D2.4: AUTOWARE Data Management and Distribution

Document Owner	CNR
Contributors	Theofanis Raptis, Andrea Passarella, Marco Conti, Giovanni Mainetto (CNR)
Reviewers	FhG, UMH
Dissemination level	Public
Dissemination Nature	Report
Version	1.0
Date	29/06/2018

Version History

Nr.	Date	Author (Organization)	Description
0.1	01/05/2018	CNR	Deliverable structure and ToC
0.2	12/05/2018	CNR	Contributions to Section 3
0.3	20/05/2018	CNR	Contributions to Section 4
0.4	31/05/2018	CNR	Contributions to Section 5
0.5	10/06/2018	CNR	First consolidated version
0.6	25/06/2018	FhG	Internal review returned
0.7	26/06/2018	CNR	Revisions
0.8	27/06/2018	UMH	Internal review returned
0.9	28/06/2018	CNR	Revision
1.0	29/06/2018	CNR	Final version

Project Partners

Software Quality Systems	SQS
Asociación de Empresas Tecnológicas Innovalia	INNO
Technologie Initiative SmartFactoryKL e.V.	SmartFactory
Josef Stefan Institute	JSI
TTTech Computertechnik AG	TTT
Consiglio Nazionale Delle Ricerche	CNR
imec	imec
Robovision	Robovision
Universidad Miguel Hernández	UMH
Fraunhofer-Gesellschaft zur Förderung der angewandten Forschung e.V.	FhG
Blue Ocean Robotics	BOR
Fundación Tekniker	Tekniker
SMC Pneumatik GmbH	SMC

Executive Summary

Smart data distribution is a core component in the AUTOWARE communications and data management framework, as it implements the logic of a control plane for cognitive data distribution across all communication layers of AUTOWARE architecture, is based on the available communication technologies and is supporting the needs of the higher level services and applications. Task 2.4 investigates novel smart data distribution solutions which cooperate with cloud-based service provisioning and communication technologies. The solutions presented in D2.4 determine when it is appropriate to move data towards locations where services can be provided. In this context, AUTOWARE exploits storage and computation resources on various elements of the industrial network. AUTOWARE decentralized data management and distribution proposals also contribute towards the design of automation processes that are more capable to dynamically reconfigure. To achieve these objectives, the designed data management and distribution schemes provide distributed methodologies and smart algorithms.

T2.4 started at M7 of AUTOWARE with a thorough investigation of the state of the art on novel smart data distribution solutions that cooperate with cloud-based service provisioning and communication technologies in industrial IoT networks. Typically, in industrial IoT networks, data generated by monitoring IoT devices are collected, elaborated and sent to controllers and actuators. We concluded that, in the vast majority of the proposed solutions, centralized schemes are used. More specifically, the data are transferred to a central network controller, from where they are accessed by any other industrial node requiring them. Since the routing of data from IoT sensors to actuators is an integral part for maintaining critical delay requirements, this data distribution pattern, although robust, may result in significant overheads and severely suboptimal resource consumption. Additionally, the potential inflexibility of this pattern in large scale networks, as well as in dynamic conditions might lead to negative effects in the network, ranging from losing data to missing critical deadlines and consuming higher amounts of energy than needed.

The AUTOWARE smart data distribution component is decentralized, dynamic and hierarchical, so as to support cooperation with cloud-based service provisioning and communication technologies. More specifically, at first, T2.4 provides decentralized data distribution among the various nodes of the AUTOWARE ecosystem. This decentralization is targeting the efficient management of massive data generation and consumption and the exploitation of resources available locally at individual nodes. Then, it copes with dynamically changing network parameters by employing adaptive data placement and replication techniques. Finally, it provides a hierarchical data management, by employing a mix of centralized and decentralized control processes and a hierarchy of data managers. D2.4 addresses three core issues and presents the corresponding key technical contributions: (i) testing of the overall concept, by considering a flat network topology, (ii) hard data access latency deadlines, and exploitation of the presence of fog nodes, and (iii) dynamic network reconfigurations. Those contributions are validated by both large-scale simulation models and real-world results on an experimental testbed.

As a first technical contribution for T2.4, and in order to address the aforementioned shortcomings, we introduce the novel concept of a distribute Data Management Layer (DML), whereby nodes can cooperate so as to store data within the network. In its full potential, the DML is decoupled yet able to interact with the underlying network plane. For example, given a set of data, the sets of nodes generating and requesting them, and a maximum access delay that requesting nodes can tolerate, the DML is able to efficiently identify a limited set of proxies in the network where data are stored. Given the mentioned constraints, we conduct an initial investigation and simulations so as to test the validity of the DML concept. From the technological point of view, even though multiple alternatives can be taken into account, we have primarily focus on communication technologies which can potentially support IEEE 802.15.4e, due to its flexibility and due to the fact that it is becoming a de-facto standard in the sector. We address the (computationally difficult) problem of finding which network nodes to select as proxies and we propose a simple method to solve it. We demonstrate, via simulations of large

scale industrial IoT networks, that the proposed method (i) guarantees that access delay stays below the given threshold, and (ii) significantly outperforms centralized and even distributed approaches, both in terms of access latency and in terms of maximum latency guarantees. Through those findings, we verify that the DML concept can be more than useful in industrial IoT networks, and we obtain a roadmap for the continuation of T2.4. As a result, during M12-M18, we extended this technical contribution in two ways.

At first, we address the problem of the maximization of the network lifetime, given the proxy locations in the network, the initial limited energy supplies of the nodes, the data request patterns (and their corresponding parameters), and the maximum latency that consumer nodes can tolerate since the time they request data. We prove that the problem is computationally hard and we design an offline centralized heuristic algorithm for identifying which paths in the network the data should follow and on which proxies they should be cached, in order to meet the latency constraint and to efficiently prolong the network lifetime. We implement the method and evaluate its performance using an FIT IoT-LAB testbed, comprised of IEEE 802.15.4-enabled WSN430 network nodes. We demonstrate that the proposed heuristic guarantees data access latency below the given threshold, and performs well in terms of network lifetime with respect to a theoretically optimal solution.

Then, we focus on maintaining the network configuration in a way such that application requirements are met after important network operational parameters change due to some unplanned events (e.g., heavy interference, excessive energy consumption), while guaranteeing an appropriate use of node energy resources. We provide several efficient algorithmic functions which reconfigure the paths of the data distribution process, when a communication link or a network node fails. Those functions regulate how the local path reconfiguration should be implemented and how a node can join a new path or modify an already existing path, ensuring that there will be no loops. We demonstrate through simulations the performance gains of the designed method in terms of energy consumption and data delivery success rate.

Contents

List of Figures

List of Tables

Acronyms

AODV	Ad Hoc On Demand Distance Vector
CRC	Cyclic Redundancy Check
D2CIF	Directed Two Commodity Integral Flow Problem
DistrDataFwd	DIstributed Data Forwarding
DM	Data Manager
DML	Data Management Layer
IoT	Internet of Things
LCA	Lowest Common Ancestor
LCED	Latency Constrained Edge Data Distribution Problem
LTE	Long Term Evolution
MCU	Microcontoller Unit
NP	Non-deterministic Polynomial Time
PDD	Priority Data Distribution
RPL	IPv6 Routing Protocol for Low-Power and Lossy Networks
TTL	Time To Live
WG	Working Group

1 Introduction

1.1 Background

Usually, in large-scale industrial networks (Fig. 1), the acquired data is transferred to a central network controller using wireless or wired links (1). The controller analyses the received information and, if needed, changes the behavior of the physical environment through actuator devices (2). However, routing the data centrally, as well as imposing data transfers back and forth in the network may lead to severely sub-optimal paths (3), which in turn negatively affect the overall network latency. The fact that automation control may span multiple physical locations and include heterogeneous data sources also pushes towards decentralization. Moreover, the adoption of Internet of Things technologies with the associated massive amounts of generated data makes decentralized data management inevitable (4). Cloud technologies are considered a great opportunity to implement different types of data-centric automation services at reduced costs, but deploying control-related services in clouds poses significant challenges such as loss of control, delays and jitters. Consequently, data-centric operations and decentralization are two fundamental cornerstones of modern industrial automation technologies, according to the paradigms of Industry 4.0 and form the basis for decision making and control operations. On the other hand, the use of statistical data analysis (Big Data) techniques for control is well established, and several approaches have been proposed to optimize data management and distributed processing, yet smart data distribution policies that take care of replicating data to locations from where they can be accessed when needed within appropriate deadlines, are still to be investigated.

Edge computing, also referred to as *fog computing*, implements technical features that are typically associated with advanced networking and can satisfy those requirements (5). Fog computing differs from cloud computing with respect to the actual software and hardware realizations as well as in being located *in spatial proximity to the data consumer* (for example the user could be a device in the industrial IoT case). In particular, components used to realize the fog computing architecture, can be characterized by their non-functional properties. Such non-functional properties are for example, real-time

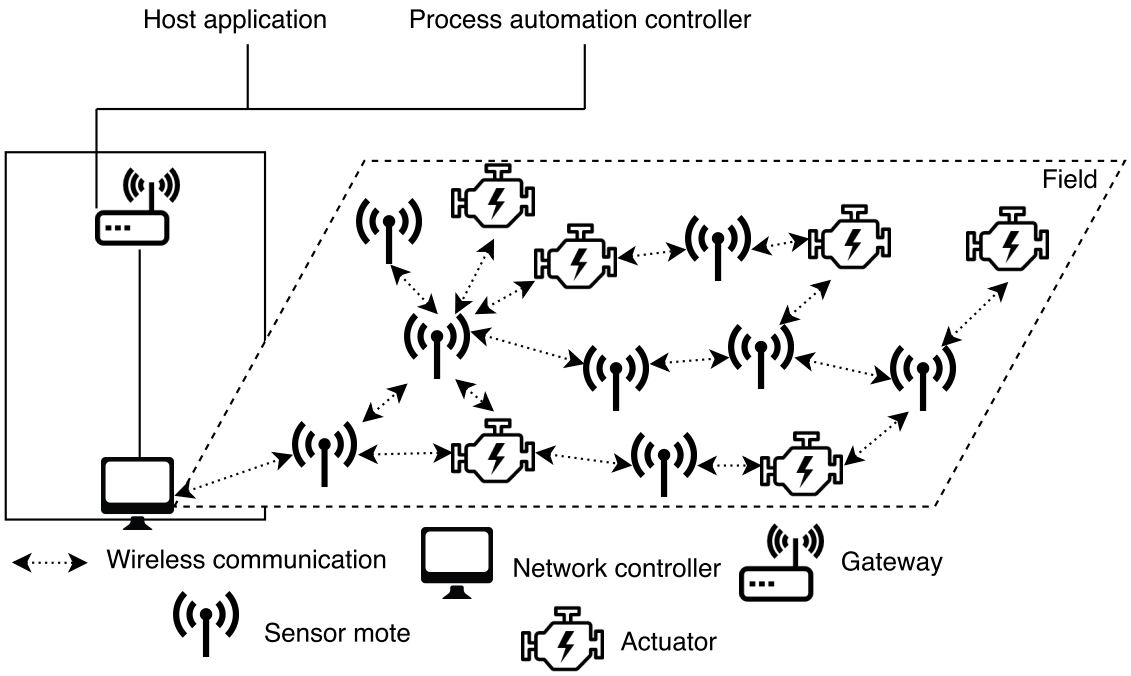


Figure 1: A typical industrial network setting.

behavior, reliability, and availability. Furthermore, fog nodes can follow industry-specific standards (e.g., IEEE 802.15.4e (6) or WirelessHART (7)) that demand the implementation as well as verification and validation of software and/or hardware to follow formal rules.

1.2 Distributed data management

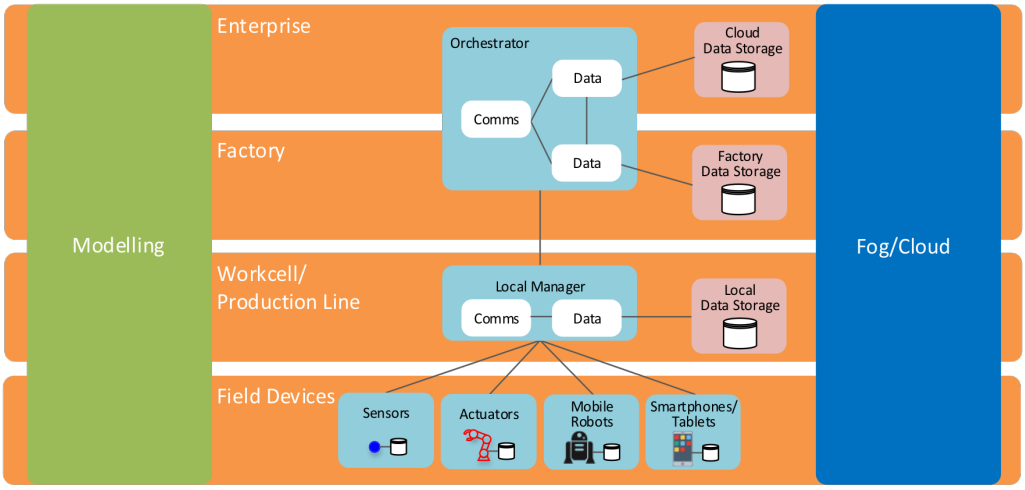
Distributed data management, a key component of fog computing, can be a very suitable approach to cope with these issues (8). In the context of industrial networks, one could leverage the set of nodes present at the edge of the network to distribute functions that are currently being implemented by a central controller (1). Many flavors of distributed data management exist in the networking literature, depending on which edge devices are used. In the context of AUTOWARE, we introduce a decentralized data distribution, and we use the *multitude of sensor nodes present in an industrial physical environment* (e.g., a specific factory) to implement a *distributed data management* whereby sensor nodes cache data they produce, and provide these data to each other upon request. In this case, the choice of the sensor nodes where data are cached must be done to guarantee a maximum delivery latency to nodes requesting those data.

