Khronos: Middleware for Simplified Time Management in CPS

Stefanos Peros
imec-DistriNet, KU Leuven
Leuven, Belgium
stefanos.peros@cs.kuleuven.be

Stéphane Delbruel
imec-DistriNet, KU Leuven
Leuven, Belgium
stephane.delbruel@cs.kuleuven.be

Sam Michiels
imec-DistriNet, KU Leuven
Leuven, Belgium
sam.michiels@cs.kuleuven.be

Wouter Joosen
imec-DistriNet, KU Leuven
Leuven, Belgium
wouter.joosen@cs.kuleuven.be

Danny Hughes
imec-DistriNet, KU Leuven
Leuven, Belgium
danny.hughes@cs.kuleuven.be

Abstract

Cyber Physical Systems (CPS) combine communication, computation and data storage capabilities to oversee and control physical processes in domains including manufacturing, medical monitoring and smart grids. CPS behavior can be remotely monitored by aggregating event data from various sensors, forwarded over wireless networks. One of the main challenges for CPS application developers is to manage event arrival-time boundaries and to trade off between timeliness and completeness: waiting too long until all events arrive can fail to produce a useful result, while not waiting long enough may lead to faults because the status information is incomplete. Monitoring the production lines in a factory, for example, depends on the aggregation of event data from multiple sensors in the distributed CPS, such as temperature and movement. Yet, predicting time-boundaries for individual event arrivals is difficult, if not impossible, for an application developer, because the wireless network and the sensing devices introduce latencies which vary continuously along with the load, status or environmental conditions of the network and the sensors. This paper proposes Khronos, a middleware that automatically determines timeouts for event arrivals that improve timeliness, given completeness constraint(s) specified by the CPS application developer and taking into account variations in event propagation delays. Extensive evaluations on a physical testbed show that Khronos considerably improves timeliness under varying network configurations and conditions, while satisfying application-specific completeness constraints.

CCS Concepts

• **Information systems** → *Stream management*; • **Networks** → *Network monitoring*; • **Computer systems organization** → *Embedded and cyber-physical systems*; • **Software and its engineering** → **Middleware**;

Keywords

Cyber-physical systems, Late arrivals, Middleware, Time management

ACM Reference Format:
Stefanos Peros, Stéphane Delbruel, Sam Michiels, Wouter Joosen, and Danny Hughes. 2019. Khronos: Middleware for Simplified Time Management in CPS. In *DEBS '19: The 13th ACM International Conference on Distributed and Event-based Systems (DEBS '19), June 24–28, 2019, Darmstadt, Germany.* ACM, New York, NY, USA, Article 4, 12 pages. https://doi.org/10.1145/3328905.3329507

1 Introduction

Physical objects are being enhanced with computational capabilities to serve as components of larger distributed cyber physical systems (CPS) [3]. The objective to digitally transform industries, including manufacturing, transport & logistics and utilities [8, 13, 16, 20], builds upon distributed CPS infrastructures and the possibility to remotely monitor CPS behavior by aggregating event data from various sensors, typically forwarded over a wireless network.

CPS applications often rely on sensors that measure a physical property of the environment, at a specified sampling period. However, packets can still be generated at different rates, referred to as packet inter-generation delay [17], due to the lack of a shared and accurate time-source along with device imperfections. This can lead to non-deterministic packet arrival times, even in the presence of emerging network technologies that provide deterministic network latency, e.g. Time-Sensitive Networking. Packet inter-generation delay, together with the presence of varying network latency in state-of-the-art wireless network technologies, can result in non-deterministic packet arrival times at the gateway. In many cases, it is impossible to distinguish between a non-arrival due to a fault or due to delay, which can be crucial for detecting failures in distributed systems [6].

The detection of complex events may depend on the occurrence of multiple events, such as the arrival of sensor data, to compute a result. The problem lies in predicting time-boundaries for individual event arrivals, for which state-of-the-art solutions rely on the application developer. This is difficult, if not impossible, due to the heterogeneity and dynamism in the network, platform and applications, along with the application developer's limited knowledge of the underlying infrastructure [1, 2]. For many CPS applications, complex events need to be computed in a timely fashion to produce useful output, e.g. detecting production line down-times in manufacturing and sending an alert to a human operator. However, when the window is closed too soon, not all dependent events may have arrived, leading to incorrect results under incomplete information. Having clear control over the trade-off between timeliness and completeness is of prime importance for CPS applications.

In this paper, we propose Khronos, a middleware solution that aims to simplify the development and maintenance of CPS applications. Khronos allows the developers to precisely trade off timeliness versus completeness of the data produced by the underlying CPS infrastructure. The middleware shields the application developer from the burden of manually specifying timeouts for each data stream, by supporting the specification of completeness constraints: the minimum fraction of packets expected to have arrived from a device data stream. An extensive evaluation on a physical testbed shows that Khronos not only ensures constraint satisfaction in the presence of dynamism and heterogeneity, but also improves timeliness by automatically setting timeouts, based on the observed status of the underlying network. The complete code-base of Khronos and the data sets used in the evaluation are open-source[1], to ensure the reproducibility of our work and promote collaboration.

The remainder of the paper is structured as follows: Section 2 provides additional background and explores the problem through the lens of a real-world industrial use-case. Section 3 provides an overview of the related work and the identified middleware requirements. Section 4 describes the architecture of Khronos and the prediction technique. Section 5 presents the implementation details of Khronos and the CPS network. Section 6 discusses the evaluation setup and results. Finally, Section 7 concludes the paper and Section 8 discusses future work.

2 Background

Technological advancements in the area of Industry 4.0 have created new business opportunities based on integrating CPS with manufacturing to form Cyber-Physical Production Systems (CPPS) [12, 28, 31], which increase efficiency and reduce manufacturing costs [14]. Manufacturing plants are enhanced with actuators and sensors that measure various physical properties, such as product displacement, machine temperature, liquid flow rate, etc [21]. Sensor data are transmitted over the network to a central point (human operator or controller) that takes actions to improve the operation of the manufacturing plant.

Typical network technologies used in industrial CPS applications are wireless mesh and star networks. In mesh networks, messages may need to traverse the network across many hops, traveling through several devices before reaching their final destinations. Based on the underlying medium access control protocols, e.g. Carrier-Sense Multiple Access (CSMA) and Time Synchronized Channel Hopping (TSCH), per-hop latency varies from tens of milliseconds to several seconds. Variable latency is also prevalent in wireless star networks, such as LoRa and BLE, due to radio intereference [22, 23]. As a result, predicting packet arrival times in these networks and specifying corresponding timeouts in the application remains an open challenge.

State-of-the-art solutions rely on the application developer's infrastructure knowledge to manually specify timeouts for each

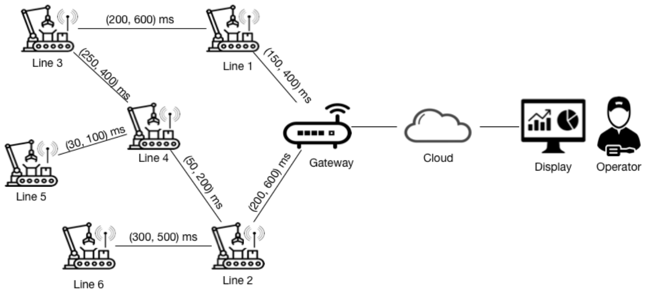

Figure 1: Industrial plant overview consisting of six production lines.

device data stream at compile-time. Static timeouts are hard to specify in advance, since the application developer has limited knowledge of the infrastructure [1, 2], and are highly inflexible in the presence of dynamism, as their performance depends entirely on the current state of the network. Recent work [17] attempts to address this issue, focusing on probabilistic approaches to manage late event arrival times, yet still relies on the user to specify timeouts and further configuration parameters that directly impact their performance. In practice, static timeouts are determined using rules-of-thumbs, such as a multiple of the sampling period or adding the average network delay [24].

Middleware can be used to interface between the various modules of such CPS architectures [13], allowing abstraction and flexibility on the application side, and custom management for the edge of the architecture. However, the diversity and heterogeneity of CPS architectures makes it hard for a general purpose middleware to keep track of the afore mentioned trade-off. A dedicated middleware is required that enables developers to easily manage the timeliness and completeness of events flowing from the CPS to their application, without requiring additional configuration from the user by relying on his/her knowledge of the underlying infrastructure [7, 13].

