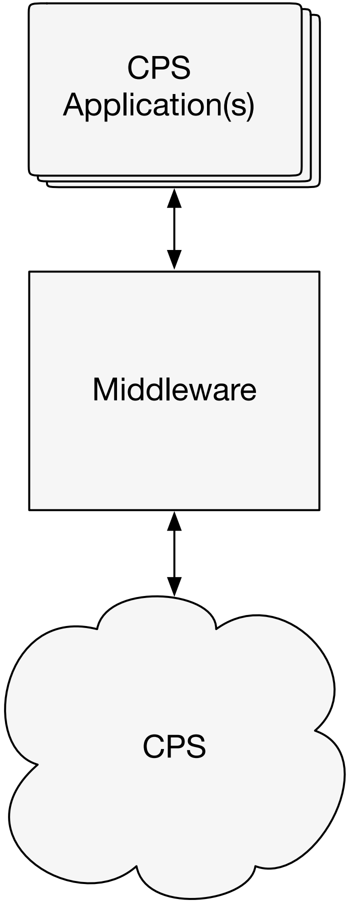
# Requirements for CPS Middleware

- **A.** Completeness constraint per device
- **B.** Not rely on developer
- **C.** Dynamism
- **D.** Heterogeneity
- **E.** Context

# Approach

# Khronos

- satisfy application completeness constraint(s)
- automatically determine timeout(s)
  - per sensor data stream
  - per completeness constraint
  - per packet arrival






KU LEUVEN DistriNet

# Prediction Technique(1/3)

- Inspired by TCP's Retransmission TimeOut (**RTO**)
  - **non-deterministic** ACK arrival times
    - varying network latency
  - **trade-off**: completeness vs timeliness
    - too long -> slow speed
    - too short -> unnecessary retransmissions

# Prediction Technique(2/3)

- **Timeout**

$$TO(t_i) = S(t_i) + K * \mathbb{V}(t_i) + D_T$$

- **Smoothed Arrival Time**

$$S(t_i) = \alpha S(t_{i-1}) + (1 - \alpha)R(t_i)$$

- **Smoothed Arrival Time Variance**

$$\mathbb{V}(t_i) = \beta \mathbb{V}(t_{i-1}) + (1 - \beta)|S(t_{i-1}) - R(t_i)|$$

# Prediction Technique(2/3)

- **Timeout**

$$TO(t_i) = \boxed{S(t_i)} + K * \boxed{\mathbb{V}(t_i)} + D_T$$

- **Smoothed Arrival Time**

$$S(t_i) = \alpha S(t_{i-1}) + (1 - \alpha)R(t_i)$$

- **Smoothed Arrival Time Variance**

$$\mathbb{V}(t_i) = \beta\mathbb{V}(t_{i-1}) + (1 - \beta)|S(t_{i-1}) - R(t_i)|$$

# Prediction Technique(3/3)

- Lightweight
  - **O(n)**, where n the number of completeness constraints
  - **10 operations** to compute next timeout
    - 5 multiplications + 5 additions
- Simple
  - no configuration **post** deployment (**req. B**)

# Sensitivity Factor K

- **K** = f(**constraint**)
- offline mapping
  - ~ 3 weeks of network monitoring
- smallest K that satisfies given constraint
  - overprovision **x2**

$$TO(t_i) = S(t_i) + \boxed{K} * \mathbb{V}(t_i) + D_T$$

# API(1/2)

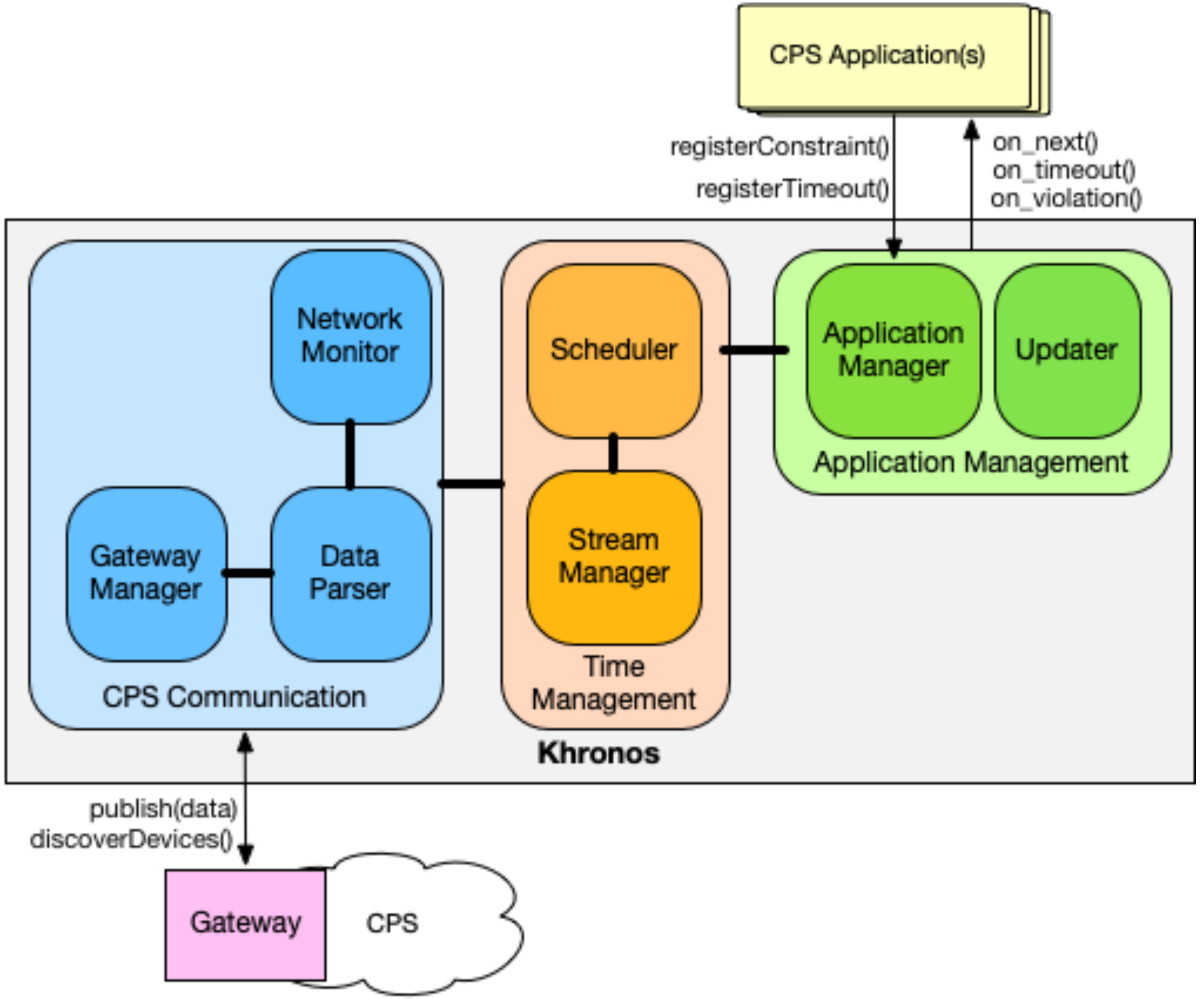
- register constraint (**req. A**):
  - `registerCompleteness(device, constraint, on_next, on_timeout, on_violation)`
- register (static) timeout:
  - `registerTimeout(device, timeout, on_next, on_timeout)`