The AUTOWARE smart data distribution component is a technological building block spanning all layers from field devices to cloud. It is based on a hierarchical set of distributed entities related to software developers, technology providers, as well as integrators. It implements the logic of a control plane for cognitive data distribution across all the communication layers (field, workcell, factory, enterprise), based on the available communication technologies and supporting the needs of the higher-layer services and applications. As shown in Fig. 2a, at the logical layer, Data Managers (DM) decide where data is replicated, moved, and stored. Each DM decides which decisions can be taken autonomously by individual nodes embedding cognitive functions based on the requirements of the specific application and supports complete decentralization of functionalities to individual nodes. It also implements the logic of a control plane for cognitive data distribution. As shown in Fig. 2b, it considers a set of "pipes" where data flows. The control plane decides trough which nodes the "pipes" should pass and through which underlying communication and networking primitives the data should flow. Different types of data-oriented automation functions at reduced costs can be implemented, like interactions with external data providers or requestors, inter-cell data distribution planning and management and coordination of the DMs. The core data management functions are shown in Fig. 3, where at the same time, the positioning with respect to also the communications management functions developed in T2.3 is visible.

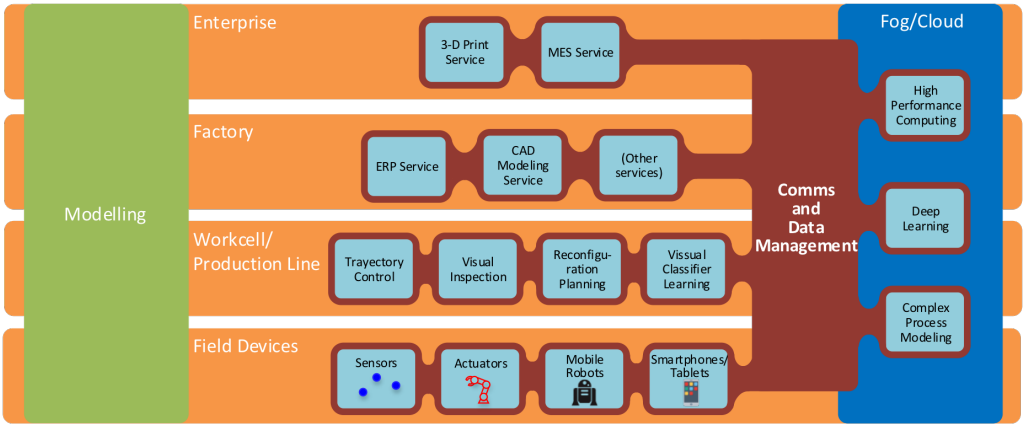
The AUTOWARE smart data distribution component is decentralized, dynamic and hierarchical, so as to support cooperation with cloud-based service provisioning and communication technologies. More specifically, at first, T2.4 provides decentralized data distribution among the various nodes of the AUTOWARE ecosystem. This decentralization is targeting the efficient management of massive data generation and consumption and the exploitation of resources available locally at individual nodes. Then, it copes with dynamically changing network parameters by employing adaptive data placement and replication techniques. Finally, it provides a hierarchical data management, by employing a mix of centralized and decentralized control processes and a hierarchy of data managers. D2.4 addresses three core issues and presents the corresponding key technical contributions: (i) testing of the overall concept, by considering a flat network topology, (ii) hard data access latency deadlines, and exploitation of the presence of fog nodes, and (iii) dynamic network reconfigurations. Those contributions are validated by both large-scale simulation models and real-world results on an experimental testbed.

1.3 A small scale proof of concept at IK4-TEKNIKER

Before proceeding to large-scale experimental solution testing through testbed and simulations in the context of T2.4, we designed and implemented an protorype proof of



(a) Control Plane.



(b) User Plane.

Figure 2: AUTOWARE hierarchical communication and data management architecture.

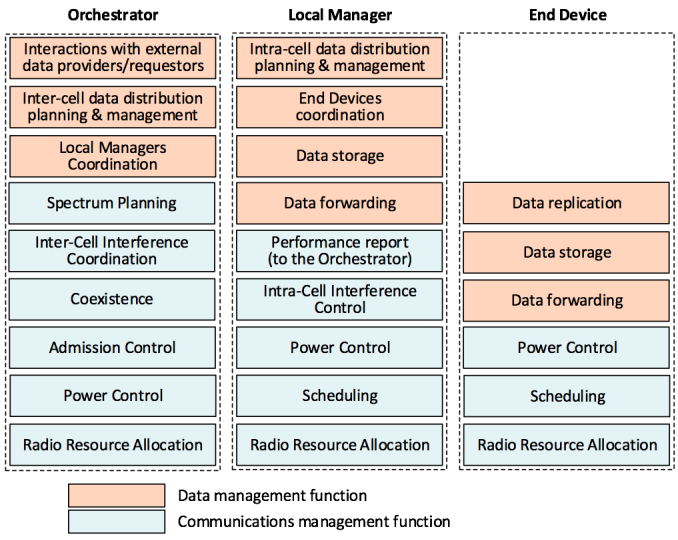


Figure 3: Different AUTOWARE communication and data management functions.

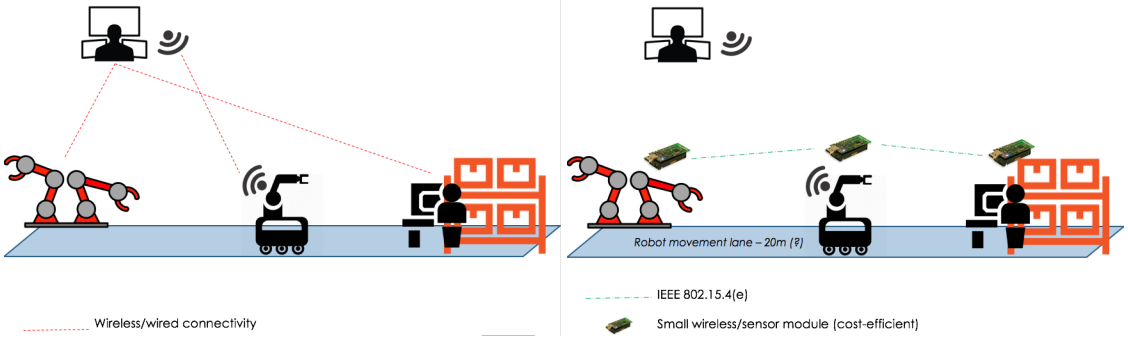
concept, so as to verify the applicability of distributed data management in industrial environments. In order to demonstrate how we can apply the data management concepts in prototype industrial environments and showcase their feasibility, we designed and implemented a small scale demonstration at the premises of IK4-Tekniker. The core of the concept is regarding how we can use smart data distribution at the IK4-Tekniker industrial facilities so as to achieve: (i) cost-effective operations, (ii) fault tolerance, (iii) dynamic, plug&play smart data distribution solutions.

A typical scenario at IK4-Tekniker is shown in Fig. 4a. A mobile robot is responsible for fetching objects to a robotic bi-manipulator. The objects are located at a set of shelves, where a human operator is responsible for manually loading the objects on the mobile robot. Both the robot and the operator are aware of which object is currently needed at the bi-manipulator, as there is a centralized wireless or wired communication infrastructure, coordinated by a central controller and the data can be sent and received through the communication links. However, due to the harsh conditions in several industrial environments, it is not unusual for the main centralized operational network to go offline, for a variety of reasons. When there is a situation like this in the current scenario, the mobile robot and human operator cannot be aware of which object is currently needed at the bi-manipulator, which in turn results in a failure of the production process.

In order to address this problem, we suggested the adoption of a distributed data management approach. More specifically, we employed a secondary, lightweight data distribution layer and implement it by using small, low-cost wireless sensor motes. The proof of this concept was showcased by placing three motes in the network as shown in Fig. 4b, one on the bi-manipulator, one on the mobile robot and one on the set of shelves. The motes used were IEEE 802.15.4 enabled, a fact that renders them compatible with typical industrial networking protocols, such as IEEE 802.15.4e and WirelessHART. This demonstration is proving that with low-cost we can achieve high fault tolerance and reliability.

1.4 Roadmap

In Section 2, we provide an overview of the state of the art. In Section 3, we introduce the *Data Management Layer (DML)*, which operates independently from and complements the routing process of industrial IoT networks. Assuming that applications in such networks require that there is (i) a set of *producers* generating data (e.g., IoT sensors), (ii) a set of *consumers* requiring those data in order to implement the application logic (e.g., IoT actuators), and (iii) a maximum latency $L_{\max}$ that consumers can tolerate in receiving data after they have requested them, the DML offers an efficient method for regulating the data distribution among producers and consumers. The DML selectively assigns a special role to some of the network nodes, that of the *proxy*. Each node that can become a proxy potentially serves as an intermediary between producers and consumers, even



(a) Centralized network operational. (b) Centralized network not operational.

Figure 4: The proof of concept demonstration at IK4-TEKNIKER.

though the node might be neither a producer nor a consumer. If properly selected, proxy nodes can significantly reduce the access latency; however, when a node is selected as a proxy, it has to increase its storing, computational and communication activities. Thus, the DML minimizes the number of proxies, to reduce as much as possible the overall system resource consumption[1].

In Section 4, we consider the problem of network lifetime maximization, given the proxy locations in the network, the initial limited energy supplies of the nodes, the data request patterns (and their corresponding parameters), and the maximum latency that consumer nodes can tolerate since the time they request data. We prove that the problem is computationally hard and we design an offline centralized heuristic algorithm for identifying which paths in the network the data should follow and on which proxies they should be cached, in order to meet the latency constraint and to efficiently prolong the network lifetime. We implement the method and evaluate its performance using an FIT IoT-LAB testbed (10), comprised of IEEE 802.15.4-enabled WSN430 network nodes. We demonstrate that the proposed heuristic (i) guarantees data access latency below the given threshold, and (ii) performs well in terms of network lifetime with respect to a theoretically optimal solution.

Another typical problem of entirely centralized and offline computations regarding data distribution scheduling, is that they can become inefficient in terms of energy, when applied in industrial IoT networks. In industrial environments, the topology and connectivity of the network may vary due to link and sensor-node failures (11). Also, as a result of factory reconfigurations required in the Industry 4.0 framework, very dynamic conditions which make communication performance much different from when the central schedule was computed, possibly causing sub-optimal performance, may result in not guaranteeing application requirements. These dynamic network topologies may cause a portion of industrial sensor nodes to malfunction. With the increasing number of involved battery-powered devices, industrial IoT networks may consume substantial amounts of energy; more than would be needed if local, distributed computations were used.

In Section 5 we focus on maintaining the network configuration in a way such that application requirements are met even if important network operational parameters change due to some unplanned events (e.g., heavy interference, excessive energy consumption), while guaranteeing an appropriate utilization of energy resources. We provide several efficient algorithmic functions which locally reconfigure the paths of the data distribution process, when a communication link or a network node fails. The functions regulate how the local path reconfiguration should be implemented and how a node can join a new path or modify an already existing path, ensuring that there will be no loops. The proposed method can be implemented on top of existing data forwarding schemes designed for industrial IoT networks. We demonstrate through simulations the performance gains of our method in terms of energy consumption and data delivery success rate.

[1] Note that, the coherency of data that reside on proxies can be achieved in a variety of ways (9), and is beyond the scope of this work.

2 State of the art

2.1 Proxies in industrial networks

Traditionally, industrial application systems tend to be entirely centralized. For this reason, distributed data management has not been studied extensively in the past, and the emphasis has been put on the efficient computer communication within the industrial environment. The reader can find state of the art approaches on relevant typical networks in (4), (12) and (13).

Although distributed data management can provide a new set of approaches for industrial applications, similar concepts have been used in some networking fields. For example, in the field of *Content Distribution Networks*, authors of (14) have proposed SCAN. SCAN utilizes an underlying distributed object routing and location system, which combines dynamic replica placement with a self-organising application level multicast tree to meet client QoS and server resource constraints. Proxy placement has been a usual research topic in various fields. Some representative fields are *Information Retrieval over the Internet* (15), *Content Distribution Networks* (16) and *Multimedia Streaming* (17). In most of the cases, proxies can significantly improve metrics such as system resources, traffic patterns, data segmentation, etc. However, the solutions proposed to date are not suitable for the industrial environments due to the different constraints, optimization objectives and technological implementations.

2.2 Latency guarantees

Some interesting related works are the following: In (18), the authors present a centralized routing method, and, consequently, they do not use proxies. In (19), the authors address a different optimization objective, focusing on minimizing the maximum hop distance, rather than guaranteeing it as a hard constraint. Also, they assume a bounded number of proxies and they examine only on the worst-case number of hops. Finally, the presented approach is somewhat graph-theoretic, which makes it hard to apply on real industrial IoT networks. In (20), the authors present a cross-layer approach which combines MAC-layer and cache management techniques for adaptive cache invalidation, cache replacement and cache prefetching. Again, the model is different, as we assume a completely industrial oriented MAC layer, based on, e.g., IEEE802.15.4e, and a different problem, focusing on the delay aspects, instead of cache management. In (21), the authors consider a different problem that ours: replacement of locally cached data items with new ones. As the authors claim, the significance of this functionality stems from the fact that data queried in real applications are not random but instead exhibit locality characteristics. Therefore, the design of efficient replacement policies, given un underlying caching mechanism is addressed. In (22), although the authors consider delay aspects and a realistic industrial IoT model (based on WirelessHART), their main objective is to bound the worst-case delay in the network. Also, they do not exploit the potential presence of proxy nodes, and consequently, they stick to the traditional, centralized industrial IoT setting. In (23), the authors consider a multi-hop network organized in clusters and provide a routing algorithm and cluster partitioning. Our DML concepts and algorithms can work on top of this approach (and of any clustering approach), for example by allocating the role of proxies to cluster-heads. In fact, clustering and our solutions address two different problems.