Use case

This section describes a concrete industrial use-case from the customization and packaging division of a Fortune 500 fast-moving consumer goods company, to further illustrate the problem. An overview of the packaging plant is shown in Figure 1, including the device communication links and corresponding link latency range (e.g. latency between 200ms and 600ms between Line 3 and Line 1). The packaging plant has six production lines, labeled Line 1 to Line 6. Each line consists of various machines that box, seal and wrap items as they pass through and can be reconfigured depending on the product. Sensors at every line count the number of items that are processed and store metadata, e.g. item weight and machine temperature. Sensors transmit a packet to the gateway at fixed sampling periods, containing the recorded metadata as payload over the past time period. The metadata is forwarded to a back-end for analysis, e.g. the cloud, and the results are shown on a display. Typically, the displayed results are the output of complex events that aggregate measurements from multiple sensors, e.g. correlation between measured machine temperature and product weight. The problem lies in determining the refresh period of the display, which

[1]Available at: https://github.com/mazerius/khronos

is non-trivial since packets can travel across different paths in the network, resulting in non-deterministic arrival times due to varying link latency. Despite the small scale of the network, the recorded packet arrival times at the gateway show substantial variance, up to the order of seconds. Unlike the typical Internet, the challenge in WSN technologies is largely due to the resource-constrained nature of the devices and the unreliability of the physical environment.

State-of-the-art solutions require the operators to specify a static refresh period for the display. Operators in industry typically use an arbitrarily large refresh period to ensure completeness of the results at the expense of timeliness. As a result, the display lags behind in time, making it difficult for the operators to interpret the results and act in a timely fashion.

Since the displayed information is a function of event data received by one or more sensors, the dashboard refresh period depends on the packet arrival times and timeouts. Human operators are application experts, and thus know how important it is to wait for each sensor data prior to refreshing the displayed result(s), which is expressed by the completeness constraint. As a result, they benefit from a solution that automatically determines packet arrival timeouts in a way that improves timeliness, while satisfying their application completeness constraints through continuous monitoring of the underlying infrastructure. Operators then require less time to develop and maintain the CPS application and at the same time make timelier decisions, improving the overall operation efficiency of the packaging plant.

For example, the dashboard displays a metric that indicates the operational efficiency of each machine in the production lines, which is a function of input data from two devices: an object detection and a weight sensor respectively. The measured item weight does not vary substantially across items of the same type. As a result, the operational efficiency of production lines that only process a single type of items can be updated more frequently on the display, since it can be computed without always waiting for the item weight input data to arrive. Waiting long enough so that one out of four updates includes the item weight would suffice in that case, corresponding to a 25% completeness constraint.

3 Related work

Middleware and related frameworks are key components of complex systems, especially when dealing with the constraints of CPS.

Due to the broader range of environments and related constraints on network resources found in CPS, the limited support offered by modern systems is not enough and a one-size-fits-all solution is not an option. This need for more specific solutions is raised by Mohamed et al. [13] who identifies that general-purpose distributed middleware are not flexible enough to tackle unique challenges in CPS. The challenges of middleware for CPS, including the support for real-time operations (e.g. decision making), autonomous operations, data integrity and correctness, have been addressed in the past as a subset of these in a generic form for a specific feature, or focusing on one for a more broader group of CPS applications. The authors [13] rely on past work to emphasize the specificity and diversity of a CPS ecosystem, and propose a more context-aware approach to consolidate the generic approach and limit the spread of the specific solutions.

Among these past works, Zhang et al. [30] explored the issues of real-time middleware used as platforms for distributed systems with time constraints when facing workloads with both aperiodic and periodic tasks. In order to tackle the lack of flexibility from existing systems, their contribution of configurable middleware components providing effective on-line admission control and load balancing for distributed computing platforms is an important step for CPS. However, the authors do not address timeliness challenges that occur due to the underlying network and its resource-constrained devices and middleware reconfiguration options to cope with its uncertainties and maintain real-time support.

Significant research efforts focus on the management of late event arrivals in the context of Complex Event Processing systems, such as punctuation [24, 27], speculation [15, 18] and buffer-based data-structures [9–11, 29]. However, these approaches often rely on the user (e.g. application developer) to specify key configuration parameters, limiting their effectiveness in applications where the user has limited knowledge of the underlying system, as in CPS.

The authors in [17] address these shortcomings and the need to leverage completeness over timeliness by proposing ProbSlack, a probabilistic approach towards managing late event arrivals in the presence of varying packet inter-generation and network delay. However, ProbSlack relies on a user-specified period T to refresh its models for the two delays, which has a significant impact on its performance. Furthermore, ProbSlack is designed under the assumption that events arrive in the order in which they are generated, which is not necessarily the case in large CPS networks, where packets from devices close to the gateway can arrive faster than packets generated further away.

Requirements

In the context of the industrial use case and the related work, we identify five requirements for CPS middleware:

- **A)** The middleware should enable CPS application developers to specify completeness constraints of their applications on a per-device basis, through a set of provided services.
- **B)** The middleware should not rely on user knowledge of the underlying infrastructure and require no further configuration after deployment.
- **C)** The middleware should adapt to changes in the CPS infrastructure to satisfy the application constraints in the face of network and application dynamism.
- **D)** The middleware should satisfy the application constraints for a wide variety of different infrastructures and application requirements.
- **E)** The middleware should provide CPS applications with context regarding the completeness and timeliness.

4 Architecture

The proposed middleware acts as a generic bridge between the underlying CPS and the applications that run on top of them. The identified requirements that Khronos addresses are highlighted in Section 3. The provided API, that enables CPS applications to specify completeness constraints for device data streams, is explained in Section 4.1. Next, Khronos' architecture and key responsibilities of each component are discussed in Section 4.2. Finally, the prediction technique used by the middleware to automatically determine event arrival times for sensor data is described in Section 4.3.

4.1 Application Programming Interface

Khronos offers developers a simple API, which consists of two operations:

- `registerCompleteness(device, constraint, on_next, on_timeout, on_violation)`
- `registerTimeout(device, timeout, on_next, on_timeout)`

`registerCompleteness(..)` addresses requirement **A)** and takes five arguments: the device, the constraint and three callback methods. `device:<String>` identifies the CPS device data stream(s) and can be a unique identifier (e.g. serial number) or a wildcard (e.g. sensor type) that refers to a group of devices. `constraint` is the value for the completeness constraint, expressed as a fraction. Given the completeness constraint, the middleware updates the timeout for the next packet, whenever a packet has arrived from the corresponding device. `on_next(value:<Sensor Data>, timeout:<Double>, completeness:<Double>)` is the callback method that is invoked by the middleware whenever data from the specified device arrives on time. It takes as arguments the value of the arrived sensor data and the corresponding timeout and completeness. `on_timeout(timeout, completeness)` is the callback method that is invoked by the middleware whenever the timeout is reached and no sensor data have arrived. It takes as arguments the value of the timeout and the current completeness. `on_violation(value, timeout, completeness)` is the callback method that is invoked by the middleware whenever the completeness is below the constraint when the timeout is reached or a packet has arrived. It takes as arguments the value of the sensor data (if any) and the current timeout and the completeness. For example, a simple application can define `on_next(...)` to update the average temperature whenever new temperature data arrives, `on_timeout(...)` to count the number of occurred timeouts and `on_violation(...)` to spawn a pop-up alert window upon constraint violation.

`registerTimeout(..)` enables developers to register a static timeout for a sensor device data stream and takes four arguments: the device, the static timeout and two callback methods, which contain application logic and are thus specified by the CPS application developer. `device: <String>` is identifies the CPS device data stream(s) and can be a unique identifier or a wildcard that refers to a group of devices. `timeout` is the value for the static timeout for packet arrivals from the given device, expressed in time units (e.g. seconds). Given the timeout, the middleware recomputes the completeness for the given device whenever it receives a new packet from it. `on_next(value, timeout, completeness)` is the callback method that is invoked by the middleware whenever data from the specified device arrives on time. It takes as arguments the value of the arrived sensor data, the given timeout and the current completeness, addressing requirement **E)**. `on_timeout(timeout, completeness)` is the callback method that is invoked by the middleware whenever the timeout for packet arrival from this device is reached. It takes as arguments the current timeout and completeness.