# API(2/2)

- Three **callback** methods (**req. E**):
  - `on_next(value, timeout, completeness)`
    - packet arrives before timeout
  - `on_timeout(timeout, completeness)`
    - timeout occurs before packet arrival
  - `on_violation(value, timeout, completeness)`
    - completeness < constraint

# Architecture

› Three layers



# Implementation

# Network

- Wireless mesh
  - 33 devices (20 sensors)
- SmartMesh IP
  - broadly used in IIoT & CPS applications
  - TSCH(default), CSMA/CA
  - self-forming & self-maintaining

# Middleware

- Raspberry Pi 3
- Python v3.6
  - flask (REST)
  - Pyro 4.6 (RMI)
- CoAP & websocket
  - gateway communication

# Evaluation

# Evaluation

- **Performance** of predicted time windows
  - network & application **dynamism** (req. C)
    - 4 experiments
  - network & application **heterogeneity** (req. D)
    - 4 experiments

# Metrics (1/2)

- Prediction Error (**PE**)

$$PE_{d,\rho} = \frac{1}{n}\sum_{k=1}^{n} distance(p_k, to_k), \quad distance(p_k, to_k) = abs(p_k - to_k)$$

  - *d*: device, *ρ*: constraint, $p_k$: k'th arrival time, $to_k$: k'th timeout
  - measured in **seconds**
  - ↓PE → ↑timeliness

# Metrics (2/2)

- Constraint Violation % (**CV%**)
- ***ρ* satisfied** when:
  - ***completeness* ≥ *ρ*,** over 99.999% of the time
  - *completeness*: fraction of packets that arrive before timeout
    - measured as moving average
- if *ρ* = 1.0, best-effort



KU LEUVEN DistriNet

# Alternative Approaches

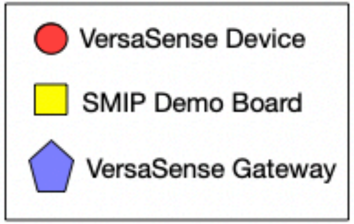
- Double Sampling Period (**DSP**)
  - $TO(t_i) = 2 * (Sampling\ Period)$
- Sampling Period Network Delay (**SPND**)
  - $TO(t_i) = (Sampling\ Period) + avg(latency)$
- Static Timeout Oracle (**STO**)
  - $TO(t_i, \rho) = smallest\ timeout\ that\ satisfies\ \rho$
  - **theoretical**, reference benchmark