2.3 Maximizing the lifetime

The most relevant work regarding lifetime maximization is (24), in which the authors focus on how to efficiently place the proxies in industrial and content delivery networks, in order to perform well with respect to the data access latency. However, this work does not consider techniques on how to prolong the network lifetime. There are also a few works in

the wireless sensor networking field focusing on how to move data to multiple locations, most of which targeting energy efficient routing to multiple control centers. None of those works is taking into account neither latency constraints nor deployed proxies. In (25), the main objective is to maximize the network lifetime by a fair allocation of resources, using a specific metric of optimality, the lexicographic optimality. In (26), the authors present a multi-source, multi-consumer model, but they do not provide a source-consumer mapping. Consequently, the target objective is fulfilled if all sources successfully propagate their data to any consumer. In (27), they consider a different energy efficiency target objective. More specifically, they try to minimize the number of links used in the routing process by maximizing the overlapping links among source-sink paths. This strategy in our case could lead to overstressed nodes which in turn lead to reduced lifetime.

2.4 Dynamic reconfigurations

As there are numerous relevant previous works in the literature, we provide some information about the most representative and most related ones to this work, which are (24), (28), (22) and (29). Although some of those works use proxy nodes for the efficient distributed management of network data, they all perform path selection computations centrally. Placement and selection strategies of caching proxies in industrial IoT networks have been investigated in (24). Data re-allocation methods among proxies for better traffic balancing are presented in (28). Delay aspects in a realistic industrial IoT network model (based on WirelessHART), and bounding of the worst case delay in the network are considered in (22). Reliable routing, improved communication latency and stable real-time communication, at the cost of modest overhead in device configuration, are demonstrated in (29). Different to those works, in the context of AUTOWARE, we present methods which exploit the local knowledge of the network nodes so as to perform distributed, local path reconfiguration computations towards more efficient energy dissipation across the network.

2.5 Motivating examples

Two indicative application areas where the DML and the relevant algorithms could provide additional value are the following: In (30), the authors present a typical situation in an oil refinery where miles of piping are equipped with hundreds and thousands of temperature, pressure, level and corrosion sensors, which are deployed in a large geographical area. Those sensor motes not only perform industrial condition monitoring tasks, but they also use the industrial oriented communication technology TSCH. In (31), the authors present two large-scale deployments of hundreds of nodes; one in a semiconductor fabrication plant, and another, on-board an oil tanker in the North Sea. They use Mica2 and Intel Mote nodes, very similar to our sensor motes of choice. In both those applications, due to the exact fact that the sensor motes are not able to communicate directly with the controller (transmission range restrictions), the system designers naturally consider a multi-hop propagation model. The targeted decentralization of the data distribution process in those large-scale condition monitoring and predictive maintenance application areas could lead to economic benefits for the industrial operator and maintenance of some important metrics in the network at good levels, while ensuring that the end-to-end latency is acceptable, without introducing overwhelming costs in the system for the purchase of expensive equipment.

3 Average data access latency guarantees

In this section, in order to manage the data distribution process and decrease the average latency in the network, we introduce the *Data Management Layer (DML)*, which operates independently from and complements the routing process of industrial IoT networks. The main idea behind the DML is decoupling the Network plane from the Data Management Plane. Fig. 5 depicts the DML high-level functionality. Assuming that applications in such networks require that there is (i) a set of *producers* generating data (e.g., IoT sensors), (ii) a set of *consumers* requiring those data in order to implement the application logic (e.g., IoT actuators), and (iii) a maximum latency $L_{\max}$ that consumers can tolerate in receiving data after they have requested them, the DML offers an efficient method for regulating the data distribution among producers and consumers. The DML selectively assigns a special role to some of the network nodes, that of the *proxy*. Each node that can become a proxy potentially serves as an intermediary between producers and consumers, even though the node might be neither a producer nor a consumer. If properly selected, proxy nodes can significantly reduce the access latency; however, when a node is selected as a proxy, it has to increase its storing, computational and communication activities. Thus, the DML minimizes the number of proxies, to reduce as much as possible the overall system resource consumption[2]. More specifically, the main innovations are the following:

- We propose a *distributed data management* approach to store data on a number of locations in an industrial environment, as opposed to the current industrial state-of-the-art approaches where all data are centrally stored and served from a unique location. We introduce the DML for *minimizing the number of proxies* in an industrial IoT network and to *reduce* as much as possible the overall system *resource consumption*.
- We provide a *multi-faceted performance evaluation*, both through *experiments*, and through *simulations* for achieving scales much larger than what available experimental conditions allow. At first, we implement the DML with 95 real devices and evaluate its performance in FIT IoT-LAB testbed (10). Then, we use the simulation model, we validate it against the experimental results and we evaluate the DML performance in *larger network sizes* and more *general topologies*.
- We demonstrate that the proposed method (i) guarantees that the *access latency stays below the given threshold*, and (ii) significantly *outperforms traditional centralized and even distributed approaches*, both in terms of average data access latency and in terms of maximum latency guarantees.
- We also demonstrate an *additional flexibility* of the proposed approach by showing that it can be tuned both to guarantee that the *average* latency stays below $L_{\max}$, or that the *worst-case* latency stays below $L_{\max}$.

In Section 3.1, we provide the model of the settings we consider, as well as the necessary notation. In Section 3.2, we introduce the DML and the problem that it addresses. In Section 3.3, we evaluate the performance of the DML in comparison with two other methods used in industrial environments. We also validate the simulation model used afterwards. In Section 3.4 we present simulation results in scenarios that are not possible to evaluate with the available experimental testbeds.

3.1 System modeling

The network. We consider networks of industrial IoT devices which usually consist of sensor motes, actuators and controller devices. We model those devices as a set of

[2]Note that, the coherency of data that reside on proxies can be achieved in a variety of ways (9), and is beyond the scope of the current work.

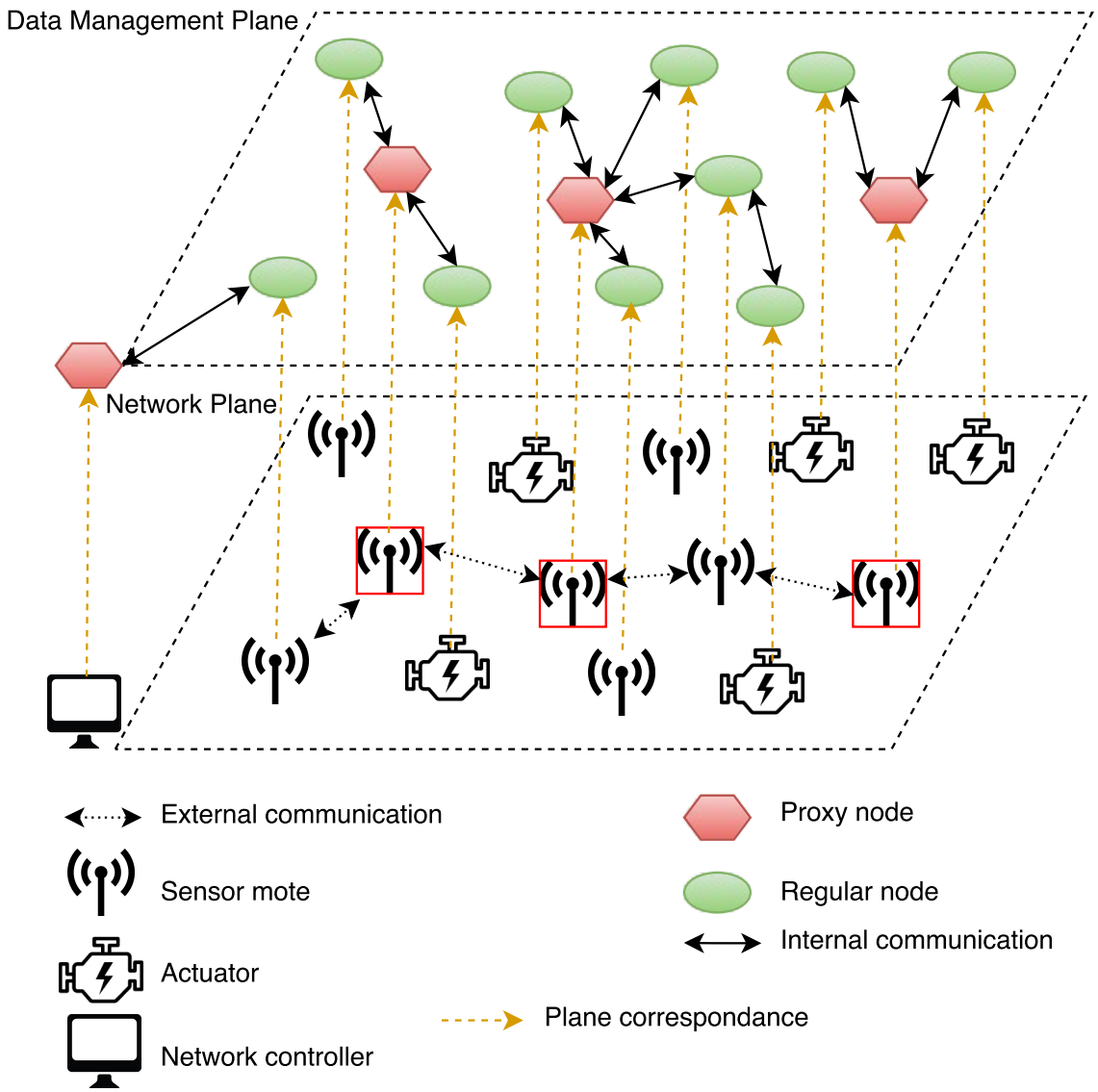


Figure 5: Decoupling of the Data Management Plane and the Network Plane.

$S = \{s_1, s_2, ..., s_n\}$ nodes, with a total number of $|S| = n$ nodes, deployed in an area of interest $\mathcal{A}$. The central network controller C is set as the first device in the network, $C = s_1$. The communication range r_u of a node u depends on the transmission parameters and the radio propagation conditions of the underlying routing protocol and the constraints of the technological implementation. Nodes $u, v \in S$ are able to communicate with each other if $r_u, r_v \geq \epsilon(u, v)$, where $\epsilon(u, v)$ is the Euclidean distance between u and v.

We assume that the controller C is able to maintain *centralized network knowledge*. This is usual in industrial applications, in which the locations of the nodes are known, traffic flows are deterministic and communication patterns are established a priori. We assume that C knows all the shortest paths in the network and comes with an $n \times n$ matrix $\mathbf{D}$, where $\mathbf{D}_{u,v}$ is the length of the shortest path between nodes u and v[3]. Note that only the control of the data management plane is centralized, while, when using the DML, the data management plane itself is distributed and cooperative, as data are stored on proxies $p \in P \subset S$, which cooperate to achieve the optimal performance of the network according to the objective function defined next. The network controller C is also serving

[3]The offline shortest path computation between two nodes is a classic problem in graph theory and can be solved polynomially, using Dijkstra's algorithm (32).

as a proxy, with, and thus we have that $|P| \geq 1$ in all cases.

Data production and consumption. In typical industrial applications, like condition monitoring, sensor nodes perform monitoring tasks (producers) and in some cases their sensor data are needed either by other sensor nodes, which could need additional data to complement their own local measurement, or by actuator nodes, which use the sensor data so as to perform an actuation (consumers). When needed, a consumer u can ask for data of interest using the primitives defined by the underlying routing protocol from a sensor node v (from a proxy p, when using the DML)[4]. We define the set of m consumers as $S_c \subset S$, with $|S_c| = m < n$. When a proxy p receives a data request from a consumer u, it propagates the requested data back, along the same routing path, starting at v and finishing at the consumer u. Note that the length of this individual data delivery path is twice the length of the path between u and p. We assume that the data generation and access processes are not synchronized. Specifically, we assume that data consumers request data at un unspecified point in time after data has been generated by data producers and transferred to the proxies.

The latency constraint. Let $l_{u,v}$ be the single-hop data transmission latency from a node $u \in S$ to another node $v \in S$ (34). We define as access latency the amount of time required for the data to reach consumer u, after u's request, when the data follow a multi-hop propagation between u and v ($v = p$, in the DML case). We denote access latency as $L_{u,v} = l_{u,s_i} + ... + l_{s_j,v} + l_{v,s_j} + ... + l_{s_i,u}$. We define as average access latency across all consumers the quantity $\overline{L} = \frac{\sum_{\forall u \in S_c} L_{u,v}}{m}$. Industrial applications are typically time-critical, and consequently the industrial operator requires a maximum data access latency threshold L_{max}. This is an important constraint in the network and the implementation of a data delivery strategy should ensure that the average multi-hop access latency does not exceed the threshold value. In other words, the following inequality should hold: $\overline{L} \leq L_{\text{max}}$. Note that this formulation is amenable to different purposes. If $L_{u,v}$ is the mean latency between u and v, the above inequality guarantees that the average of the mean latencies is below L_{max}. If it is the worst-case latency between u and v, the inequality provides a guarantee on the average worst-case latency. In the following we show that the DML can be used in both cases.

3.2 The Data Management Layer

The basic function of the DML is the selection of some nodes which will act as *proxies*, and the establishment of an efficient method for data distribution and delivery, using the proxies. More specifically, the role of the DML is to define a set $P \subset S$, the elements of which are the selected proxies. The number of the proxies can range from 1 to $n - 1$. The case of 1 proxy is equivalent to having only the controller C operating as a single point of data distribution. In this case, the data distribution is functioning as in traditional industrial IoT environments.