4.2 Components

Figure 2 shows a complete overview of Khronos' architecture. Khronos acts as a generic bridge between external CPS applications and the gateways of the underlying CPS infrastructure. The middleware consists of three layers: CPS Communication, Time Management and Application Management. The rest of the section describes the key responsibilities of each layer and the role of its sub-components in greater detail.

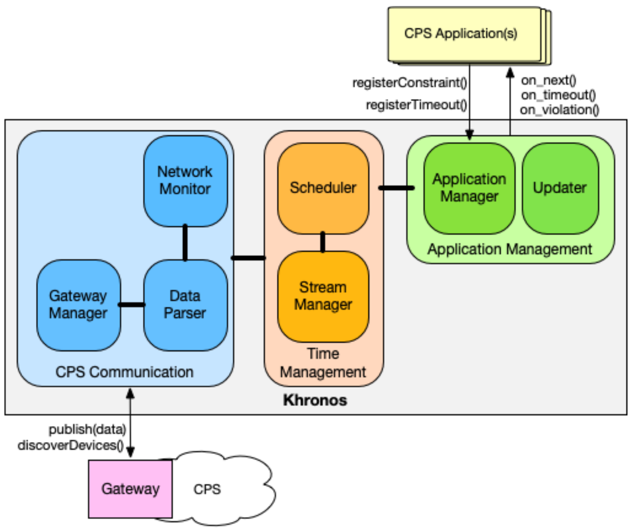

Figure 2: Khronos component diagram.

CPS Communication. This layer is responsible for managing communication between the underlying CPS network. The `Gateway Manager` is responsible maintaining an overview of the underlying devices that connect to the `Gateway`. It listens for published sensor data from each `Gateway` and forwards it to the `Data Parser` for parsing. The parsed data is then passed to the `Network Monitor` and the `Time Management Layer`. The `Network Monitor` maintains an overview of network statistics, including discovered devices and communication latency between Khronos and the gateway(s). For industrial-scale networks with a large number of gateways, the modularity of this layer can be exploited to improve scalability by distributing its components across the gateways and thus decoupling the CPS communication logic from the rest of the middleware.

Time Management. This layer is responsible for processing network statistics and packet arrival times to coordinate the callbacks of each CPS application. The `Stream Manager` maintains completeness statistics and determines the timeouts per device data stream for the completeness constraint(s) specified by the CPS application. The `Scheduler` coordinates application calls using the above timeout(s) by invoking the application callback methods, through the `Application Management` layer, whenever a packet arrives or a timeout is exceeded.

Application Management. This layer is responsible for communication with external CPS applications. The `Application Manager` provides applications with the API described in section 4.1 and is responsible for registering the completeness constraint(s) and/or static timeouts to the `Scheduler`. The `Updater` is responsible for executing `on_next`, `on_timeout` and `on_violation` application callbacks, as instructed by the `Scheduler`.

4.3 Prediction technique

Khronos automatically (re)computes the timeouts of device packet arrivals, based on the application completeness constraint(s) for that device. These timeouts are determined using an approach similar to the Retransmission TimeOut (RTO) timer in the Transmission Control Protocol (TCP), a well-established transport layer protocol for communications over the Internet [25].

TCP's RTO is a durable solution that works on top of a wide, heterogeneous and dynamic infrastructure and also tackles the problem of determining timeouts for non-deterministic packet arrivals. Both RTO and Khronos are faced with a similar challenge: determining how long to wait for packet arrival before taking action, in the presence of varying network latency. In both cases, the trade-off between timeliness and completeness is determined by these timeouts. RTO's approach is simple and lightweight, since it uses exponentially weighted moving averages instead of storing past observations to compute the timeout in every step, which fits the resource-constrained model of CPS. In RTO however, the timeout occurs at the sender, while in CPS they happen at the receiver who also needs to take into account the packet inter-generation delay of the sender. Finally, CPS applications can have flexible completeness constraints, while RTO's design is limited to covering 99% of all packet arrivals.

In TCP, the RTO needs to determine how long to wait for the acknowledgment to arrive after a segment has been sent, before re-transmitting the segment. Short timeouts result in unnecessary re-transmissions and possibly network congestion, while long timeouts negatively impact performance. RTO keeps track of two exponentially weighted moving averages: the smoothed round-trip time (SRTT) and round-trip time variance (RTTVAR), with smoothing factors $\alpha = 7/8$ and $\beta = 3/4$ respectively, as specified in RFC 6298 [19]. SRTT is the best current estimate of the round-trip time to the destination, and RTTVAR is the variance in round-trip times. The timeout is computed as $RTO = SRTT + K * RTTVAR$, where $K = 4$ since less than 1% of all packets arrive more than four standard deviations too late. The role of K in this formula is to over-provision by adding that many times the variance to the mean in order to cover over 99% of packet arrival times [25].

Khronos approach

In the context of our use-case, Khronos needs to determine how long it should wait for each packet to arrive such that the application completeness constraints are satisfied for each device data stream without unnecessarily long timeouts. For each completeness constraint, referring to a particular device data stream, Khronos computes the smoothed arrival time $S(t_i)$ and the arrival time variance $\mathbb{V}(t_i)$, whenever a new packet arrives at a timestamp t_i. $S(t_i)$ is the best current estimate for the next packet arrival time and $\mathbb{V}(t_i)$ the variance in arrival times. These are computed by the formulas:

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

$$\mathbb{V}(t_i) = \beta\mathbb{V}(t_{i-1}) + (1 - \beta)|S(t_{i-1}) - R(t_i)| \tag{2}$$

where $R(t_i)$ is the actual arrival time of the packet at timestamp t_i and α, β the smoothing factors, set to the same values as in RTO, which are empirically derived. A timeout based on $S(t_i)$ alone is too inflexible for large variance in arrival times, which is accounted for by $\mathbb{V}(t_i)$. The timeout $TO(t_i)$ for the next packet at timestamp t_{i+1} is computed as:

$$TO(t_i) = S(t_i) + K * \mathbb{V}(t_i) \tag{3}$$

As in RTO, K determines how sensitive the timeout is to packet arrival time variance. In this use-case, the percentage of packet arrival times that should be covered is specified by the application completeness constraint, which determines the value of K, as explained further in this section.

Finally, the formula is slightly modified to account for the communication delay D_T between the CPS gateway and the middleware:

$$TO(t_i) = S(t_i) + K * \mathbb{V}(t_i) + D_T \tag{4}$$

Khronos periodically pings each gateway to refresh D_T.

The computation cost of our approach is linear ($O(n)$), where n the number of registered completeness constraints. Concretely, whenever a packet arrives from a device, Khronos performs, per-constraint for that device, five multiplications and five additions to compute the next timeout.

Determining K

K is determined empirically by monitoring the underlying CPS network under various conditions. The network is monitored over a period of three weeks, during which packet arrival times are observed under different conditions . In the first week, no disturbances are introduced. In the second week, the sampling period is gradually increased and decreased. In the third week, the network size is reduced and the network latency is increased by re-configuring the network manager. Next, we determine a one-to-one mapping between a given completeness constraint ρ and the smallest value of K that satisfies the constraint across all device data streams in the network. Finally, we over-provision by multiplying each value of K with a factor 2 to improve the robustness of the approach. This simple training phase ensures that the application developers can use the middleware without further configuration, as stated by requirement **B)**. The resulting K values are discussed in Section 5.2.

5 Implementation

This section discusses the key technologies that implement the underlying CPS network and the middleware.

5.1 Network

In industrial CPS applications, like the use case in Section 2, two widely used network technologies are Time Slotted Channel Hopping (TSCH) and Carrier-Sense Multiple Access (CSMA) mesh networks. In a TSCH mesh, all motes are precisely synchronized to tens of microseconds. Time is organized in slots which are allocated to motes in the network, allowing them to know in advance when to turn the radio on or off. Frequency bands are separated in channels, and communications are done using those different channels at different times, resulting in reliable, low-power communication.

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

In CSMA networks, motes sense the shared medium before transmitting to verify the absence of other traffic. In wireless networks, CSMA is often enhanced with Collision Avoidance (CSMA/CA) in order to improve performance, where motes wait for a random period of time after sensing the medium is not free, before retrying.