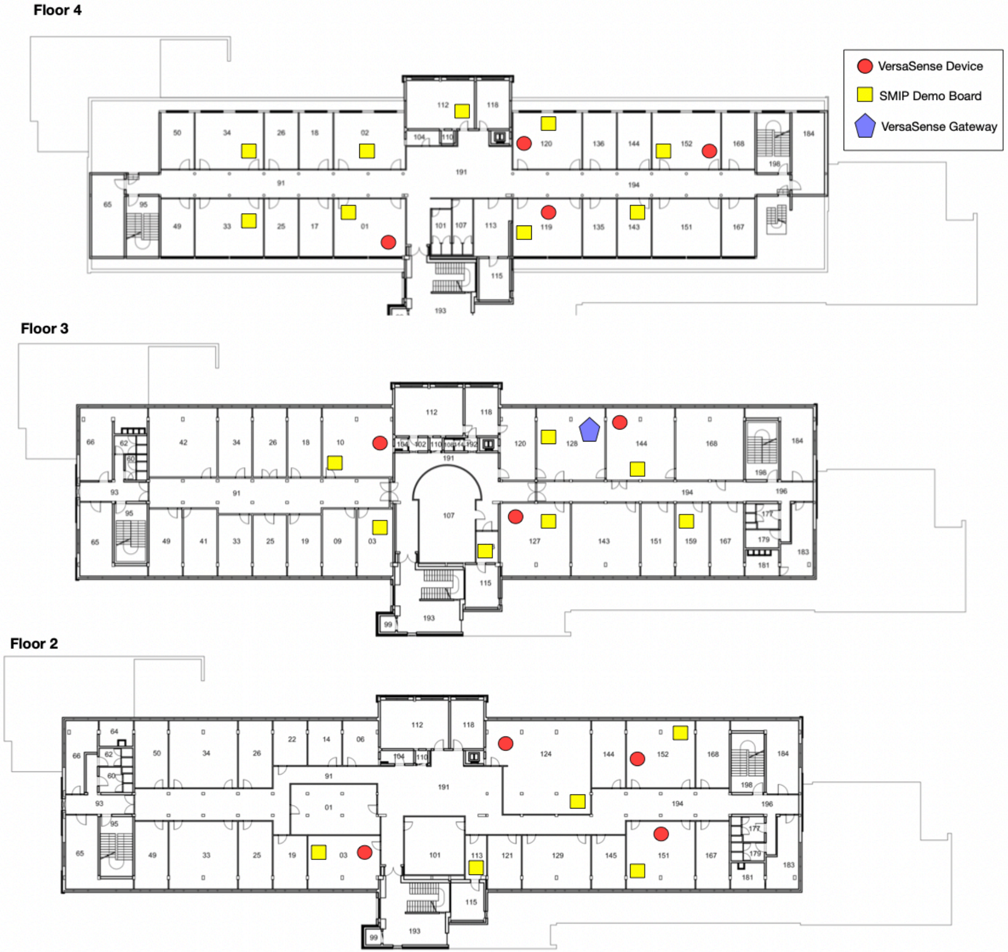
# Default Topology

› Gateway in Floor 3

**Table 1: Deployed peripherals and their settings.**

| Identifier | Peripheral Type | Quantity | Sampling |
|---|---|---|---|
| 3302/5500 | Sensor (Presence) | 1 | 10s |
| 9803/9805 | Sensor (Light) | 3 | 120s |
| 3303/5702 | Sensor (Temperature) | 3 | 120s |
| 8040/8042 | Sensor (Pressure) | 3 | 60s |
| 9903/9904/2 | Sensor (Thermocouple) | 1 | 10s |
| 1010/9000 | Sensor (Battery) | 10 | 900s |