This demarcated model of data exchanges can be formulated as a *publish/subscribe (pub/sub)* model (35). In a pub/sub model, a consumer subscribes to data, i.e., denotes interest for it to the corresponding proxy, and the relevant producer publishes advertisements to the proxy. The DML assumes that the pub/sub process is regulated at the central controller C, which maintains knowledge on the sets of producers, consumers and requests. Based on this, C can find an appropriate set of proxies based on the algorithm we present next. Inside the network, the proxies are responsible for matching subscriptions with publications i.e., they provide a rendezvous function for storing the available data according to the corresponding subscriptions. The producers do not hold references to the consumers, neither do they know how many consumers are receiving their generated data.

All proxies do not necessarily store the same data; they store different data depending on the consumers they are assigned to. The selection of the proxies should be done

[4]For example, nodes could use RPL, in storing or non-storing mode (33) and a low-power multi-hop MAC protocol, like IEEE 802.15.4e (13).

balancing two requirements. On the one hand, the number of proxies should be sufficient to make sure each consumer finds data "close enough" to guarantee that $\overline{L} \leq L_{\text{max}}$. On the other hand, as the role of proxy implies a resource burden on the selected nodes, their number should be as low as possible. The proxy selection problem can thus be formulated as an integer program. More specifically, given a set S of nodes, a set $S_c \subset S$ of consumers and an access latency threshold L_{max}, the network designer should target the minimization of the number of proxies needed in the network so as to guarantee $\overline{L} \leq L_{\text{max}}$. We define two sets of decision variables, (a) $x_p = 1$, if $p \in S$ is selected as proxy and 0, otherwise, (b) $y_{u,p} = 1$, if consumer $u \in S$ is assigned to proxy $p \in S$ and 0, otherwise. Then, the integer program formulation is the following:

$$\text{Min.:} \quad \sum_{p \in S} x_p \tag{1}$$

$$\text{S. t.:} \quad \sum_{u \in S_c} \sum_{p \in S} \frac{L_{u,p} \cdot y_{u,p}}{m} \leq L_{\text{max}} \tag{2}$$

$$\sum_{p \in S} y_{u,p} = 1 \qquad \forall u \in S_c \tag{3}$$

$$y_{u,p} \leq x_p \qquad \forall u \in S_c, \forall p \in S \tag{4}$$

$$x_p, y_{u,p} \in \{0, 1\} \qquad \forall u \in S_c, \forall p \in S \tag{5}$$

The objective function (1) minimizes the number of proxies[5]. Constraint (2) guarantees that $\overline{L} \leq L_{\text{max}}$. Constraints (3) guarantee that each node has to be assigned to one and only one proxy. Constraints (4) guarantee that nodes can be assigned only to proxies. Constraints (5) guarantee that all nodes are considered for potentially being selected as proxies and that all nodes requesting data are assigned to a proxy.

The proxy selection problem is computationally intractable, since it can be formulated as an integer program. This means that it is impossible to optimally calculate in polynomial time the minimum proxies needed while staying below L_{max}. Also, the formulation of the problem considers the latency of communication $L_{u,p}$, and not an abstract number of hops. This makes the formulation realistic for industrial environments, but even more difficult a problem, as it becomes also infeasible to assign the real values to $L_{u,p}$ of constraints (2). This is due to the fact that we are not able to know the exact values of the individual transmission latencies $l_{u,v}$, before they happen. To address this issue, we provide a heuristicwhich takes into account the latencies $L_{u,p}$, instead of number of hops, thus defining the `ProxySelection+` algorithm (Algorithm 1). `ProxySelection+` is a myopic algorithm which does not give the optimal solution. The use of simple heuristics like the one in `ProxySelection+` shows that the DML is able to outperform the traditional centralized methods, even when adopting simple methods.

`ProxySelection+` sets the controller C as the first proxy the network, and it gradually increases the number of proxies (counter) until it reaches a number with which the average access latency $\overline{L}$ does not violate the maximum latency threshold L_{max}. In every iteration (lines 5-9), the exact selection of the of the next proxy in the network is performed using a myopic greedy addition (lines 6-8). Each candidate node is examined and the one whose addition to the current solution reduces the average access latency the most is added to the incumbent solution. To this end, the latency between a candidate proxy (k in line 7) and a consumer (u in line 7) is estimated as the length of the shortest path $\mathbf{D}_{k,u}$ that is connecting them multiplied with the expected latency on each hop ($l^{(h)}$ in line 7). $l^{(h)}$ needs to be initiated through preliminary measurements. This happens through an initialization phase (line 1) during which the network designer measures different single-hop data transmission latencies within the industrial installation and gathers a sufficiently representative dataset of $l_{u,v}$ measurements from different pairs of nodes $u, v \in S$ across

[5]Note that in various industrial scenarios, some nodes might too weak to perform any other operations than generating and propagating a minimal set of data. The problem formulation which represents those scenarios is a special case of the problem formulation that we consider in this work, with $p \in S'$, where $S' \subset S$.

Algorithm 1: `ProxySelection+`

Input : $S, S_c, r_u, L_{\text{max}}$

$l^{(h)} \leftarrow$ assign value through initialisation phase at the industrial installation

$\mathbf{D}_{u,v} = \text{Dijkstra}(S, r_u), \forall u, v \in S$ (32)

$P = \{C\}$

counter $= 1$

while counter $< n$ **and** $\overline{L} > L_{max}$ **do**

  for $i = 2$: counter **do**

    $p = \arg\min_{v \in S} \sum_{u \in S_c} \min_{k \in P \cup \{v\}} \frac{l^{(h)} \cdot \mathbf{D}_{k,u}}{m}$

    $P = P \cup \{p\}$

  counter $++$

Output: P

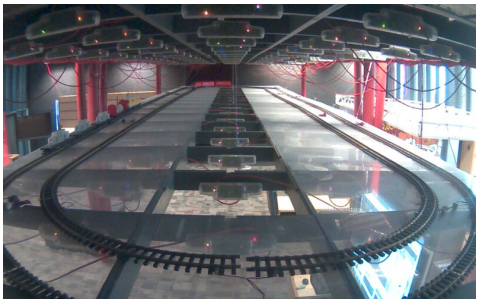
the network. By using the mean of measured latencies, we obtain a guarantee on the average mean latency. By using the highest measured value, we obtain a constraint on the average worst-case latency, implementing the guarantee explained in Section 3.1. The computational complexity of the `ProxySelection+` is polynomial with worst case time of $\mathcal{O}(V^4)$. However, this worst-case performance is very difficult to be experienced in practice, since in order to have $\mathcal{O}(V^4)$ time, all n nodes of the network have to be chosen as proxies; something highly unlikely.

3.3 Implementation and experimental evaluation

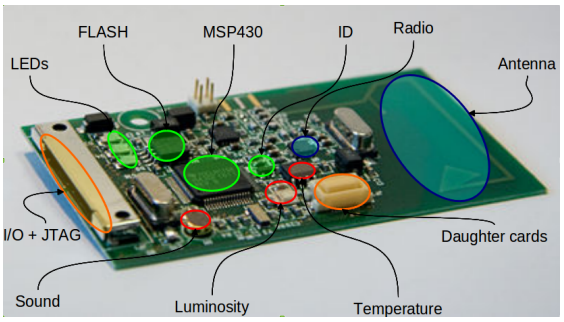
3.3.1 Experimental strategy

Strategic purpose. The strategic purpose of the experimental evaluation with real devices is to provide a *realistic* demonstration on how efficient data management methods can significantly improve the data access latency in industrial IoT networks, by using a limited number of proxies. The realistic approach is of paramount importance in our implementation strategy. For this reason, we follow some important steps. For the experimental implementation and evaluation we use the Euratech testbed from the FIT IoT-LAB platform (10). IoT-LAB a large-scale collection of open wireless IoT testbeds, operated by the French CNRS and INRIA research institutions (10). The Euratech testbed is deployed in the Inria Lille - Nord Europe showroom and 224 nodes are deployed as follows: two horizontal layers in grid formation of 5×19 nodes each and 34 nodes attached to a wall, at a distance of 0.60 m to each other. Fig. 6a displays a photo of the testbed. We use a network with technical specifications representative of the industrial IoT paradigm (e.g., low-power nodes, IEEE 802.15.4 radio interface, large number of devices, etc.). Among all supported operating systems (e.g., TinyOS, OpenWSN, Contiki), we conducted our implementation on TinyOS. We carefully choose the L_{max} threshold, according to actual industrial requirements and expert groups' recommendations. In order to have a benchmark for the performance of our method, we also implement two additional representative methods, based on routing mechanisms that are usual in current industrial IoT networks. We vary several experimental parameters so as to investigate the performance consistence of our method under different settings. Finally, we validate a simulation model based on the real world settings, with which we can further investigate changing parameters that are impossible or too time-consuming to investigate on the testbed.

Experiment design. We use a total number of $n = 95$ nodes in Euratech testbed which form a 2D horizontal grid, as shown in Fig. 6a. Occasionally, during the experiments, there are some dead nodes, that is nodes that have ran out of available power and are not able to function. This occasional unavailability of a subset of nodes renders the experiment even more realistic, since dead node situations frequently occur in real industrial IoT networks. The nodes that were used in the experiments are WSN430 open nodes, the

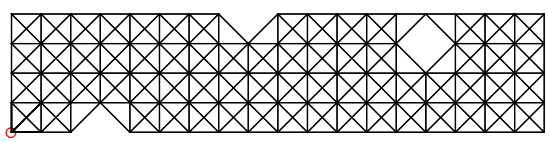


(a) Euratech testbed (10).

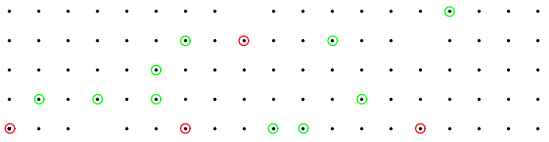


(b) WSN430 open node.

Figure 6: Experimental setup.



(a) Network topology of the Euratech testbed. In the lower left corner the network controller C is visible as a red dot.



(b) Output of the `ProxySelection+` algorithm. The green dots represent the consumers and the red dots represent the proxies.

Figure 7: Euratech topology and `ProxySelection+` output.

design of which is displayed in Fig. 6b. The WSN430 open node is a mote based on a low power MSP430-based platform, with a set of standard sensors and an IEEE 802.15.4 radio interface at 2.4 GHz, using a CC2420 antenna (36), which can support IEEE 802.15.4e and WirelessHART settings, typical of industrial communications. We programmed and operated the nodes under TinyOS 2.1.1, a reference operating system for sensor nodes.

Since the testbed nodes are placed in a short distance to each other, we adjust their transmission range, so as to obtain a realistic multi-hop topology. We configured the antenna TX power such that, according to the CC2420 antenna datasheet (36) and the measurements provided in (37), the transmission range is about 3m. However, given that this value has been measured in ideal conditions, without taking into account external factors such as obstacles and interference, we program every node $u \in S$ to consider as neighbor every other node $v \in S$ with $\epsilon(u, v) \leq 1$m. Given this configuration, we obtain a topology which is depicted in Fig. 7a. Note that, the three "gaps" in the topology are a result of the dead nodes of the deployment which are unable to communicate with other nodes. We set the percentage of requesting nodes to $m = 0.1 \cdot |S|$, selected uniformly at random from S, and we set $C = s_1$ as the central network controller, which corresponds to the node with `node_id` $= 1$ in the Euratech testbed (lower left node in Fig. 7a).

Setting the L_{max} threshold. In order to perform the experiments in the most realistic way, it is important that the L_{max} value is aligned with the official communication requirements of future network-based communication solutions for Industry 4.0, for the targeted industrial applications. Both the *WG1 of Plattform Industrie 4.0* (reference architectures, standards and norms) (38) and the *Expert Committee 7.2 of ITG* (radio systems) (39) set the latency requirements for condition monitoring applications to 100ms. However, in order to provide a complete and diverse set of results, we also measure the performance of our method for different values of L_{max}.

Performance benchmarks. In order to measure the performance of the DML with respect to traditional industrial IoT alternatives, we implement two additional data delivery strategies. The first method is the most traditional data delivery strategy in current industrial IoT environments and imposes that all data requests and data deliveries are being routed through the controller C. More specifically, the request is routed from consumer u

Table 1: Measured send/receive latency.

Type of measured latency	Notation	Value (ms)
Highest latency reported	$l^{(\max)}$	23
Mean latency	$l^{(\text{mean})}$	17.4
Lowest latency reported	-	13

to C and then from C to producer v. At the next step, the data is routed from v again to C and then from C to u. We call this mode of operation *non storing mode* and it is obvious that it is completely centralized and not cooperative. Note that this would be the simplest data management approach that can be implemented in relevant routing mechanisms like RPL (33), where intermediate nodes are not allowed to cache data (thus, the RPL terminology non-storing mode).