In the context of our use-case a wireless mesh network is deployed to connect the underlying CPS devices. Additionally, mesh networks introduce increased complexity when dealing with network latency due to multi-hop communication. As a result, we focus on a wireless mesh network and do not use LoRa or BLE for the implementation. The key technology used for the underlying wireless embedded network is SmartMesh IP (SMIP)[2], which is broadly used in industrial CPS applications, such as the manufacturing plant described previously in the use case.

By default, a SMIP network is a TSCH mesh but it can be easily reconfigured to a CSMA mesh through the bbmode setting. In the evaluation, we use this parameter to test our implementation on top of both a TSCH and a CSMA/CA wireless mesh. A SMIP network is a wireless, multi-hop mesh network that self-forms and self-maintains to guarantee high network reliability and ultra low-power. Due to this self-adaptation, several network parameters, including allocated bandwidth, latency and hop-depth, can change over time without any system parameter reconfiguration, leading to non-deterministic packet arrival times.

For this paper, a real-life testbed is built that consists of 33 physical devices: 22 SmartMesh IP motes[3] (DC9003A-B) and 10 VersaSense wireless devices[4] (Model P02) connected through a VersaSense Edge Gateway[5] (Model M01). The SmartMesh IP motes are not equipped with sensors: their role is to act as routers that forward packets they receive across the network, enabling a widespread deployment with a large number of hops.

The VersaSense wireless devices are built on top of SmartMesh IP and provide plug-and-play support: up to four sensors or actuators, known as peripherals, can be connected on each VersaSense device. Each VersaSense device is also equipped with a built-in peripheral that measures the battery-level.

The VersaSense Edge Gateway, which is the network manager, acts as a bridge between the wireless sensor network and Khronos.

Table 1 shows the types of peripherals that are deployed in the testbed along with their quantities and default sampling periods. Each VersaSense device is equipped with at most one peripheral of the same type. These peripherals are fully self-identifying, requiring no further manual intervention. In the rest of the paper, we use the term 'device' to refer to a peripheral connceted to a VersaSense device. It is uniquely defined by the peripheral identifier and the IPv6 address of the VersaSense device.

5.2 Middleware

Khronos runs on a Raspberry Pi 3 and is developed in Python v3.6 as a Representational State Transfer[6] (REST) server, using the `flask` framework and implements the API that was described in Section 4.1. REST is a stateless communication protocol that separates the concerns of the client and server, enabling transparent communication between software systems. In this proof-of-concept implementation, Khronos communicates with applications using Pyro 4.6, a python library that enables remote method invocation (RMI) across the network, on objects created by the client application(s) and stored locally in the client machines. These objects implement the callback methods discussed in Section 4.1: `on_next(...)`, `on_timeout(...)` and `on_violation(...)`. Clients application(s) register these objects to the Pyro name server, which provides them with a URI per object. Application(s) provide this URI as an argument to Khronos when calling the provided API methods, instead of directly passing the callback functions, leading to easier client-server integration. Khronos invokes each of the callbacks accordingly, based on whether or not constraint violation occurred, which then executes the corresponding method locally on the client machine.

The Raspberry Pi is in the same local area network (LAN) as the Versasense Edge Gateway. Khronos obtains the relevant network status information from the Versasense Edge Gateway through a CoAP [7] API, which is a client-server model similar to REST but designed for resource-constrained devices. Additionally, the VersaSense Edge Gateway listens for connections to a websocket[8], which enables full-duplex communication over a single TCP connection. Khronos connects to the websocket to receive the raw sensor data stream, which is processed by the rest of the middleware.

Resulting K

Khronos uses the technique discussed in Section 4.3 to automatically compute the timeouts for individual packet arrivals. K is used in formula (4) to determine the sensitivity to packet arrival time variance. Based on the described methodology, we determine K for a wireless TSCH mesh network. The same values can be used for Khronos on top of a CSMA/CA wireless mesh network, shown by the experiments performed in Section 6. For other network technologies, such as LoRa and BLE, recomputation of K might be necessary. The resulting K values are shown in Table 2 for various completeness constraints ρ. Intuitively, since K determines the sensitivity of the timeout to change, the higher the completeness constraint, the larger the resulting K. For $\rho = 1.0$, K is in theory infinitely large so that packets always arrive on time. In practice, the results show that for $\rho = 1.0$, the constraint violation saturates at 0.3% for $K >= 300$.

[2]https://www.analog.com/en/products/rf-microwave/wireless-sensor-networks/smartmesh-ip.html
[3]https://www.analog.com/en/design-center/evaluation-hardware-and-software/evaluation-boards-kits/dc9003a-b.html
[4]https://www.versasense.com/pdf/VersaSense-Pxx.pdf
[5]https://www.versasense.com/pdf/VersaSense-M01.pdf
[6]https://en.wikipedia.org/wiki/Representational_state_transfer
[7]http://coap.technology/
[8]https://en.wikipedia.org/wiki/WebSocket

Table 2: K values for different completeness constraints ρ.

ρ	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

6 Evaluation

This section discusses the evaluation of Khronos, focusing on two key aspects, based on the requirements in Section 3: (1) the performance of the predicted time windows in the presence of application and network heterogeneity and (2) its ability to adapt to network and application dynamism. For that purpose, we have conducted an extensive set of experiments on a physical testbed that evaluate the performance of Khronos. The evaluation metrics and the approaches Khronos is compared against are discussed in Section 6.1. Next, the empirical evaluation results are presented in Section 6.2 for each set of experiments.

6.1 Setup

All experiments conducted in this evaluation are performed on top of data collected from the real-life testbed consisting of 22 forwarding devices and 20 sensors, as discussed in Section 5.1. The quantity and types of sensors are shown in Table 1. The dataset repository will be made available in the final version. The experiments aim to extensively evaluate Khronos across two dimensions: heterogeneity and dynamism, both typical for CPS networks. Its performance is evaluated against the approaches described in Section 6.1.1, using the metrics discussed in Section 6.1.2.

6.1.1 Approaches

Khronos is compared against three state-of-the-art approaches [17, 24] which use a fixed timeout per device data stream:

DSP (Double Sampling Period). DSP sets the timeout for each packet arrival from a device equal to twice its sampling period. This leads to significantly large timeouts, ensuring high completeness at the expense of timeliness.

SPND (Sampling Period Network Delay). SPND sets the timeout timeout for each packet arrival equal to the device sampling period plus the average network delay. This typically leads to smaller timeouts compared to DSP, at the expense of completeness.

STO (Static Timeout Oracle). STO is a theoretical approach that knows in advance all the packet arrival times from each device. STO computes a fixed timeout based on the completeness constraint for each device data stream. The timeout is equal to the smallest value that satisfies the given constraint for that device data stream across the experiment.

Khronos. For each constraint, Khronos automatically computes timeout predictions for the next packet arrival from the corresponding device whenever a packet arrives, as discussed in Section 4.3.

DSP is an example that opts for completeness, where fixed timeouts are set arbitrarily large enough and SPND opts for timeliness, where timeouts are equal to the device sampling period plus a fixed offset, e.g. the average network latency. In practice, only DSP, SPND and Khronos can be used, since STO requires perfect knowledge of the future to compute the timeouts. However, STO is a reference benchmark as it demonstrates the best possible performance when using fixed timeouts under perfect information.

Table 3: Default SMIP network manager configuration parameters [26].

Parameter	txpower	basebw	numparents	bbmode	bbsize	bwmult
Value	8	50000	2	0	1	1000

6.1.2 Metrics

We measure the performance of each approach using two evaluation metrics:

Prediction Error (PE). This is the average absolute distance, measured in seconds, between the predicted timeout and actual packet arrival time across all packets. PE, for a device data stream d and a completeness constraint ρ, is computed using the formula:

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

where n the total number of packet arrivals for device data stream d, p_k the arrival time of the kth packet, to_k is the timeout for the kth packet and $distance(p_k, to_k)$ the function:

$$distance(p_k, to_k) = abs(p_k - to_k)$$

Intuitively, the smaller the PE, the closer the timeouts are to the actual event arrival times, leading to more timely reactions.