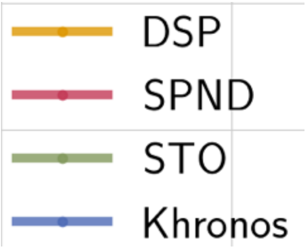



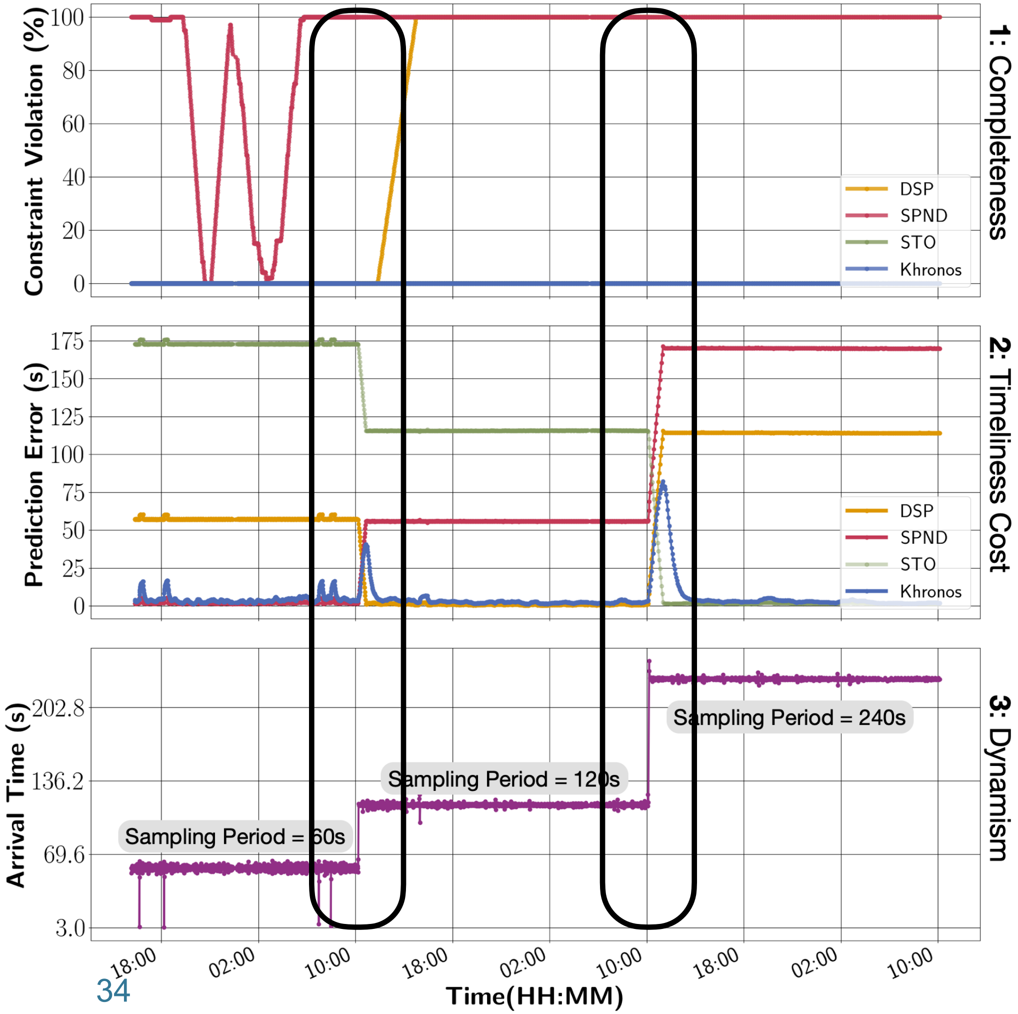
# Dynamism

- **Sampling Period**
- Network Size
- Network Latency

# Sampling Period

- 60s→120s→240s
- every ~24 hours
- $\rho$ = 0.8
- default topology



# Heterogeneity

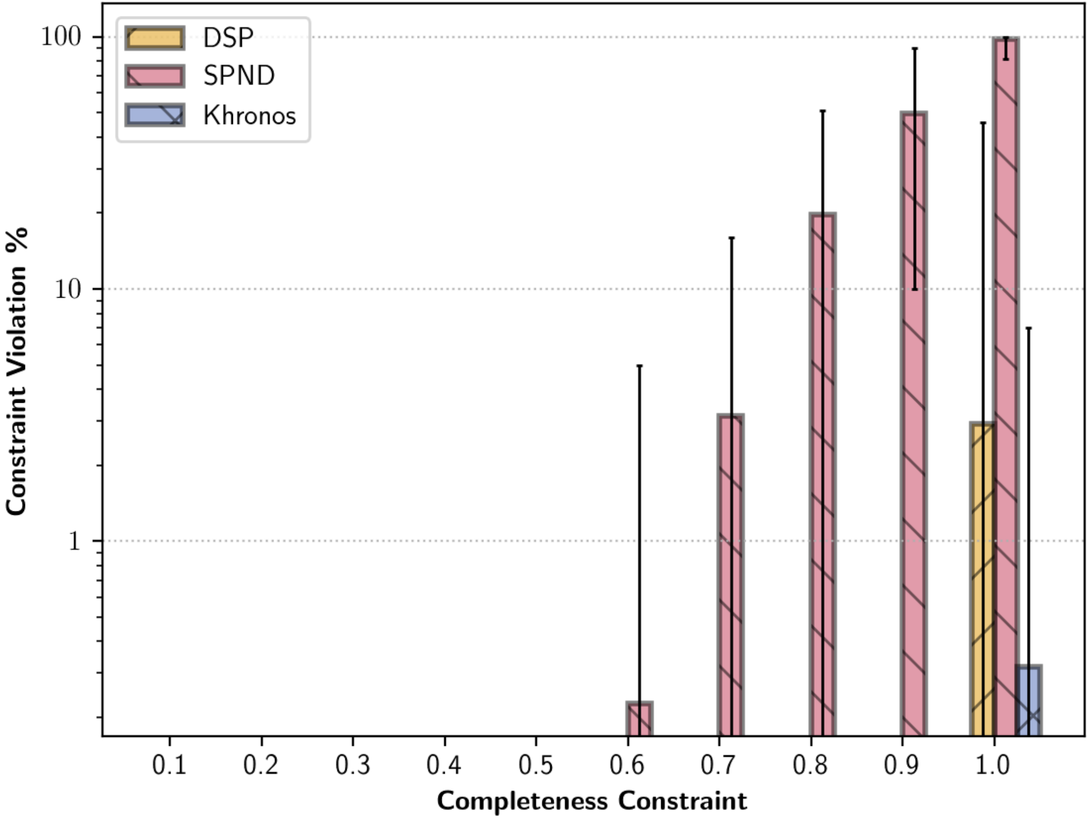
- **Range of Completeness Constraints**
- Medium Access Control Protocol
- Sampling Period
- Network topology

# Range of Completeness Constraints(1/3)

- $\rho \in$ <0.1, 0.2, … 1.0>
- default topology
- default sampling periods

# Range of Completeness Constraints(2/3)

- **Constraint Violation %**
- SPND violates $\rho >= 0.6$
- $\rho = 1.0$
  - Khronos ~ 0.32%
  - 3x less than DSP

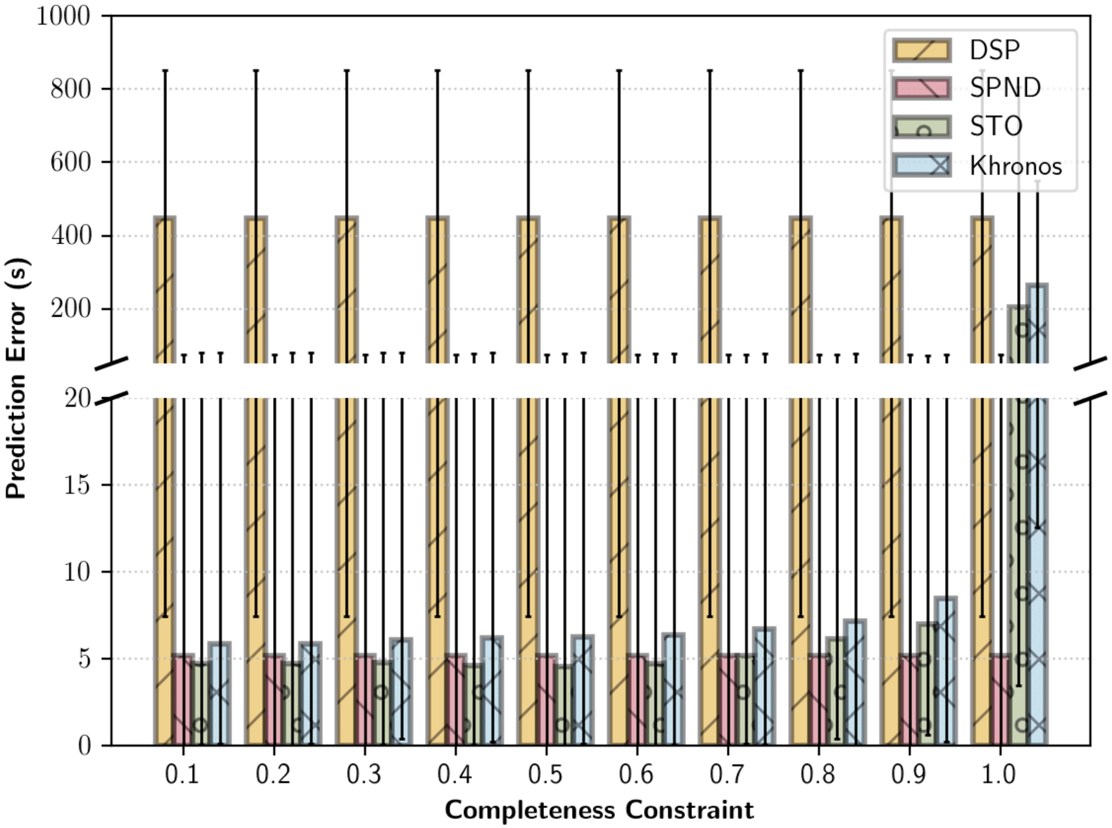





KU LEUVEN DistriNet

# Range of Completeness Constraints(3/3)

- **Prediction Error (s)**
- PE(Khr) < PE(DSP)
- PE(Khr) ~ SPND/STO
- $\rho$ = 1.0
  - PE(Khr) < PE(DSP)
  - CV(Khr) < CV(DSP)