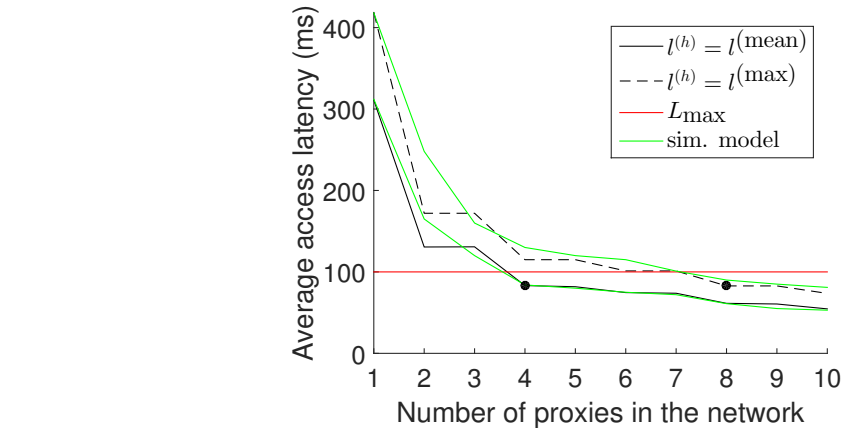
The second method is another, less commonly used in industrial IoT settings, but nevertheless useful alternative. It imposes that all data requests and data deliveries are being routed through the lowest common ancestor (LCA) of the routing tree, routed at the controller C. The LCA of two nodes u and v in the routing tree is the lowest (i.e., deepest) node that has both u and v as descendants. We call this mode of operation *storing mode*, because the LCAs should store additional information about their descendants, and it is obvious that it is a distributed alternative. Again, this is the simplest method that one would implement with routing mechanisms like RPL in storing mode, i.e., when intermediate nodes between communication endpoints are allowed to cache content. Storing mode thus provides a distributed method.

We made those choices after careful consideration of the current realistic industrial networking status-quo. The selected protocols are standardized components of a reference and well-established communications and data management stack. In fact, they are considered the state-of-the-art, for current and future wireless industrial applications, as discussed extensively in (13) and (12). More specifically, the stack is presented in detail in (12), and is considered as the de-facto standard for future industrial networks. In the following, for convenience, we use the term "special nodes" when we refer to the network controller, the proxies or the LCAs.

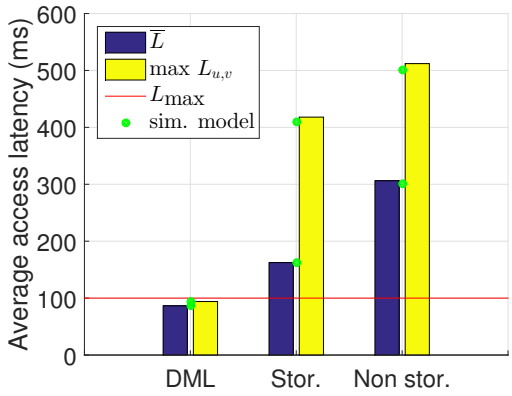
3.3.2 Experimental results

Running the `ProxySelection+` algorithm. We ran the initialization phase of `ProxySelection+`, so as to assign values to $l^{(h)}$, by measuring times that are needed for the data exchange of a sensor measurement from a sensor node to another. We measured the time needed for the sensor reading to be sent and received from a node to another. This latency includes time spent in the back-off phase (which cannot be predicted), time spent in sending the signal over the radio, and time spent during the propagation. We consider the propagation latency negligible, since radio waves are traveling very fast and we are not able to measure the time elapsed using the nodes' timers. In order to obtain reliable results, we repeated the propagation measurements for different pairs of transmitting and receiving nodes of Euratech testbed, for 30 times for each pair. We concluded to the measurements that are shown in the Table 1 (highest, lowest and mean values), after measuring the relevant times using WSN430 with CC2420 and TinyOS. We can see that the latency values of data propagation from one node to another significantly vary. While the lowest latency can be 13ms, the highest propagation latency $l^{(\max)}$ is 23ms. The mean latency $l^{(\text{mean})}$ of the values collected from the repetition of this experiment is 17.4ms[6]. After running `ProxySelection+` with $L_{\max} = 100$ms, $m = 0.1 \cdot |S|$, and $l^{(h)} = l^{(\text{mean})}$, we get the proxy placement that is depicted in Fig. 7b. We can easily see that `ProxySelection+` is balancing the proxy allocation in the network, so as to guarantee a small data access

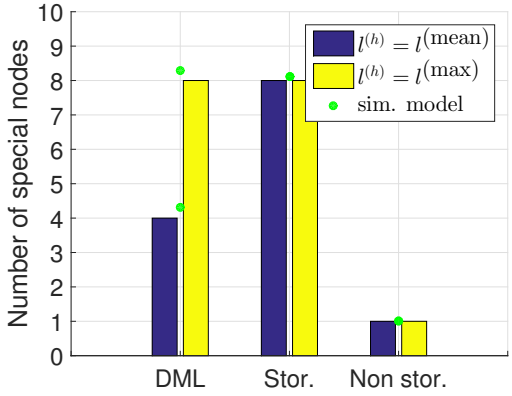
[6]Other sources of latency related, e.g., to computation at the receivers have been found to be in the order of μs, and therefore are neglected. Also, the measuring methodology we used does not depend on the specific conditions under which these measures are taken.



(a) $\overline{L}$, different $|P|$, $m = 0.1 \cdot |S|$.



(b) $\overline{L}$ and maximum $L_{u,v}$.



(c) Number of special nodes used.

Figure 8: Experimental results in the IoT-LAB Euratech testbed. The validation of the simulation model is displayed in green.

latency to all the requesting nodes.

Increasing and decreasing the number of proxies. Fig. 8a displays the average access latency $\overline{L}$ for different number of proxies in the network. In order to obtain this plot, we run `ProxySelection+` and we gradually add and remove proxies, so as to investigate the effect of changing the number of proxies on $\overline{L}$. In the case where we set $l^{(h)} = l^{(\text{mean})}$, the DML ensures that $\overline{L}$ will not surpass $L_{\max}$, by assigning 4 proxies in selected positions. If we further decrease the number of proxies, we have that $\overline{L} > L_{\max}$, and the latency constraint is not met. At the leftmost point of the plot, we can see the latency achieved when using only one proxy (the controller C, or in other words, when the DML functionalities are absent), which is much higher than when employing additional proxies. When we replace the value of $l^{(h)}$ with $l^{(h)} = l^{(\max)}$, and we re-run the algorithm, we observe similar behavior in the performance, but in this case with 8 selected proxies.

$\overline{L}$ achieved. Fig. 8b displays the results on the average access latency for the three alternative methods. The yellow bar for the DML method is the $\overline{L}$ value when we consider the worst case of $l^{(h)} = l^{(\max)}$. This is an important point to make, as the Figure shows that, by adapting the number of proxies, DML is able to always guarantee the constraint, irrespective of whether $l^{(h)}$ is formulated as an average of mean latencies, or as an average of worst-case latencies. We can see that the efficient management of proxies provided by the DML results in a better performance compared to the other two alternatives. This fact is explained by the nature of `ProxySelection+`, which receives as input the $L_{\max}$.

Number of proxies used. We compare the three methods with respect to the number of special nodes that they use. The DML is using proxies, the non storing mode is using the controller C and the storing mode is using LCAs. The use of special nodes is wasteful on

resources. For example, the proxies store the data requested and the correspondence of producers and consumers, and the LCAs hold routing information about their descendants. In Fig. 8c, we can see that the DML is performing really well compared to the storing mode and uses less special nodes. Of course, the non storing mode is using just one special node, but this has a severe impact on the latency achieved, as shown in Fig. 8b. Even when the DML uses more proxies to guarantee worst-case latencies, their number if comparable to the case of storing mode. However, the DML drastically reduces the latency in this case, thus achieving a much more efficient use of proxies.

3.4 Large-scale simulations

The testbed environment gives us an important ability to test the methods on real conditions and derive useful indications. However, at the same time it does not allow us to perform larger scale, or variable experiments, easily and fast. For this reason, we developed a simulation model based on the system modeling presented in Section 3.1. The simulation environment we use is Matlab. We verify that the simulations are meaningful via validation, by comparing the results obtained with the simulation model to those of the testbed experiments, and then we extend our performance evaluation through simulations.

3.4.1 Validation of the simulation model and simulation settings

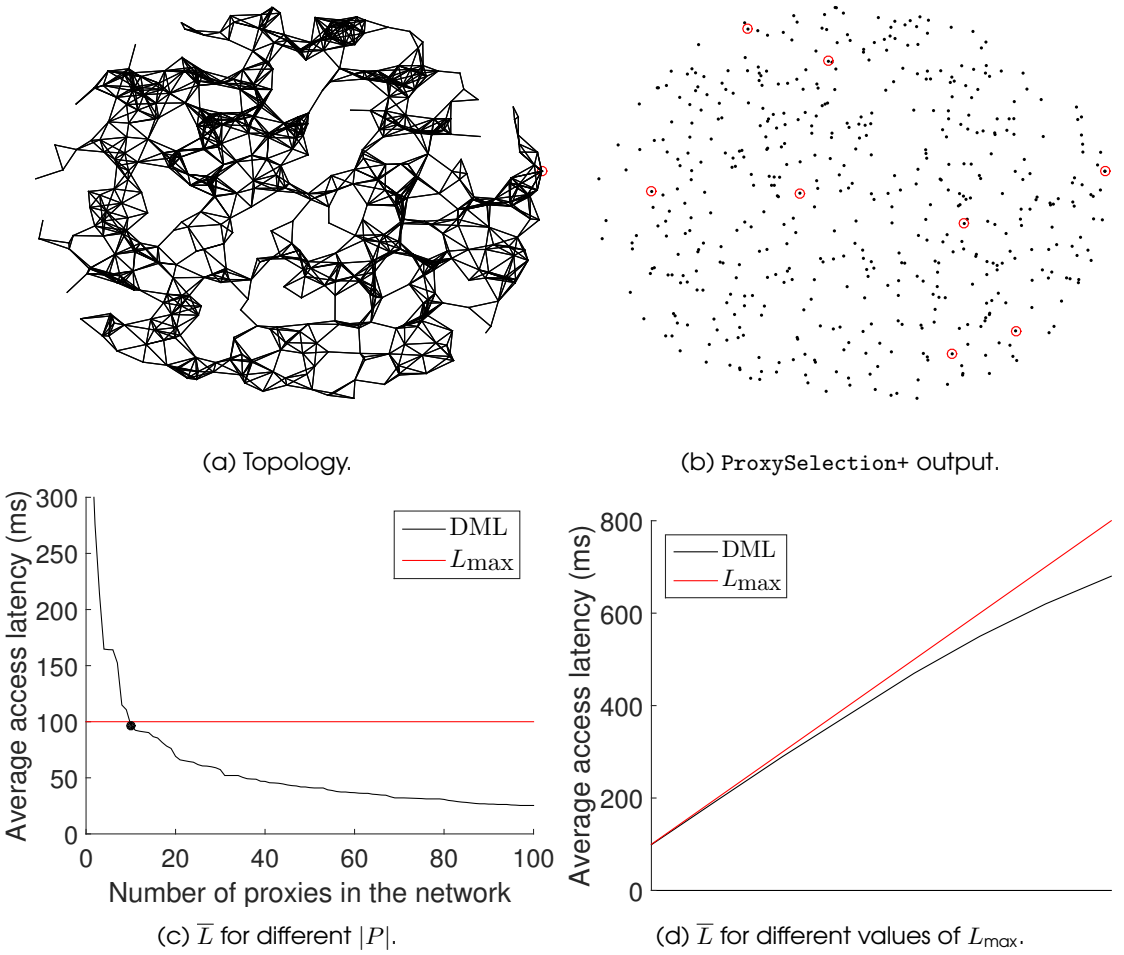
We constructed, in simulation, instances similar to the one that was tested in the Euratech testbed. The results obtained are displayed with green color in Fig. 8. It is clear that the results obtained by the simulation model are very similar to the results obtained during the real experiment, and therefore we can extract reliable conclusions from the simulation environment.

Fig. 9a displays a typical network deployment of 500 nodes, with the corresponding wireless links and with the controller C lying on the far right edge of the network, depicted as a red circle. Fig. 9b displays the locations of the final set P of proxies depicted as red circles after running `ProxySelection+`. The spatial display of Fig. 9b, shows that the final selection results in a balanced proxy selection, ensuring that even isolated nodes, which are located near sparse areas of the network, also have access to a proxy.

In the simulations we focus on showcasing different aspects of the data management and distribution process. We construct larger and different deployments and topologies than the ones of the Euratech testbed, we investigate different values of L_{max}, we consider diverse percentages of requesting nodes and we also measure the energy consumption. The deployment area $\mathcal{A}$ is set to be circular the nodes are deployed uniformly at random. We construct networks of different number of nodes, inserting the additional nodes in the same network area and at the same time decreasing the communication range r_u appropriately, so as to maintain a single strongly connected component at all times. An example of a generated network of 500 nodes is depicted in Fig. 9a. In the following, we present results where $l^{(h)}$ is measured as the mean of latencies. Fig. 9c shows the value of $\overline{L}$ obtained in the case of Fig. 9a, qualitatively confirming the results shown in Fig. 8a.

3.4.2 Simulation results

Different values of L_{max}. We tested the performance of the DML for different values of L_{max}, in networks of 500 nodes, with $m = 0.4 \cdot |S|$. The results are shown in Fig. 9d. The red points represent the values for the maximum latency threshold L_{max} provided by the industrial operator. The average access latency achieved by the DML is always below the threshold, due to the provisioning of `ProxySelection+` Algorithm. In fact, we can see that the more the value of L_{max} is increased, the larger the difference between $\overline{L}$ and



(a) Topology.

(b) `ProxySelection+` output.

(c) $\overline{L}$ for different $|P|$.

(d) $\overline{L}$ for different values of $L_{\max}$.

Figure 9: Network with $n = 500$, $m = 0.4 \cdot |S|$.