Constraint Violation (CV). This is the percentage of packet arrivals for which the constraint is violated. Completeness is the fraction of packets that arrived before the timeout. It is measured as a moving average over the past 100 packets that arrived from the same device: smaller window sizes result in coarse grained values, while larger window are less sensitive to change. A completeness constraint is satisfied when over 99.999% of the time, the measured completeness is equal to or above the constraint. In theory, to ensure that a completeness constraint ρ of 1.0 or 100% is satisfied over 99.999% of the time, the corresponding timeouts would be quasi-infinitely large. This is impractical, since timeouts should still occur within a reasonable amount of time. Thus, for 100% completeness constraint we compare each approach on a best-effort basis.

6.1.3 Parameters

Each approach is evaluated for completeness constraints $\rho \in [0.1, 1.0]$. Unless specified otherwise, by default the results are illustrated for a completeness constraint $\rho = 0.8$. The default values used for the most important network manager configuration parameters are shown in Table 3.

6.2 Results

This section provides an overview of the performed experiments and results, comparing Khronos against the approaches discussed previously in Section 6.1.1. The experiments evaluate Khronos across two dimensions: dynamism and heterogeneity.

6.2.1 Dynamism

This section evaluates the capability of Khronos to satisfy application completeness constraints in the presence of network and application dynamism, as specified by requirement **C)**. From the use case and literature study, we identify three sources of dynamism.

First, applications can change device sampling periods over time in order to achieve their goal. Second, devices can fail (join) at any time, which results in a smaller (larger) network size, expressed as the number of operational motes. Third, network parameters can be re-configured on the spot to impact the network latency. In the case of SMIP, the re-configuration requires a reset of the network manager to take effect. In the experiments to follow, devices are deployed across three floors of our departmental building, up to one floor away from the gateway. The acquired results show a similar trend for each of the 20 unique devices. Due to space limitations, we illustrate these for an arbitrary selected device, with identifier `9803/9805|fd34...dfe8`.

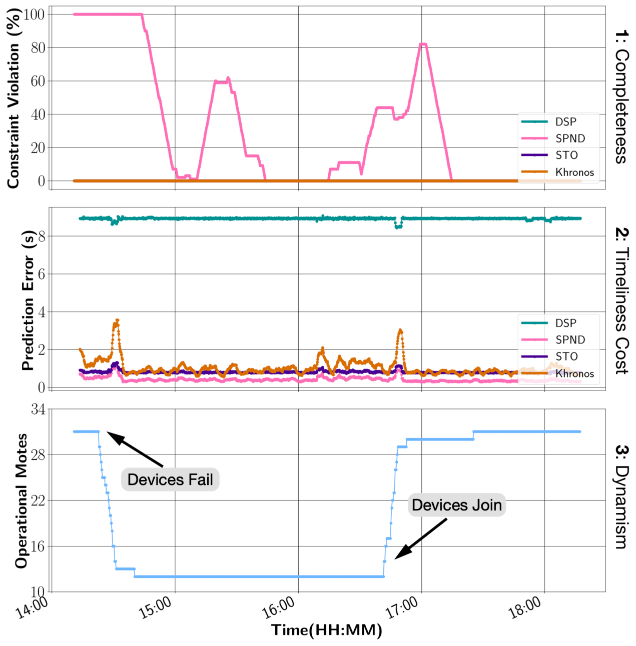

Figure 3: Timeplot for a single device, illustrating the impact of network size dynamism on the performance of each approach, given completeness constraint $\rho = 0.8$. The four approaches are compared in terms of the prediction error (Subplot 2), measured in seconds, and completeness constraint violation percentage (Subplot 1).

Impact of Network Size

In this experiment, we test the hypothesis that Khronos can consistently satisfy application completeness constraints in the presence of network size dynamism, by turning off and on 66.67% of the devices.

Figure 3 shows the impact of changing the network size on the constraint violation and prediction error, over five hours, for a sampling period of 10 seconds and $\rho = 0.8$. The left and right arrows in Subplot 3 indicate the two events: reducing and increasing the network size respectively. Khronos reacts to the changes by increasing the timeouts, which lead to temporarily larger prediction errors (Subplot 2) but ensure the constraint violation remains at 0% (Subplot 1). SPND is the only approach that violates the constraint throughout this experiment, as indicated by the pink line in Subplot 1. Overall, Khronos always satisfies the constraint, just like DSP and STO, bus has a far smaller prediction error than DSP, shown in Subplot 2. Note that DSP's prediction error is proportional to the sampling period, which can be up to many orders of magnitude larger than Khronos'.

Performance with Dynamic Sampling Periods

In this experiment, we test the hypothesis that Khronos can consistently satisfy application completeness constraints in the presence changing sampling periods, by re-configuring the devices. We evaluate the performance of each approach in two scenarios: step-wise increase and step-wise decrease of the sampling period.

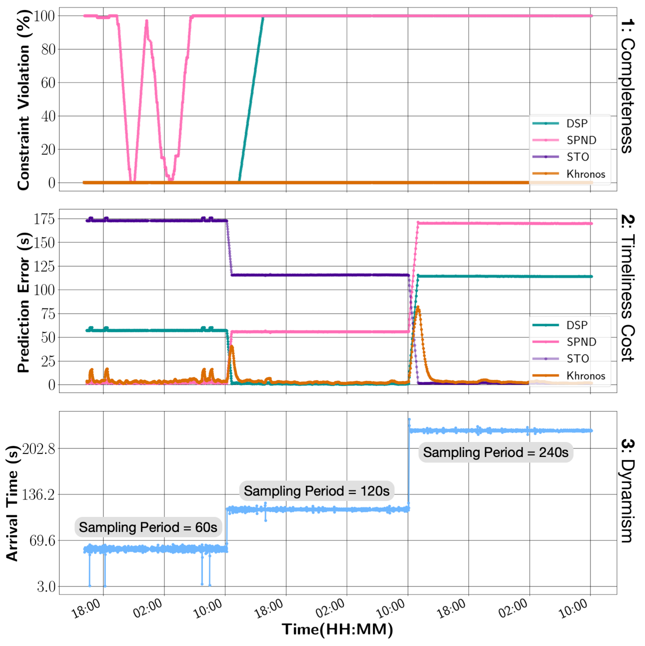

Figure 4: Timeplot for a single device, illustrating the impact of increasing sampling period, given completeness constraint $\rho = 0.8$. The four approaches are compared for the prediction error, measured in seconds, (Subplot 2) and completeness constraint violation percentage (Subplot 1).

Figure 4 shows the impact increasing the sampling period over three days. The sampling period is increased from 60 to 120 seconds and from 120 to 240 seconds, shown by the arrival times in Subplot 3. Khronos reacts to both changes by increasing the timeouts, leading to two peaks in the resulting prediction error (Subplot 2). The benefits of Khronos in terms of completeness are shown in Subplot 1: DSP and SPND both fail the constraint (Subplot 1) after the first change, in contrast to Khronos which always has a constraint violation smaller than 0.001%. Additionally, Khronos is the only approach that achieves a consistently low prediction error compared to the alternative approaches. The large prediction errors of DSP, SPND and STO clearly show the limitations of static timeouts, even in the presence of perfect knowledge of the future.

Figure 5 shows the impact of decreasing the sampling period over the course of three days. The sampling period is decreased from 240 to 120 seconds and from 120 to 60 seconds, shown in Subplot 3. Since

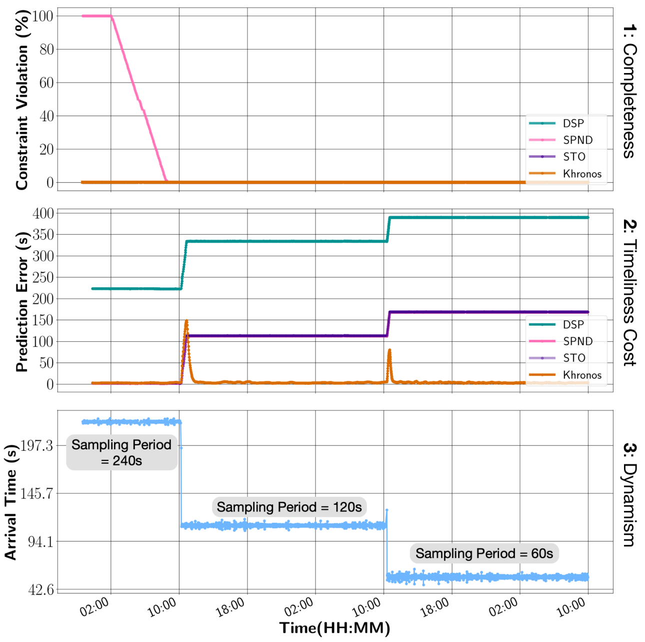

Figure 5: Timeplot for a single device, illustrating the impact of decreasing sampling period, given completeness constraint $\rho = 0.8$. The four approaches are compared in terms of the prediction error, measured in seconds, (Subplot 2) and completeness constraint violation percentage (Subplot 1).