KU LEUVEN DistriNet

# Conclusion

# Conclusion

- CPS integrated with critical physical processes
  - e.g. manufacturing, healthcare, smart grids
- reacting **timely** under **complete** information is **crucial**
  - **heterogeneity** and **dynamism**
    - platform, network and application

# Conclusion

- Khronos
  - trade-off **timeliness vs completeness** in CPS applications
  - specification of completeness **constraints**
  - **automatically** determine timeouts
    - improve timeliness
    - lift burden of manual timeouts from developer

# Conclusion

- Extensive evaluation on physical testbed
  - dynamism
  - heterogeneity
- Khronos outperforms alternative approaches
  - **consistent** constraint satisfaction
  - **smaller** timeouts
    - up to two order(s) of magnitude

# Thank you!

Email: stefanos.peros@cs.kueluven.be
Repository: https://github.com/mazerius/khronos

# References


- 1. Florian Myter, Christophe Scholliers, and Wolfgang De Meuter. 2017. Handling partial failures in distributed reactive programming. In Proceedings of the 4th ACM SIGPLAN International Workshop on Reactive and Event-Based Languages and Systems (REBLS 2017). ACM, New York, NY, USA, 1-7.
- 2. Rivetti, Nicolo & Zacheilas, Nikos & Gal, Avigdor & Kalogeraki, Vana. (2018). Probabilistic Management of Late Arrival of Events. 52-63. 10.1145/3210284.3210293.

# References

› 3. Christophe De Troyer, Jens Nicolay, and Wolfgang De Meuter. 2017. First-class reactive programs for CPS. In Proceedings of the 4th ACM SIGPLAN International Workshop on Reactive and Event-Based Languages and Systems (REBLS 2017). ACM, New York, NY, USA, 21-26. DOI: https://doi.org/10.1145/3141858.3141862

# References


- 4. Kensuke Sawada and Takuo Watanabe. 2016. Emfrp: a functional reactive programming language for small-scale embedded systems. In Companion Proceedings of the 15th International Conference on Modularity (MODULARITY Companion 2016). ACM, New York, NY, USA, 36-44. DOI: https://doi.org/10.1145/2892664.2892670

[download]


- 

# Future Work

# Future Work

- Online training for sensitivity factor K
  - smaller deployment overhead
  - e.g. incremental learning, control theory, …
- Reactive Programming
  - suitable for CPS application development[3,4]
  - integrate Khronos API with ReactiveX framework(s)

# Motivation

- why RTO?
  - durable solution
  - on top of wide, heterogeneous, dynamic infrastructure
  - lightweight
    - 2x EWMA (SRTT and SAT)

# API(3/3)

- register constraint

```
// register 25% completeness constraint for device 'LightSensor1'
// update average light value when packet arrives within timeout
// create pop-up on screen when timeout occurs
// write error message to log file when constraint is violated
registerCompleteness('LightSensor1', 0.25, updateAverage(data),
    alert('Timeout!'), logger.write('Constraint Violation!'))
```

- register (static) timeout

```
// register static timeout of 40 seconds for device 'LightSensor1'
// update average light value when packet arrives within timeout
// create pop-up on screen when timeout occurs
registerTimeout('LightSensor1', '40s', updateAverage(data),
    alert('Timeout!'))
```

# Network

- Real-life SMIP testbed
- 33 devices
  - 1x VersaSense Gateway (M01)
  - 10x VersaSense wireless devices (P02)
    - 20x peripherals (sensors)
  - 22x SMIP motes (DC9003A-B)
    - forward sensor data

# Middleware(2/2)

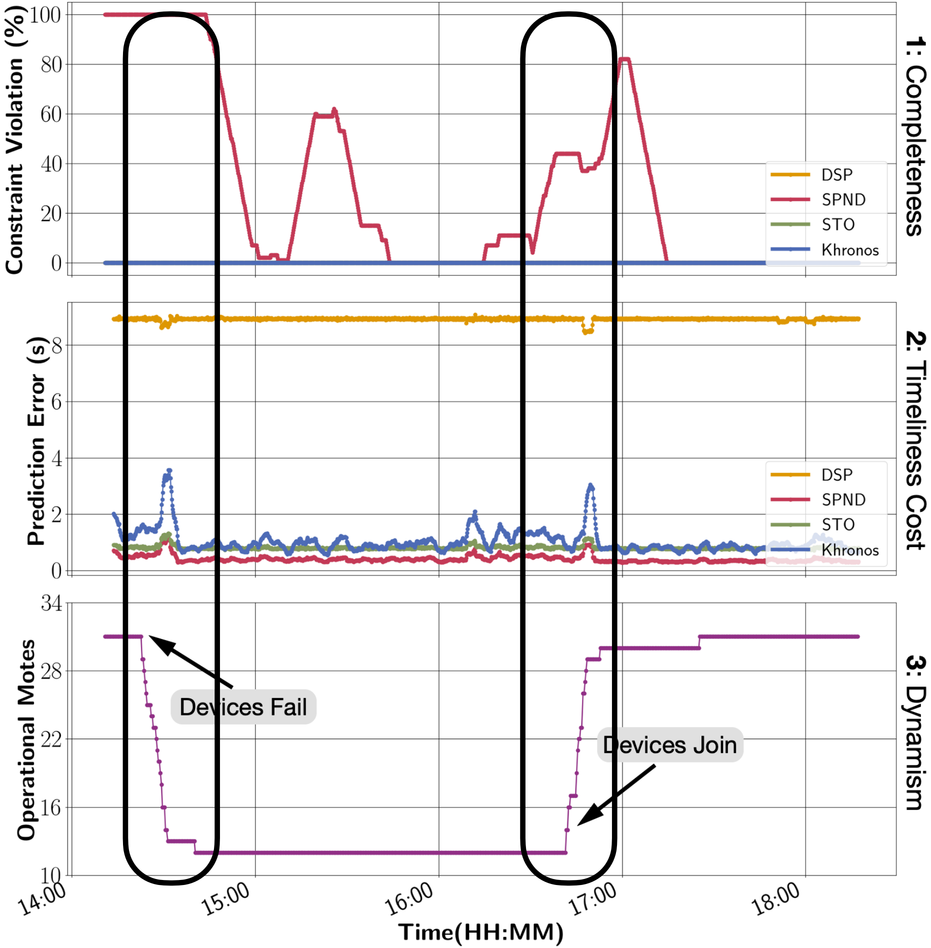
- resulting K
- based on TSCH
- same values used for CSMA/CA