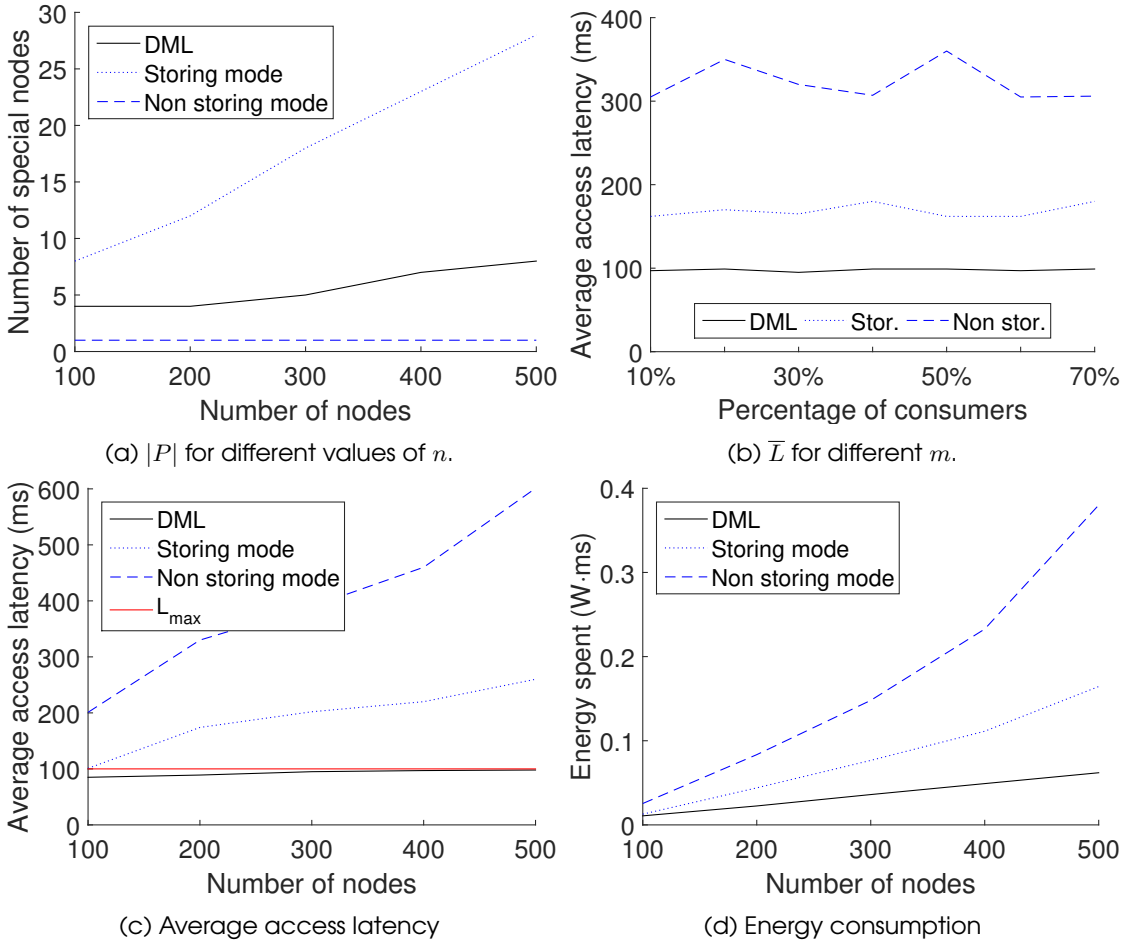
$L_{\max}$ becomes. This happens because for higher $L_{\max}$ values, the latency constraint is more relaxed, and lower $\overline{L}$ can be achieved more easily.

Number of proxies used. We compare the three methods with respect to the number of special nodes that they use. As usual, the DML is using proxies, the non storing mode is using the controller C and the storing mode is using LCAs. As we mentioned earlier, the use of special nodes is wasteful on resources. In Fig. 10a, we can see that the DML is performing really well compared to the storing mode and uses much less special nodes. Of course, the non storing mode is using just one special node for any network size, but this has a severe impact on the latency achieved.

Different percentages of requesting nodes. Another possible factor that could affect the $\overline{L}$ achieved, is the percentage of consumers. In Fig. 10b, we can see that $\overline{L}$ remains constant for any percentage of requesting nodes, in every of the three alternatives. This shows that DML is able to automatically adapt the number of proxies, so as to guarantee the latency constraints irrespective of the number of consumers.

Average latency achieved. Fig. 10c displays the results on the average access latency for the three alternatives, for different numbers of nodes in the network. We can see that the efficient management of proxies provided by the DML results in a better performance compared to the other two alternatives. $\overline{L}$ achieved by the DML respects the latency constraint and always remains lower than $L_{\max}$ (red line).

Energy consumption. Another aspect that we can easily evaluate in simulation, is the energy cost in terms of communication, related to data access. We evaluate this as the cost of transmissions required to serve consumers' requests. In order to obtain the desired



(a) $|P|$ for different values of n.

(b) $\overline{L}$ for different m.

(c) Average access latency

(d) Energy consumption

Figure 10: Comparison of the three methods.

results in units of energy, we transform the dBm units provided in the CC2420 datasheet (36) to mW and we multiply with the time that each node of the network is operational. Fig. 10d displays the energy consumption in the entire network for the three alternatives, for different numbers of nodes. The energy consumption for communication is lower in the case of the DML because low latency comes with less transmissions in the network, resulting in fewer energy demands.

4 Maximum data access latency guarantees

In this section, we exploit the the presence of a limited set of proxy nodes which are more capable than resource-limited IoT devices in the resource-constrained network. Different to the previous section, we focus on network lifetime and on maximum (instead of average) data access latencies. The problem we consider is the maximization of the network lifetime, given the proxy locations in the network, the initial limited energy supplies of the nodes, the data request patterns (and their corresponding parameters), and the maximum latency that consumer nodes can tolerate since the time they request data. We prove that the problem is computationally hard and we design an offline centralized heuristic algorithm for identifying which paths in the network the data should follow and on which proxies they should be cached, in order to meet the latency constraint and to efficiently prolong the network lifetime. We implement the method and evaluate its performance using an FIT IoT-LAB testbed (10), comprised of IEEE 802.15.4-enabled WSN430 network nodes. We demonstrate that the proposed heuristic (i) guarantees data access latency below the given threshold, and (ii) performs well in terms of network lifetime with respect to a theoretically optimal solution.

In Section 4.1, we provide the model of the settings we consider, as well as the necessary notation. In Section 4.2, give a formal definition of the problem we consider, namely the Latency Constrained Edge Data Distribution Problem. In Section 4.3, we provide an efficient heuristic algorithm for addressing the problem, as well as a theoretically optimal solution, the performance of which can serve as an upper bound to the performance of our algorithm in Section 4.4. Finally, in Section 4.5 we present the experimental findings.

4.1 System modeling

We model an industrial IoT network as a graph $G = (V, E)$ where every node $u \in V$ has a limited amount of energy supply E_u. The network features two types of nodes: resource constrained sensor and actuator nodes and a limited number of proxy nodes in a set P, with $P \subset V$, $|P| \ll |V - P|$, and $E_p \gg E_u, \forall u \in V, p \in P$. A node $u \in V$ can propagate data using suitable industrial wireless technologies (e.g., IEEE 802.15.4e) to a set of nodes which lie in its neighborhood N_u. N_u contains the nodes $v \in V$ for which it holds that $\gamma \cdot \rho_u \geq \delta(u, v)$, where ρ_u is the transmission range of node u (defined by the output power of the antenna), $\delta(u, v)$ is the Euclidean distance between u and v, and γ is a neighborhood adjustment parameter, calibrated by the network operator, with $0 < \gamma \leq 1$. Parameter γ is particularly useful in indoor industrial environments where high amounts of interference exist (40), and the nodes might need to transmit messages only to other nodes which are located nearby. The set N_u is thus defining the set of edges E of the graph G. Each one-hop data propagation from u to v results in a latency l_{uv}. We denote as L_{uv} the latency of the multi-hop data propagation from u to v, where $L_{uv} = l_{ui} + l_{ii+1} + ... + l_{i+nv}$. Assuming that all network nodes operate with the same output power, each one-hop data propagation from u to v requires an amount of ϵ_{uv} of energy dissipated by u so as to transmit one data piece to v.

Occasionally, data generation occurs in the network, relevant to the industrial process. The data are modeled as a set of data pieces $D = \{D_i\}$. Each data piece is defined as $D_i = (s_i, c_i, r_i)$, where $s_i \in V$ is the source of data piece D_i, $c_i \in V$ is the consumer of data piece D_i, and r_i is the data generation rate of D_i. If the same data of a source, e.g., s_1, is requested by more than one consumers, e.g., c_1 and c_2, we have two distinct data pieces, $D_1 = (s_1, c_1, r_1)$ and $D_2 = (s_2, c_2, r_2)$, where $s_1 = s_2$. Without loss of generality, we divide time in time cycles τ and we assume that the data may be generated (according to rate r_i) at each source s_i at the beginning of each τ. We assume at the data generation and request patterns are not synchronous, and that therefore data generation need to be cached temporarily for future requests by consumers. This asynchronous data distribution can be implemented through an industrial pub/sub system (41). A critical aspect in the industrial operation is the timely data access by the consumers upon request, and, typically, the data distribution system must guarantee that a given maximum data access

latency constraint (defined by the specific industrial process) is satisfied. We denote this threshold as $L_{\max}$.

We consider that the set P of proxy nodes is strong in terms of computation, storage and energy supplies, and any node $p \in P$ can act as proxy in the network and cache data originated from the sources, for access from the consumers when needed. This relieves the IoT devices from the burden of storing data they generate (which might require excessive local storage), and helps meeting the latency constraint, as discussed in the following. When a data piece D_i is generated at source s_i, it is delivered and stored to a proxy p via a multi-hop wireless path. Upon a request from c_i, it is delivered from p via a (distinct) multi-hop path. We denote as L_{c_i} the data access latency of c_i, with $L_{c_i} = l_{c_i u} + ... + l_{vp} + l_{pv} + ... + l_{uc_i}$. In order to meet the industrial requirements the following constraint must be met:

$$L_{c_i} \leq L_{\max}, \forall c_i \in V \tag{6}$$

Given this constraint, our goal is to identify, for each data source s_i, the proxy p where its data should be cached, in order to maximize the total lifetime of the network. In the following section we formally define this maximization problem.

4.2 The Latency Constrained Edge Data Distribution problem

4.2.1 Computational intractability of the problem

Below we give a formal definition of the problem we consider. More specifically, our problem can be formulated first as a decision problem, i.e., one where we check the existence of a feasible allocation (which can be posed as a yes-no question of the input values), and then as a computation problem, i.e., one where we should find the optimal solution. We then show that even the decision version is computationally intractable; this result gives us leverage on designing efficient heuristics for the computation version of the problem.

Decision problem (LCED). *Suppose that we are given a network $G = (V, E)$, a set of proxies P deployed in the network, the energy supplies E_u and energy consumption costs ϵ_{uv} for every $u, v \in V$, and d data pieces $D_1, D_2, ..., D_d$. The Latency Constrained Edge Data Distribution problem (LCED) is to determine whether there is a feasible multi-hop data propagation schedule such that at least one $p \in P$ stores the data for every $c_i \in V$, the latency constraint $L_{c_i} \leq L_{max}$ is met for every $c_i \in V$, and, at the same time, no node in the network runs out of energy before time T.*

Note that in this version of the problem, we introduce parameter T, which is used for the proof on $\mathcal{NP}$-completeness, and will be omitted in the formulation of the computation version, as the problem will be turned into a maximization of the time until the first node dies, which is a typical measure of network lifetime in the networking literature. We show that the general version of the LCED is $\mathcal{NP}$-complete.

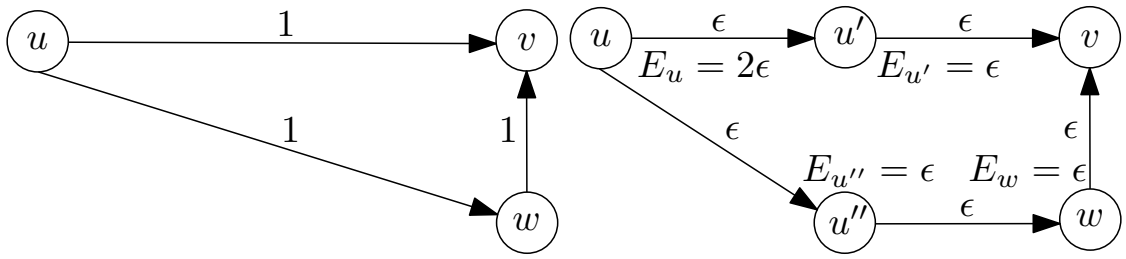
Theorem 1. *LCED is $\mathcal{NP}$-complete.*

Proof. We first note that, given a certain data propagation schedule, we can verify whether this schedule is sufficient so that no node dies, i.e., no node u spends exactly E_u energy for propagating data before time T. In particular, this can be done in $\mathcal{O}(dV)$ time, by summing the energy spent on every node in the network for every data piece generated until time T. Therefore, we have that LCED $\in \mathcal{NP}$.

For the hardness part we use the Directed Two Commodity Integral Flow problem (42, p. 216), D2CIF in short. Let $G(V, A), s_1, s_2, t_1, t_2, c(a) = 1, \forall a \in A, R_1, R_2$ be the input of D2CIF. We now transform this into an input for LCED as follows:

We set the energies needed for the transmission of a data piece $\epsilon_{uv} = \epsilon \cdot c(u, v) \overset{c(a)=1, \forall a \in A}{=} \epsilon$, the latencies required for the one-hop data propagations of a data piece $l_{uv} = l, \forall u, v \in V$ and the initial energy supplies of the nodes in the network $E_u = \sum_{v \in N_u} \epsilon_{uv}, \forall u, v \in V$. Then, we modify G in order to take into account the edge capacities of D2CIF as follows:

For every $u \in V$ with $E_u > \epsilon$, we "break" the edge (u, v) and we insert an additional node u' with $E_{u'} = \epsilon$, as shown in Fig. 11, thus replacing edge (u, v) with two new edges, (u, u') and (u', v), with energies needed for transmission of a data piece $\epsilon_{uu'} = \epsilon_{u'v} = \epsilon$. We set the number of data pieces $d = d_1 + d_2$ with $d_1 = R_1$ data pieces $(s_1, c_1, 1)$ and $d_2 = R_2$ data pieces $(s_2, c_2, 1)$. We set $c_i = p_i = t_i$ and $L_{\text{max}} < \min_{u \in N_{c_i}} l_{c_i u}, \forall i \in \{1, 2\}$. Finally, assuming that the data are generated at the beginning of each time cycle τ, we set $T = \tau$. In order to reassure that the data pieces of cycle τ have been delivered before the start of cycle $\tau + 1$, τ has to be sufficiently long. For this reason we set $\tau = l \cdot |E|$.