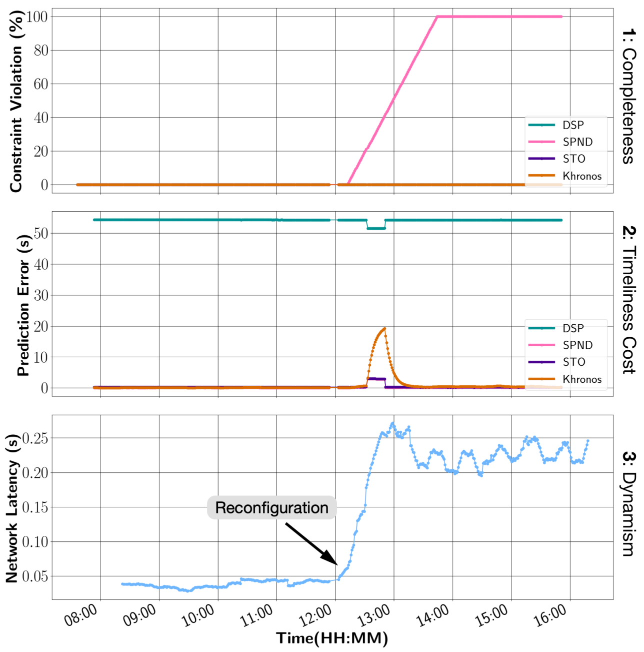

Figure 6: Timeplot for a single device, illustrating the impact of increasing network latency, given completeness constraint $\rho = 0.8$. The four approaches are compared in terms of the prediction error, measured in seconds, (Subplot 2) and completeness constraint violation percentage (Subplot 1).

SPND is defined by the initial sampling period and the sampling periods decrease, its CV decreases to 0% (Subplot 1). However, there is a clear penalty in timeliness for DSP, SPND and STO, shown by the large PE in Subplot 2. Khronos is the only approach that consistently satisfies the constraint (CV < 0.001%) while at the same time resulting in drastically smaller prediction error, compared to the other approaches across the entire experiment.

Performance with Dynamic Network Latency

In this experiment, we test the hypothesis that Khronos can consistently satisfy application completeness constraints in the presence of network reconfiguration, leading to increased network latency and variance. Figure 6 shows the impact of changing networking latency by re-configuring the network manager, over ten hours, for a sampling period of 60 seconds and $\rho = 0.8$. The network manager is restarted, indicated by the arrow in Subplot 3, with new configuration: `bwmult` = 100 and `basebw` = 1000. The new configuration leads to higher and more variable latency (Subplot 3), which Khronos detects and reacts by increasing the timeouts, leading to temporarily larger prediction error (peak in Subplot 2) to ensure the completeness constraint remains satisfied (Subplot 1). SPND violates the constraint after network reconfiguration and DSP results in a prediction error that's proportional to the sampling period.

6.2.2 Heterogeneity

This section evaluates the capability of Khronos to satisfy application completeness constraints in the presence of network and application heterogeneity, as specified by requirement **D)**. Broadly, we identify two classifications for heterogeneity: on the network and on the application level. Networks can vary in their topology based on the use-case at hand. Similarly, networks can differ in their medium access control schemes, based on the application requirements (e.g. low-latency versus reliability). Sharing the medium without synchronization (CSMA) can lead to lower network latency, while time-synchronized channel-hopping (TSCH) minimizes packet collisions by allocating dedicated communication slots to each device. Finally, applications can require devices to sample at distinct rates while imposing different completeness constraints. The rest of this subsection compares the performance of the approaches for separate completeness constraints, network topologies, medium access control protocols and sampling periods.

The results for each approach are shown as error-bar charts, where the bar height is equal to the mean across the 20 peripherals and the min and max values are respectively the lower and upper bound of the error range.

Meeting a Range of Completeness Constraints

In this experiment, we test the hypothesis that Khronos consistently satisfies a range of different completeness constraints ρ. The testbed is deployed across three building floors and each device is up to one floor away from the gateway. Devices are configured with their default sampling periods, shown in Table 1, and data has been collected over seven days. During this period, over 4 million packets arrived at the gateway across all devices.

Figure 7 illustrates the constraint violation for each ρ. A constraint is violated when this percentage is higher than 0.001%, with

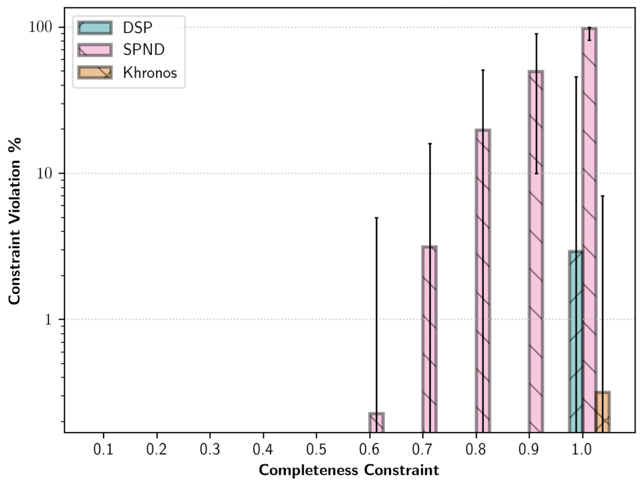

Figure 7: Percentage of completeness being below the constraint, for various constraints ρ.

the exception of $\rho = 1.0$. STO by design never violates any constraint and is thus omitted from the figure. Khronos and DSP never violate the constraint, while SPND fails to satisfy $\rho >= 0.6$. For $\rho = 1.0$, Khronos is below it only 0.32% of the time, over 10 times less than DSP.

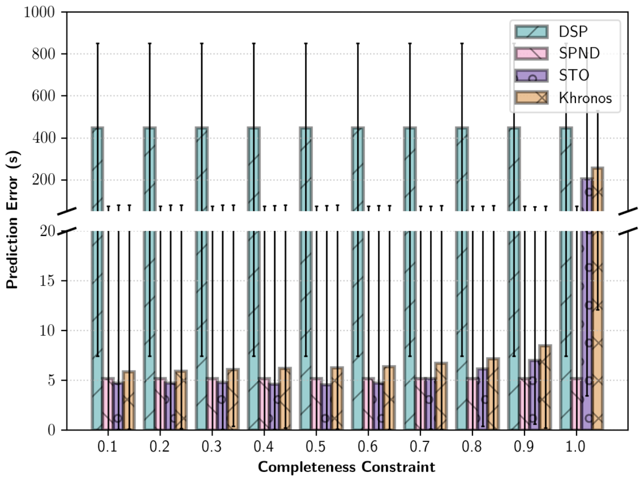

Figure 8: PE per-approach, measured in seconds, for varying completeness constraints ρ.

Figure 8 illustrates the PE each approach, computed for different ρ. Overall, DSP has the highest prediction error, with a mean of 447 seconds and a max value of 850 seconds. Khronos' PE is in the same order as that of SPND and STO.

The results show the Khronos satisfies all completeness constraints at least as well as DSP, with a prediction error comparable to that of SPND, almost two orders of magnitude less than DSP.

Performance in Heterogeneous Network Topologies

In this experiment, we test the hypothesis that Khronos satisfies completeness constraints for different network topologies. We compare the performance of the approaches for two different deployments. In topology A, the entire testbed is deployed within one meter of the gateway. In topology B, devices are deployed across a building, up to two floors away from the gateway. For each topology, data has been collected over 72 hours and devices are configured with their default sampling periods, shown in Table 1. During this period, around 2 million packets arrived at the gateway across all devices.