Table 2: K values for different completeness constraints $\rho$.

| $\rho$ | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 | 1.0 |
|---|---|---|---|---|---|---|---|---|---|---|
| **K** | 0 | 0.1 | 0.6 | 1 | 1.2 | 1.4 | 2 | 2.8 | 4.6 | 300 |

# Network Size

- reduced up to 66.67%
- turn off devices
- $\rho = 0.8$
- default topology
- sampling period = 10s

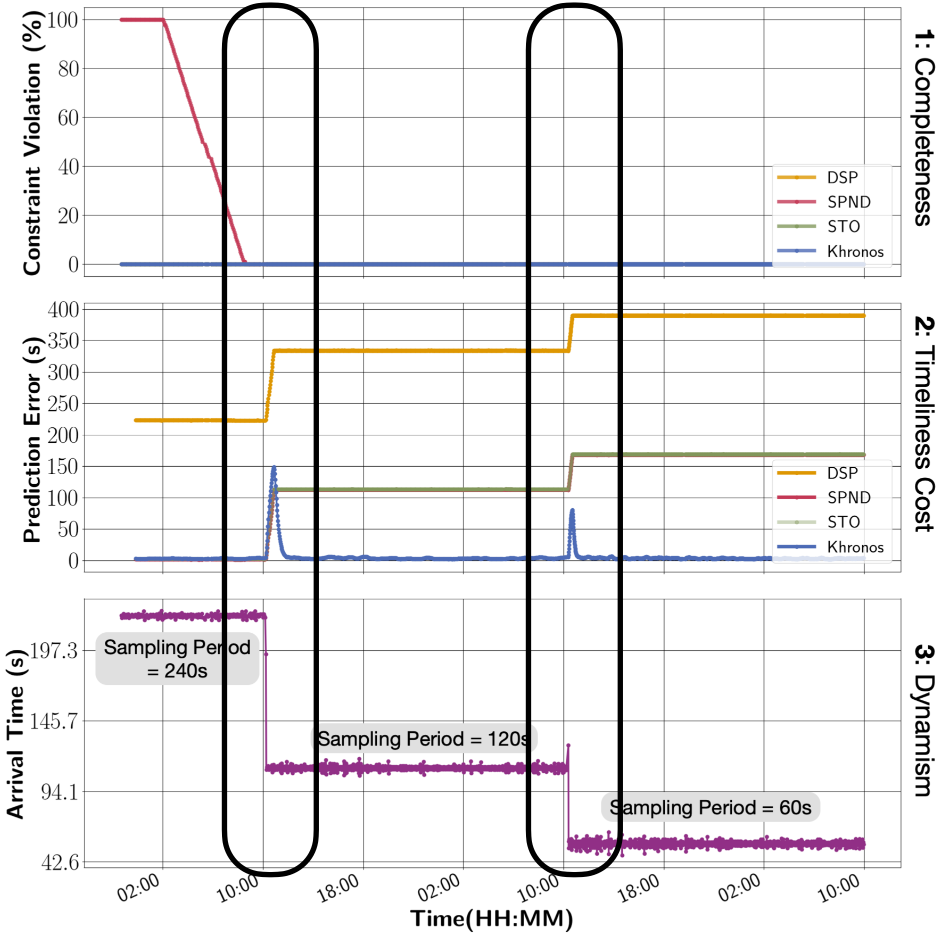





KU LEUVEN DistriNet

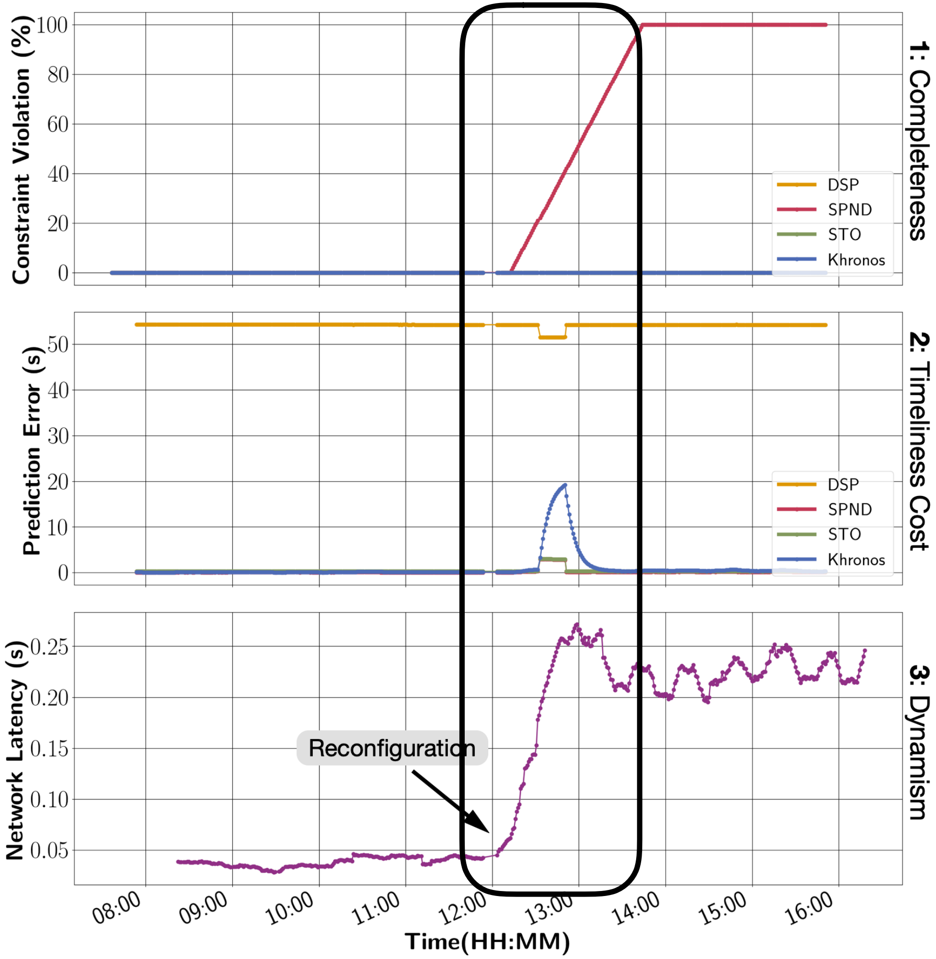
# Sampling Period(2/2)