(a) D2CIF graph (b) LCED node insertion.

Figure 11: Toy example of a graph transformation in LCED.

Notice that an answer to this instance of the LCED problem would provide an answer also to D2CIF, which means that D2CIF $\leq_m$ LCED. This completes the proof. □

4.2.2 The objective function on the maximum lifetime

We now define the computation version of the problem, by providing the target objective function. Our objective is to maximize the lifetime of the network. We define as lifetime the time elapsed from the beginning of the data distribution until the time that the first node in the network depletes its energy.

Computation problem (C-LCED). *Find a multi-hop data propagation schedule that maximizes the lifetime of the network and ensures that at least one $p \in P$ stores the data for every $c_i \in V$, and that $L_{c_i} \leq L_{max}$ for every $c_i \in V$.*

In order to construct the objective function, we introduce the decision variables, x_{uv}^i which hold the necessary information regarding the transmission of the data pieces across the edges of the graph. More specifically, $x_{uv}^i = 1$ when edge (u, v) is activated for data piece D_i. On the contrary, $x_{uv}^i = 0$ when edge (u, v) is inactive for the transmission of data piece D_i. We denote as $a_{uv} = \sum_{i=1}^{d} r_i x_{uv}^i$ the aggregate data rate of (u, v). Stacking all a_{uv} together, we get $\mathbf{x} = [a_{uv}]$, the data rate vector of node u for every $v \in N_u$. Following this formulation, the lifetime of node $u \in V$ can be defined as:

$$T_u(\mathbf{x}) = \frac{E_u}{\sum_{v \in N_u} \epsilon_{uv} a_{uv}}. \tag{7}$$

The network lifetime and objective function of the problem is thus defined as:

$$T(\mathbf{x}) = \min_{u \in V} \left\{ T_u(\mathbf{x}) \mid \sum_{v \in N_u} x_{uv}^i > 0 \right\} \tag{8}$$

4.3 An offline centralized heuristic

In this section, we provide an algorithm (Algorithm 2) for computing the values of x_{uv}^i variables, which in turn are used to compute the value of the objective function (i.e., the lifetime of the network), given the set of proxies in the network, the set of data pieces,

Algorithm 2: `PriorityDataDistribution`

Input : $G, P, D, E_u, \epsilon_{uv}, L_{\text{max}}$

1 $l^{(h)} \leftarrow$ assign value through initialization phase at the industrial installation

2 $\Omega'_{pc_i} \leftarrow \forall c_i \in V$ compute Ω_{pc_i}, using Eppstein's algorithm (43) and cut paths with $|\Omega^{(j)}_{pc_i}| \cdot l^{(h)} > L_{\text{max}}$

3 $D' \leftarrow$ sort D from highest to lowest r_i

4 **for** $i = D'_1 : D'_d$ **do**

5 $\quad P' \leftarrow P$ with $\Omega'^{(j)}_{pc_i} \neq \varnothing$

6 $\quad$ **for** $\forall p \in P'$ **do**

7 $\quad\quad \Pi_{s_i p} \leftarrow$ lifetime weighted path $s_i \rightarrow p$

8 $\quad\quad \Pi_{pc_i} \leftarrow$ lifetime weighted path $p \rightarrow c_i$

9 $\quad T^i_{\text{max}} = \max_{p \in P'} \min_{u \in \Pi_{s_i p} \cup \Pi_{pc_i}} T^i_u(\mathbf{x})$

10 $\quad \Pi_i \leftarrow \Pi_{s_i p} \cup \Pi_{pc_i}$ which achieves $T^i_{\text{max}}(\mathbf{x})$

11 $\quad x^i_{uv} = 1, \forall u, v \in \Pi_i$

Output: x^i_{uv}

the initial energy supplies of the nodes and the data access latency constraint L_{max}. The algorithm is called `PriorityDataDistribution` (PDD in short) due to the fact that it starts serving the data pieces with respect to their generation rate. The intuition behind this prioritization is that the higher the generation rate of a data piece, the more energy it will dissipate in the network. Consequently, serving at first the more demanding data pieces will ensure the allocation of nodes with long remaining lifetime to them.

The overall intuition behind the heuristic is as follows. For each data piece, we compute possible paths that would meet the maximum latency constraint. For each such path, we compute the energy drain on the nodes in the path. Therefore, we identify the node in the network with the minimum residual lifetime, if that path is activated. Among all feasible paths, we finally pick the one resulting in the maximum residual lifetime. Intuitively, the algorithm progressively balances energy consumption across nodes, thus maximizing the time until the first node dies.

In general, the values of L_{c_i} are unknown a priori, and therefore we assume that measurements of the edge latencies are available, and we take the maximum latency as (worst-case) representative. For this reason, at the first step of the algorithm, we initialize $l^{(h)}$ through preliminary measurements. This happens through an initialization phase (line 1 in Algorithm 2) during which the network designer measures different single-hop data transmission latencies within the industrial installation and gathers a sufficiently representative dataset of l_{uv} measurements from different pairs of nodes $u, v \in V$. By using the highest measured value, we obtain a base value $l^{(h)}$ on the worst-case one-hop latency (one could also store the worst-case latency on each link, and use them to compute the expected latency on the paths, improving the accuracy of the heuristic without changing the computational complexity of the algorithm).

In order to address the latency constraint, we define for every pair of c_i and p a set of paths Ω_{pc_i}. Specifically, Ω_{pc_i} is a set of k paths $\Omega^{(1)}_{pc_i}, ..., \Omega^{(k)}_{pc_i}$, where $\Omega^{(j)}_{pc_i}$ is the set of edges (u, v) of path j:

$$\Omega_{pc_i} = \left\{ \Omega^{(1)}_{pc_i}, ..., \Omega^{(k)}_{pc_i} \right\}. \tag{9}$$

The sets Ω_{pc_i} can be known during the network deployment or alternatively, they can be approximately computed by using known methods (e.g., Eppstein's algorithm (43), in $\mathcal{O}(E + V \log V + k)$ time). If the optimal paths from the consumers to the proxies which achieve $L_{c_i} \leq L_{\text{max}}$ are not given, the algorithm computes an approximate set of k such paths (line 2).

The algorithm then sorts the data pieces according to their generation rate (line 3) and iterates for each data piece D_i the following process: It considers the subset of proxies which can serve c_i respecting the data access latency constraint (line 5). For

every proxy node p, the algorithm computes the shortest weighted paths from s_i to p and from p to c_i (lines 7-8). We consider as weight of node u the remaining lifetime $T_u^{i-1}(\mathbf{x}) - T_u^i(\mathbf{x})$ during the consideration of data piece D_i, with $a_{uv} = \sum_1^i r_i x_{uv}^i$. Then, the algorithm chooses the path Π_i which achieves the longest lifetime T_{max}^i (lines 9-10). Specifically, for each path we compute the node that will die first in the network if that path is activated, and we take the path resulting in the longest time until the first node dies. The edges of the nodes of this path are then activated for data piece D_i by setting $x_{uv}^i = 1$ (line 11) and the algorithm moves to the next data piece. It can be shown that PDD requires $\mathcal{O}(DPV^2 + E + k)$ time in the worst case.

4.4 Benchmarking with a theoretically optimal solution

We additionally provide a formulation of a relaxed version of the problem. We use the performance of the solution of this formulation as a benchmark, in order to have a performance upper bound for PDD. At first, we formulate the problem as an integer program, without considering the latency constraint:

$$\text{max.:} \quad T(\mathbf{x}) \tag{10}$$

$$\text{subj. to:} \quad \sum_{v \in V} (x_{uv}^i - x_{vu}^i) = 0 \qquad \forall u \in V \setminus \{s_i, c_i\} \tag{11}$$

$$\sum_{v \in V} (x_{s_i v}^i - x_{v s_i}^i) = 1 \qquad \forall s_i \in V \tag{12}$$

$$\sum_{v \in V} (x_{c_i v}^i - x_{v c_i}^i) = -1 \qquad \forall c_i \in V \tag{13}$$

$$\sum_{v \in V} \sum_{i} \epsilon_{uv} r_i x_{uv}^i \leq E_u \qquad \forall u \in V \tag{14}$$

$$\sum_{v \in V} x_{uv}^i \leq 1 \qquad \forall u \in V, \forall i \tag{15}$$

$$x_{uv}^i, \in \{0, 1\} \qquad \forall u, v \in V, \forall i \tag{16}$$

The objective function (10) maximizes the time until the first node in the network depletes its energy. Constraints (11-13) guarantee the data flow conservation for all nodes. Constraints (14) guarantee that the total energy consumption of every node u will not exceed the initial energy value E_u. Constraints (15) guarantee that each data piece is propagated from u through one and only one edge (u, v). The variables x_{uv}^i (16) are set to be integers, according to the problem formulation. Note that if we want to ensure that at least one proxy is included at the distribution of the data pieces, we can impose an additional set of constraints which ensure that $\sum_{p \in P} x_{pv}^i \geq 1, \forall i$.

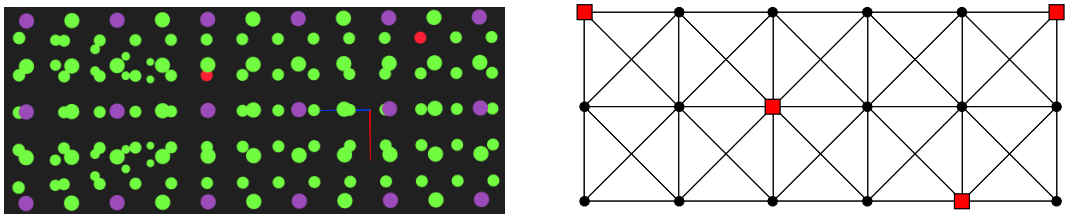
In order to be able to provide a relevant solution for this problem formulation, we consider the linear programming relaxation by replacing the constraints (16) by the weaker constraints $x_{uv}^i, \in [0, 1]$. This relaxation leads to a linear multi-commodity flow problem with fractional flows, which in our case means that an arbitrary fraction of the data piece is sent through edge (u, v) and thus a data piece can be broken in smaller pieces of any fractional size, and sent across multiple links. This does not allow us to evaluate its performance in real settings, but only in simulations, however, in all cases, the solution quality of the linear program is at least as good as that of the integer program, because any integer program solution would also be a valid linear program solution.

4.5 Experimental evaluation

For the experimental implementation and evaluation, we used the Euratech testbed of FIT IoT-LAB. For acquiring the results of the linear program and the performance upper

Table 2: Experimental parameters.

Parameter	Value
Topology	
deployment dimensions (2D grid)	2.4 m $\times$ 6.0 m
number of nodes $\|V\|$, edges $\|E\|$, proxies $\|P\|$	$18, 47, 4$
node distances $\delta(u, v)$	$1.2 - 1.7$ m
transmission range ρ_u, parameter γ	3 m, 0.6
neighborhood N_u	v, with $d(u, v) \leq 2$ m
Hardware	
MCU (ultra low-power)	MSP430
antenna (IEEE 802.15.4)	CC2420
max. battery capacity	830 mAh, 3.7 V
node energies E_u, proxy energies E_p	$0 - 1, 3$ Wh
transmission power	-25 dBm
Time	
time cycle τ	1 sec
latency threshold $L_{\max}$	120 ms
max. measured one-hop latency $l^{(h)}$	28 ms
experiment duration	20 min
Data	
percentage of consumers c_i	$0.05 - 45\%$
data piece generation rate r_i	$1 - 8$ D_i/sec
data piece size (incl. headers and CRC)	9 bytes



(a) Nodes reservation. (b) Network topology.

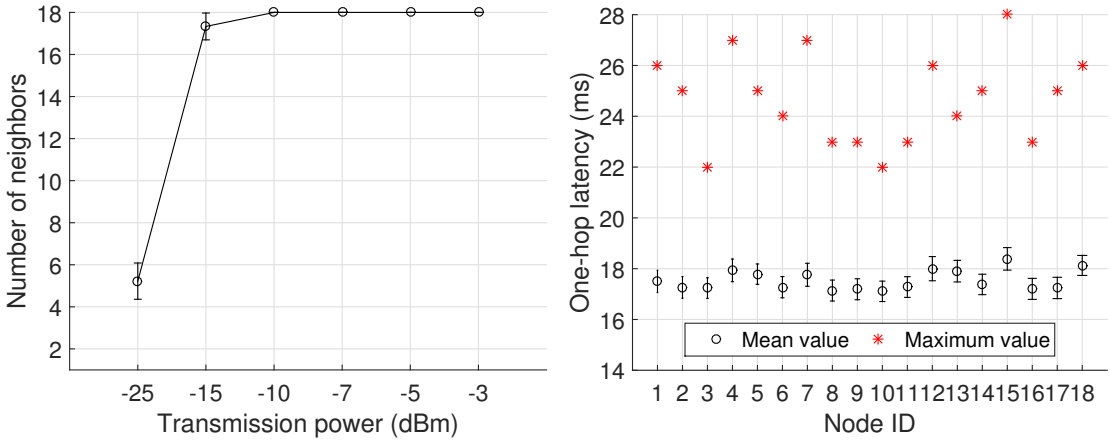
Figure 12: Experimental setup.

bound, we used the Matlab `linprog` solver. The details of the experimental setup are exposed in Table 2.