Table 4: Constraint Violation (%) per-approach for topology A and B, where completeness constraint $\rho = 0.8$.

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

The constraint violation percentage (CV) of each approach is shown in Table 4. The results show that Khronos does not violate the constraint in either topology, unlike SPND (27.8% in topology A and 42.8% in topology B) and DSP (0.045% in topology B).

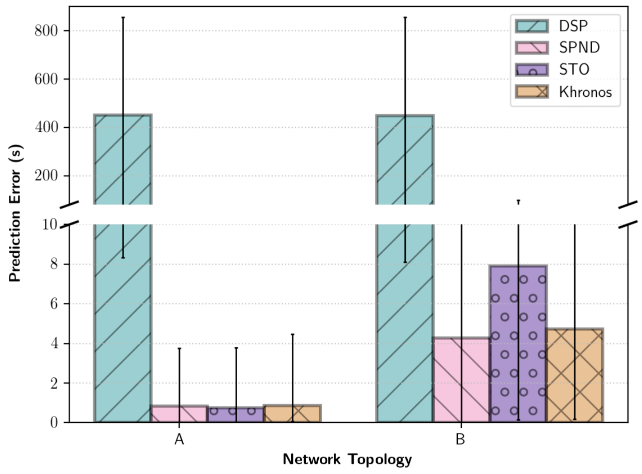

Figure 9: PE per-approach across two network topologies, measured in seconds, for $\rho = 0.8$.

Figure 9 illustrates the PE of the different approaches per-topology, for $\rho = 0.8$. In topology A, Khronos has a PE in the same order as SPND and STO, while in topology B its PE is almost half of STO and around two orders of magnitude less than DSP.

Performance with Heterogeneous Medium Access Control Protocols

In this experiment, we test the hypothesis that Khronos can satisfy completeness constraints for networks with different medium access control protocols. We compare the performance of the approaches for two different medium access control protocols : TSCH and CSMA/CA. Data is collected over 72 hours, during which around 2 million packets are received at the gateway. All devices are deployed within one meter of the gateway (Topology A).

The constraint violation percentage (CV) of each approach for a TSCH and CSMA/CA wireless mesh is shown in Table 5. The

Table 5: Constraint Violation (%) per-approach for a TSCH and CSMA/CA wireless mesh network, where $\rho = 0.8$.

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

results show that Khronos does not violate the constraint in either topology, unlike SPND (27.8% in topology A and 40% in topology B).

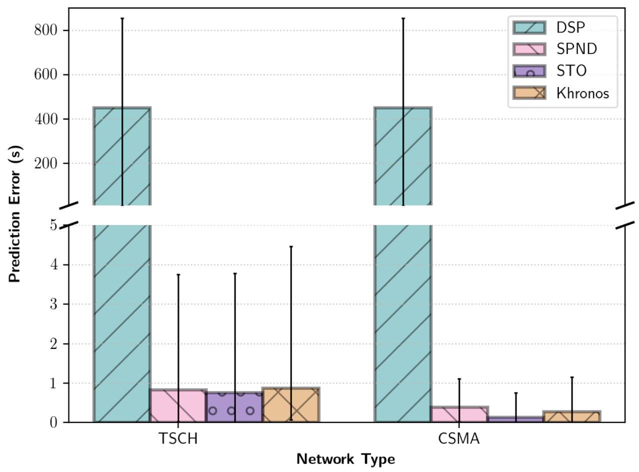

Figure 10: Prediction Error, measured in seconds, per-approach for TSCH and CSMA/CA, completeness constraint $\rho = 0.8$.

Figure 10 illustrates the PE per-approach for TSCH and CSMA/CA. While both DSP and Khronos satisfy the constraint, Khronos scores similarly to SPND and STO with a PE of 0.87 seconds (TSCH) and 0.27 seconds (CSMA/CA), drastically less than DSP's mean of 450 seconds.

Performance with Heterogeneous Sampling Periods

In this experiment, we test the hypothesis that Khronos can satisfy completeness constraints for different device sampling periods. We compare the performance of each approach for four sampling periods: 10, 60, 120 and 900 seconds. The testbed is deployed across three floors of our departmental building, each device up to one floor away from the gateway. For each constraint, data has been collected over a course of seven days and devices are configured with their default sampling periods, shown in Table 1. During this period, over 4 million packets arrived at the gateway across all devices.

The constraint violation percentage (CV) of each approach for different device sampling periods is shown in Table 6. The results show that Khronos and DSP always satisfy the constraint $\rho = 0.8$, while SPND fails it 21.5%, 20.3%, 25.16% and 16.18% of the time for a sampling period of 10, 60, 120 and 900 seconds respectively.

Figure 11 illustrates the PE per-approach for TSCH and CSMA/CA. Khronos' PE is slightly above NDSP and comparable to STO, while

Table 6: Constraint Violation (%) per-approach for different device sampling periods and constraint $\rho = 0.8$.

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%

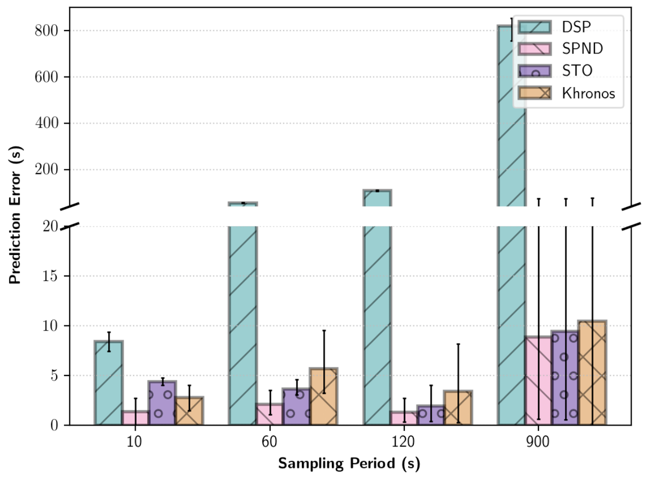

Figure 11: PE measured in seconds, per approach, for various sampling periods and $\rho = 0.8$.

satisfying $\rho = 0.8$ in contrast to NDSP. DSP satisfies the constraint at a far larger cost, with a PE proportional to the sampling period and at least two orders of magnitude higher than Khronos for a sampling period larger than 10 seconds.

7 Conclusion

CPS are increasingly integrated with critical physical processes, including manufacturing, healthcare and smart grids, enabling advanced monitoring and control to improve operational efficiency. In the presence of varying network latency, due to the heterogeneity of the underlying platform, network technologies and the dynamicity of the system, reacting in a timely manner to changes while operating over complete information remains a crucial yet open challenge.

This paper introduced Khronos, a middleware that provides services to CPS application developers that enable them to easily trade-off timeliness versus completeness in their applications. Concretely, Khronos supports the specification of completeness constraint(s) per-device data stream, shielding the developer from manually specifying packet arrival timeouts or further configuration parameters. This is achieved by monitoring the CPS infrastructure and automatically specifying the packet arrival timeouts that satisfy the specified completeness requirements. Khronos relies on a single configuration parameter K, which we determine empirically for a wireless mesh network following a simple methodology. A natural progression of this work is to analyze techniques that adapt the value of K, eliminating the need for pre-deployment configuration (e.g. by applying machine learning techniques).

Khronos is extensively evaluated on top of a physical testbed of 33 devices, connected through a state-of-the-art wireless mesh network that supports two medium access control protocols: TSCH and CSMA/CA. The results show that Khronos never violates the application completeness constraints in the presence of heterogeneity and dynamism in the underlying CPS. Additionally, Khronos greatly improves compare to state-of-the-art approaches, resulting in up to two orders of magnitude smaller timeouts. This enables CPS application developers to easily write reliable distributed applications with flexible completeness requirements.

8 Future Work

Our ongoing research efforts focus on developing expressive programming languages that ease the development of CPS applications. Literature shows that reactive programming fits well with the event-driven nature of CPS applications, due to the automatic propagation of changes throughout the program dependency graph [4, 5]. Distributed reactive programming takes into account the heterogeneity and distribution of CPS by distributing the dependency graph and application across multiple physical nodes. Existing reactive programming frameworks can benefit from integrating with Khronos by alleviating programmers from statically specifying packet arrival timeouts per device data stream.