- 240s→120s→60s
- every ~24 hours
- $\rho$ = 0.8
- default topology



# Network Latency

- basebw, bwmult
- requires network reset
- $\rho = 0.8$
- default topology
- sampling period 60s



# Medium Access Control(1/3)

- TSCH
- CSMA/CA
- ~ 72 hours per MAC protocol
  - ~ 2 million packets @ gateway
- all devices within 1 meter of gateway

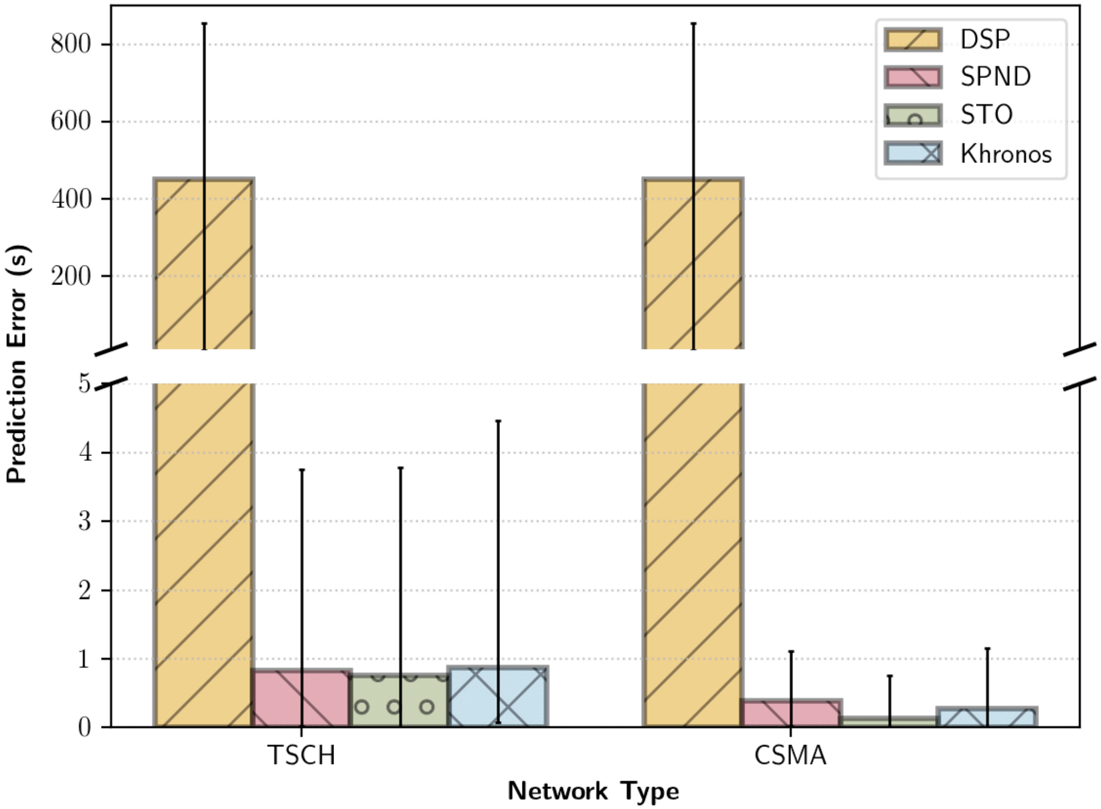
# Medium Access Control(2/3)

- **Constraint Violation %**
- $\rho = 0.8$
- only SPND fails constraint

| Approach | TSCH | CSMA/CA |
|---|---|---|
| DSP | 0% | 0% |
| SPND | 27.8% | 40% |
| STO | 0% | 0% |
| Khronos | 0% | 0% |