Acknowledgments

This research is partially funded by the Research Fund KU Leuven and the FWO D3-CPS project.

References

[1] Sven Akkermans, Stefanos Peros, Nicolas Small, Wouter Joosen, and Danny Hughes. 2018. Supporting IoT Application Middleware on Edge and Cloud Infrastructures. In *ZEUS.*

[2] Jean-Paul Arcangeli, Raja Boujbel, and Sébastien Leriche. 2015. Automatic deployment of distributed software systems: Definitions and state of the art. *Journal of Systems and Software* 103 (2015), 198 – 218. https://doi.org/10.1016/j.jss.2015.01.040

[3] Radhakisan Baheti and Helen Gill. 2011. Cyber-physical systems. *The impact of control technology* 12 (2011), 161–166.

[4] Engineer Bainomugisha, Andoni Lombide Carreton, Tom van Cutsem, Stijn Mostinckx, and Wolfgang de Meuter. 2013. A Survey on Reactive Programming. *ACM Comput. Surv.* 45, 4, Article 52 (Aug. 2013), 34 pages. https://doi.org/10.1145/2501654.2501666

[5] Ben Calus, Bob Reynders, Dominique Devriese, Job Noorman, and Frank Piessens. 2017. FRP IoT Modules As a Scala DSL. In *Proceedings of the 4th ACM SIGPLAN International Workshop on Reactive and Event-Based Languages and Systems (REBLS 2017).* ACM, New York, NY, USA, 15–20. https://doi.org/10.1145/3141858.3141861

[6] X. Defago, P. Urban, N. Hayashibara, and T. Katayama. 2005. Definition and specification of accrual failure detectors. In *2005 International Conference on Dependable Systems and Networks (DSN'05).* 206–215. https://doi.org/10.1109/DSN.2005.37

[7] Marisol García-Valls and Roberto Baldoni. 2015. Adaptive Middleware Design for CPS: Considerations on the OS, Resource Managers, and the Network Runtime. In *Proceedings of the 14th International Workshop on Adaptive and Reflective Middleware (ARM 2015).* ACM, New York, NY, USA, Article 3, 6 pages. https://doi.org/10.1145/2834965.2834968

[8] Sabina Jeschke, Christian Brecher, Tobias Meisen, Denis Özdemir, and Tim Eschert. 2017. *Industrial Internet of Things and Cyber Manufacturing Systems.* Springer International Publishing, Cham, 3–19. https://doi.org/10.1007/978-3-319-42559-7_1

[9] Yuanzhen Ji, Anisoara Nica, Zbigniew Jerzak, Gregor Hackenbroich, and Christof Fetzer. 2016. Quality-driven disorder handling for concurrent windowed stream queries with shared operators. In *Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems.* ACM, 25–36.

[10] Yuanzhen Ji, Jun Sun, Anisoara Nica, Zbigniew Jerzak, Gregor Hackenbroich, and Christof Fetzer. 2016. Quality-driven disorder handling for m-way sliding window stream joins. In *Data Engineering (ICDE), 2016 IEEE 32nd International Conference on.* IEEE, 493–504.

[11] Yuanzhen Ji, Hongjin Zhou, Zbigniew Jerzak, Anisoara Nica, Gregor Hackenbroich, and Christof Fetzer. 2015. Quality-driven continuous query execution over out-of-order data streams. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data.* ACM, 889–894.

[12] Paulo Leitão and Stamatis Karnouskos. 2015. *Industrial Agents: Emerging Applications of Software Agents in Industry.* Morgan Kaufmann.

[13] Nader Mohamed, Jameela Al-Jaroodi, Sanja Lazarova-Molnar, and Imad Jawhar. 2017. Middleware Challenges for Cyber-Physical Systems. *Scalable Computing: Practice and Experience* 18, 4 (2017), 331–346.

[14] L. Monostori, B. Kádár, T. Bauernhansl, S. Kondoh, S. Kumara, G. Reinhart, O. Sauer, G. Schuh, W. Sihn, and K. Ueda. 2016. Cyber-physical systems in manufacturing. *CIRP Annals* 65, 2 (2016), 621 – 641. https://doi.org/10.1016/j.cirp.2016.06.005

[15] Christopher Mutschler and Michael Philippsen. 2013. Reliable speculative processing of out-of-order event streams in generic publish/subscribe middlewares. In *Proceedings of the 7th ACM international conference on Distributed event-based systems.* ACM, 147–158.

[16] R. Rajkumar, I. Lee, L. Sha, and J. Stankovic. 2010. Cyber-physical systems: The next computing revolution. In *Design Automation Conference.* 731–736. https://doi.org/10.1145/1837274.1837461

[17] Nicolo Rivetti, Nikos Zacheilas, Avigdor Gal, and Vana Kalogeraki. 2018. Probabilistic Management of Late Arrival of Events. In *Proceedings of the 12th ACM International Conference on Distributed and Event-based Systems (DEBS '18).* ACM, New York, NY, USA, 52–63. https://doi.org/10.1145/3210284.3210293

[18] Esther Ryvkina, Anurag S Maskey, Mitch Cherniack, and Stan Zdonik. 2006. Revision processing in a stream processing engine: A high-level design. In *Data Engineering, 2006. ICDE'06. Proceedings of the 22nd International Conference on.* IEEE, 141–141.

[19] Y. Shafranovich. 2011. *Computing TCP's Retransmission Timer.* RFC 6298. RFC Editor. https://tools.ietf.org/html/rfc6298

[20] Jianhua Shi, Jiafu Wan, Hehua Yan, and Hui Suo. 2011. A survey of cyber-physical systems. In *Wireless Communications and Signal Processing (WCSP), 2011 International Conference on.* IEEE, 1–6.

[21] Sabrie Soloman. 2010. *Sensors Handbook, Second Edition.*

[22] R. B. Sørensen, D. M. Kim, J. J. Nielsen, and P. Popovski. 2017. Analysis of Latency and MAC-Layer Performance for Class A LoRaWAN. *IEEE Wireless Communications Letters* 6, 5 (Oct 2017), 566–569. https://doi.org/10.1109/LWC.2017.2716932

[23] Michael Spörk, Carlo Alberto Boano, and Kay Römer. 2019. Improving the Timeliness of Bluetooth Low Energy in Noisy RF Environments. In *Proceedings of the 2019 International Conference on Embedded Wireless Systems and Networks (EWSN 2019).* ACM, USA, 12.

[24] Utkarsh Srivastava and Jennifer Widom. 2004. Flexible time management in data stream systems. In *Proceedings of the twenty-third ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems.* ACM, 263–274.

[25] Andrew S Tanenbaum, David Wetherall, et al. 2014. *Computer networks.* Harlow, Essex: Pearson,.

[26] Linear Technology (Ed.). 2016. *SmartMesh IP Embedded Manager CLI Guide.* Linear Technology.

[27] Peter A. Tucker, David Maier, Tim Sheard, and Leonidas Fegaras. 2003. Exploiting punctuation semantics in continuous data streams. *IEEE Transactions on Knowledge and Data Engineering* 15, 3 (2003), 555–568.

[28] Lihui Wang, Martin Törngren, and Mauro Onori. 2015. Current status and advancement of cyber-physical systems in manufacturing. *Journal of Manufacturing Systems* 37 (2015), 517–527.

[29] Nikos Zacheilas, Vana Kalogeraki, Yiannis Nikolakopoulos, Vincenzo Gulisano, Marina Papatriantafilou, and Philippas Tsigas. 2017. Maximizing determinism in stream processing under latency constraints. In *Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems.* ACM, 112–123.

[30] Y. Zhang, C. Gill, and C. Lu. 2008. Reconfigurable Real-Time Middleware for Distributed Cyber-Physical Systems with Aperiodic Events. In *2008 The 28th International Conference on Distributed Computing Systems.* 581–588. https://doi.org/10.1109/ICDCS.2008.96

[31] K. Zhou, Taigang Liu, and Lifeng Zhou. 2015. Industry 4.0: Towards future industrial opportunities and challenges. In *2015 12th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD).* 2147–2152. https://doi.org/10.1109/FSKD.2015.7382284