# Medium Access Control(3/3)

- **Prediction Error (s)**
- PE(Khr) < PE(DSP)
- PE(Khr) ~ SPND, STO






KU LEUVEN DistriNet

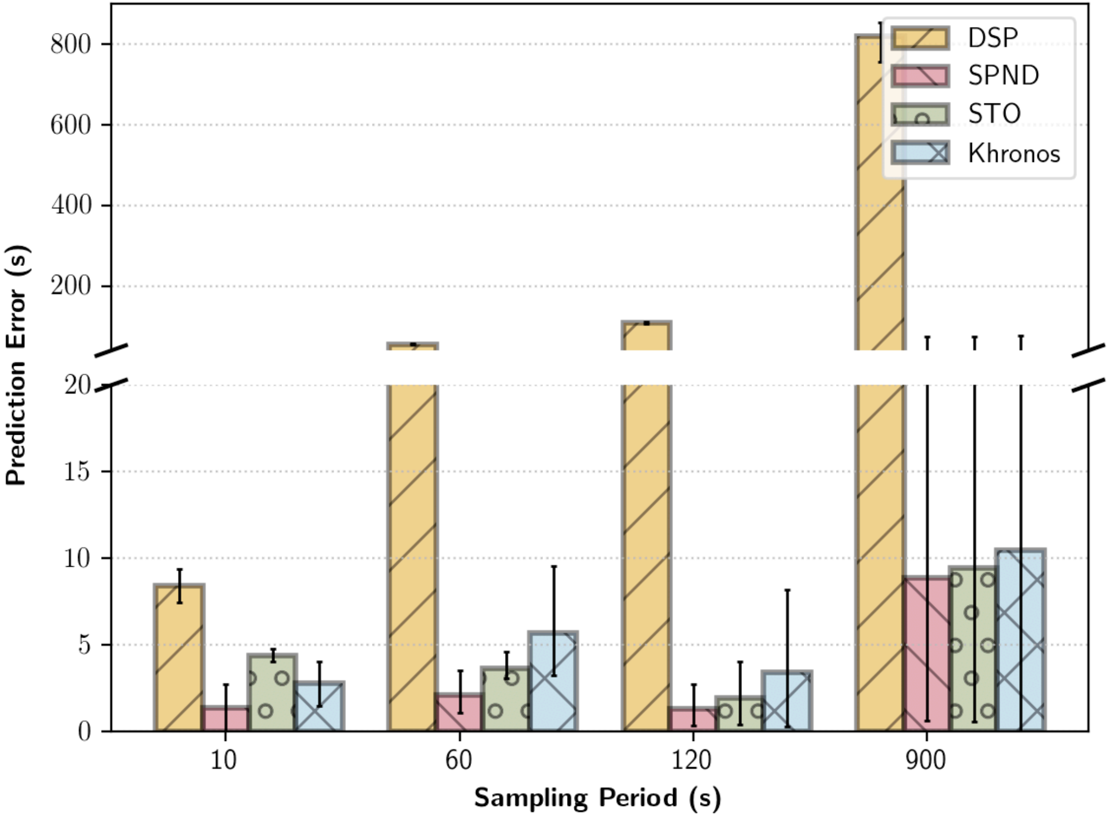
# Sampling Period(1/2)

- **Constraint Violation %**
- $\rho = 0.8$
- default deployment
- sampling periods:10s, 60s, 120s, 900s
- SPND always fails constraint

| Approach | 10s | 60s | 120s | 900s |
|---|---|---|---|---|
| DSP | 0% | 0% | 0% | 0% |
| SPND | 21.5% | 20.3% | 25.16% | 16.18% |
| STO | 0% | 0% | 0% | 0% |
| Khronos | 0% | 0% | 0% | 0% |

# Sampling Period(2/2)

- **Prediction Error (s)**
- PE(DSP) > PE(Khr)
  - ∝ sampling period
- PE(Khr) ~ SPND, STO






KU LEUVEN DistriNet

# Network Topology(1/3)

- Two topologies
  - topology **A**: within 1 meter of the gateway
  - topology **B**: up to two floors away from gateway
- ~ 72 hours of data per topology
  - ~ 2 million packets @ gateway

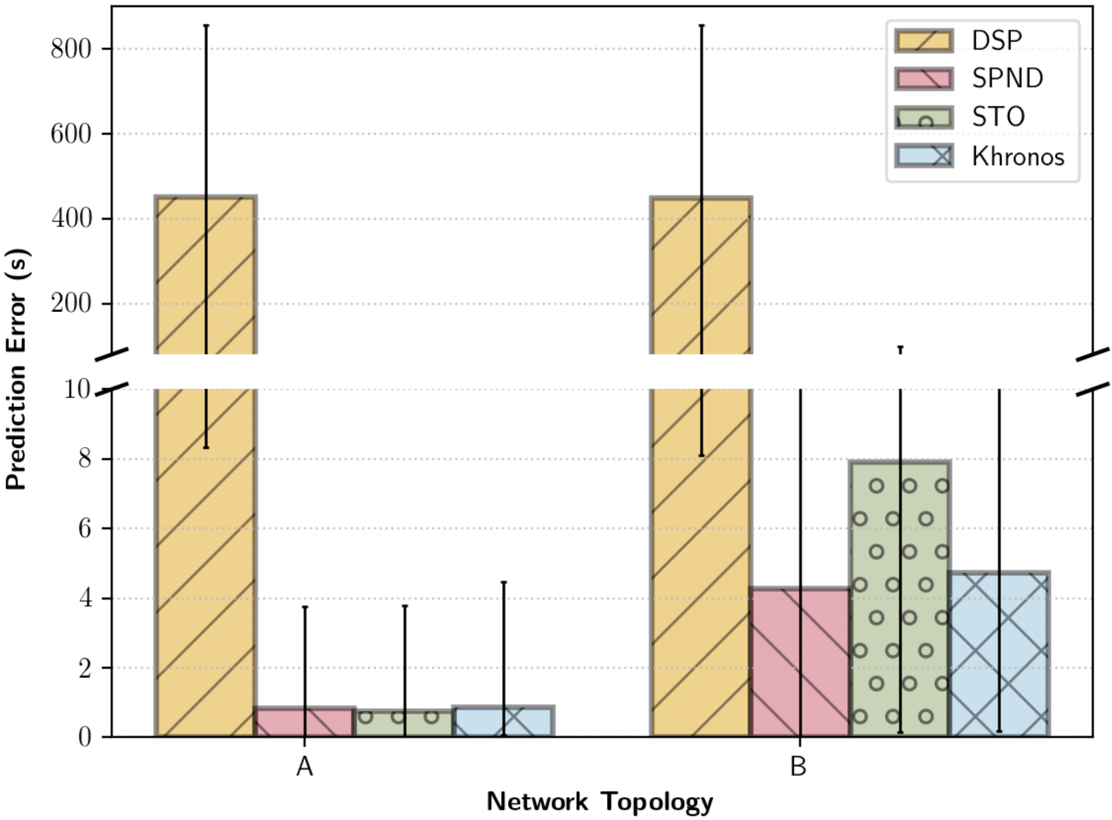
# Network Topology(2/3)

- **Constraint Violation %**
- $\rho = 0.8$
- default sampling rates
- SPND & DSP violate the constraint

| Approach | Topology A | Topology B |
|---|---|---|
| DSP | 0% | 0.045% |
| SPND | 27.8% | 42.8% |
| STO | 0% | 0% |
| Khronos | 0% | 0% |

# Network Topology(3/3)

- **Prediction Error (s)**
- $\rho = 0.8$
- PE(DSP) > PE(Khr)
- PE(Khr) ~ SPND, STO






KU LEUVEN DistriNet