4.5.1 Experimental setup and parameters

Topology. For the purpose of this study, and in order to acquire a realistic indoor industrial topology we reserved 18 of the testded's nodes (purple dots in Fig. 12a), a selection which results in node distances of $1.2 - 1.7$ m. After having investigated various output power levels (Fig. 13a), we targeted a transmission power of 3 m, with $\gamma = 0.6$, which in turn results to a neighborhood with 5 neighboring nodes on average, where $v \in N_u$ when $d(u, v) \leq 2$. We selected 4 nodes to act as proxies in the network. Given this configuration, we obtain a topology which is depicted in Fig. 12b.

Hardware. For the experiments we use the WSN430 open nodes, the design of which is displayed in Fig. 6b. The WSN430 open node is a mote based on a low power MSP430-based platform, with a set of standard sensors and IEEE 802.15.4 radio interface at 2.4 GHz, using a CC2420 antenna (36). We configured the antenna TX power at -25 dBm and, according to the CC2420 antenna datasheet (36), we acquire the preferred range $\rho_u = 3$ m. The nodes are battery operated with maximum capacity of 830 mAh at 3.7 V. We equip the nodes with $0 - 1$ Wh and the proxies with 3 Wh of energy.



(a) Average size of N_u for different TX power levels and $\gamma = 0.6$.

(b) Mean and maximum one-hop latency measurements.

Figure 13: Initialization measurements.

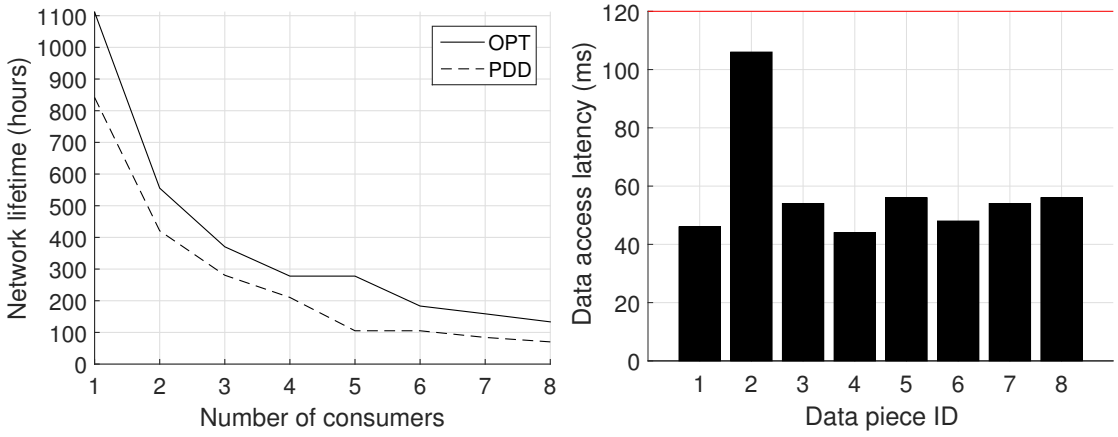
Time. In order to perform the experiments in the most realistic way, we align the L_{max} value with the official requirements of future network-based communications for Industry 4.0, for the targeted industrial applications. Both the *WG1 of Plattform Industrie 4.0* (reference architectures, standards and norms) (38) and the *Expert Committee 7.2 of ITG* (radio systems) (39) set the latency requirements for condition monitoring applications to around 100 ms, so we set the data access latency threshold to $L_{\text{max}} = 120$ ms. We set $\tau = 1$ sec and we assigned $l^{(h)} = 28$ ms after the initialization phase, as explained in the next subsection. Although the experiments duration was set to 20 minutes (which is a very short period to demonstrate the industrial operation effects on the network lifetime) due to resource sharing restrictions with other users of the FIT IoT-LAB platform, we extended our results to longer periods of time, based on the measurements we obtained.

Data. We set the percentage of data sources and consumers to be between $0.05 - 45\%$ on the network nodes number, with a data piece generation rate of $r_i = 1 - 8\ D_i$/sec. The data piece size was set to 9 bytes.

4.5.2 Experimental results

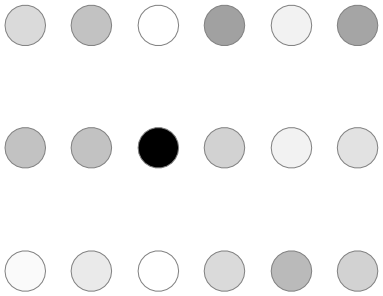
Initialization phase. We ran the initialization phase of PDD, so as to assign values to $l^{(h)}$, by measuring times that are needed for propagating a data piece. In order to obtain reliable results, we repeated the propagation measurements multiple times for all 18 nodes, for different pairs of transmitting and receiving nodes of Euratech testbed. We concluded to the measurements that are shown in the Fig. 13b (mean and maximum values), after measuring the relevant latencies using WSN430 with CC2420 and TinyOS. We can see that the latency values of data propagation from one node to another significantly vary. While the mean latencies are $17 - 19$ ms, the maximum propagation latencies observed are $22 - 28$ ms. In order to provide maximum latency guarantees with respect to L_{max}, we assign to $l^{(h)}$ the maximum value observed in the experiments, $l^{(h)} = 28$ ms.

Network lifetime. We ran the PDD defined data distribution in the network for increasing number of consumers ($c_i = 1 - 8$) and for increasing data piece generation rates ($r_i = 1 - 8\ D_i$/sec), and we compared the results to the performance of the theoretically optimal solution. The comparison is shown in Fig. 14a. We can see that PDD sufficiently approximates the performance of the optimal solution with a difference of $70 - 300$ hours in terms of network lifetime. Considering the many relaxations that the optimal solution assumes (see Section 4.4), this is a remarkably good result (the optimal solution is computed over much milder constraints, and therefore it is more a computable benchmark than an optimal solution).



(a) Network lifetime.

(b) PDD data access latency.



(c) PDD energy balance, $d = 8$.

Figure 14: Experimental results.

Data access latency. Fig. 14b depicts the data access latency achieved by each one of the consumers in network. We measured the latencies asynchronously, by individual requests of the consumers to the corresponding proxies which cached their data. We can see that the latency values lie below the data access latency threshold (red line in Fig. 14b). Note that the consumer requesting data piece D_2 has a longer access time because its position in the network is farther from a proxy, compared to the positions of the rest of the nodes.

Energy balance. For the PDD case, we measured the energy dissipated on each individual node, in order to get an indication about the energy consumption balance over the network. More specifically, we present graphically the spatial evolution of energy consumption in the network. Nodes with high energy consumption are depicted with dark colors. In contrast, nodes with low energy consumption are depicted with bright colors. As exposed in Fig. 14c, PDD achieves a balanced energy consumption over the network, frequently moving data through the nodes with high energy supplies. The reason why the darker node is one in the center is because, as shown in Fig. 12b, it is a central proxy node with 8 edges which serves a lot of requests, as well as lies on numerous paths when $d = 8$.

5 Dynamic path reconfigurations

As in the previous Sections, we assume that applications require a certain upper bound on the data delivery delay from proxies to consumers, and that, at some point in time, a central controller computes an optimal set of multi-hop paths from producers to proxies, and from proxies to consumers, which guarantee a maximum delivery delay, while maximizing the energy lifetime of the network (i.e., the time until the first node in the network exhaust energy resources). In this Section, we focus on maintaining the network configuration in a way such that application requirements are met after important network operational parameters change due to some unplanned events (e.g., heavy interference, excessive energy consumption), while guaranteeing an appropriate utilization of energy resources. We provide several efficient algorithmic functions which locally reconfigure the paths of the data distribution process, when a communication link or a network node fails. The functions regulate how the local path reconfiguration should be implemented and how a node can join a new path or modify an already existing path, ensuring that there will be no loops. The proposed method can be implemented on top of existing data forwarding schemes designed for industrial IoT networks. We demonstrate through simulations the performance gains of our method in terms of energy consumption and data delivery success rate.

In Section 5.1, we provide the model of the settings we consider, as well as the necessary notation. In Section 5.2, we give a formal definition of the network epoch, a lifetime-based metric that will help in the algorithmic design. In Section 5.3, we provide an efficient, distributed data forwarding and path reconfiguration method by presenting some of its core functions. Finally, in Section 5.4 we conduct a performance evaluation based on simulations.

5.1 System modeling

We model an industrial IoT network as a graph $G = (V, E)$. Typically, the network features three types of devices (1): resource constrained sensor and actuator nodes $u \in V$, a central network controller C, and a set of proxy nodes in a set P, with $P \subset V$, $|P| \ll |V - P|$. Every node $u \in V$, at time t, has an available amount of finite energy E_u^t. In general, normal nodes u have limited amounts of initial energy supplies E_u^0, and proxy nodes have significantly higher amounts of initial energy supplies E_p^0, with $E_p^0 \gg E_u^0, \forall u \in V, p \in P$.

A node $u \in V$ can achieve one-hop data propagation using suitable industrial wireless technologies (e.g., IEEE 802.15.4e) to a set of nodes which lie in its neighborhood N_u. N_u contains the nodes $v \in V$ for which it holds that $\rho_u \geq \delta(u, v)$, where ρ_u is the transmission range of node u (defined by the output power of the antenna) and $\delta(u, v)$ is the Euclidean distance between u and v. The sets N_u are thus defining the set of edges E of the graph G. Each one-hop data propagation from u to v results in a latency l_{uv}. Assuming that all network nodes operate with the same output power, each one-hop data propagation from u to v requires and amount of ϵ_{uv} of energy dissipated by u so as to transmit one data piece to v. A node can also transmit control messages to the network controller C by consuming ϵ_{cc} amount of energy. For this kind of transmissions, we assume that more expensive wireless technology is needed, and thus we have that $\epsilon_{cc} \gg \epsilon_{uv}$ (for example, the former can occur over WiFi or LTE links, while the latter over 802.15.4 links).

Occasionally, data generation occurs in the network, relevant to the industrial process. The data are modeled as a set of data pieces $D = \{D_i\}$. Each data piece is defined as $D_i = (s_i, c_i, r_i)$, where $s_i \in V$ is the source of data piece D_i, $c_i \in V$ is the consumer[7] of data piece D_i, and r_i is the data generation rate of D_i. Each data piece D_i is circulated in the network through a multi-hop path $\Pi_{s_i c_i}$. Each node $u \in \Pi_{s_i c_i}$ knows which is the previous node $previous(i, u) \in \Pi_{s_i c_i}$ and the next node $next(i, u) \in \Pi_{s_i c_i}$ in the path of data piece D_i. Without loss of generality, we divide time in time cycles τ and we assume that

[7] If the same data of a source, e.g., s_1, is requested by more than one consumers, e.g., c_1 and c_2, we have two distinct data pieces, $D_1 = (s_1, c_1, r_1)$ and $D_2 = (s_2, c_2, r_2)$, where $s_1 = s_2$.

the data may be generated (according to rate r_i) at each source s_i at the beginning of each τ, and circulated during τ. The data generation and request patterns are not necessarily synchronous, and therefore, the data pieces need to be cached temporarily for future requests by consumers. This asynchronous data distribution is usually implemented through an industrial pub/sub system (41). A critical aspect in the industrial operation is the timely data access by the consumers upon request, and, typically, the data distribution system must guarantee that a given maximum data access latency constraint (defined by the specific industrial process) is satisfied. We denote this threshold as $L_{\max}$.

Due to the fact that the set P of proxy nodes is strong in terms of computation, storage and energy supplies, nodes $p \in P$ can act as proxy in the network and cache data originated from the sources, for access from the consumers when needed. This relieves the IoT devices from the burden of storing data they generate (which might require excessive local storage), and helps meeting the latency constraint. We denote as L_{uv} the latency of the multi-hop data propagation of the path Π_{uv}, where $L_{uv} = l_{ui} + l_{i(i+1)} + ... + l_{(i+n)v}$. Upon a request from c_i, data piece D_i can be delivered from p via a (distinct) multi-hop path. We denote as L_{c_i} the data access latency of c_i, with $L_{c_i} = L_{c_ip} + L_{pc_i}$. We assume an existing mechanism of initial centrally computed configuration of the data forwarding paths in the network, e.g., as presented in (44). In order to meet the industrial requirements the following constraint must be met: $L_{c_i} \leq L_{\max}, \forall c_i \in V$.

5.2 Network epochs and their maximum duration

In order to better formulate the data forwarding process through a lifetime-based metric, we define the network epoch. A network epoch j is characterised by the time J (τ divides J) elapsed between two consecutive, significant changes in the main network operational parameters. A characteristic example of such change is a sharp increase of ϵ_{uv} between two consecutive time cycles, due to sudden, increased interference on node u, which in turn leads to increased retransmissions on edge (u, v) and thus higher energy consumption. In other words, $\frac{\epsilon_{uv}(\tau)-\epsilon_{uv}(\tau-1)}{\epsilon_{uv}(\tau)} > \gamma$, where γ is a predefined threshold. During a network epoch, (all or some of) the nodes initially take part in a configuration phase (central or distributed), during which they acquire the plan for the data distribution process by spending an amount of e_u^{cfg} energy for communication. Then, they run the data distribution process. A network epoch is thus comprised of two phases: *Configuration phase.* During this initial phase, the nodes acquire the set of neighbors from/to which they must receive/forward data pieces in the next epoch. *Data forwarding phase.* During this phase the data pieces are circulated in the network according to the underlying network directives.

Network epochs are just an abstraction that is useful for the design and presentation of the algorithmic functions, but does not need global synchronization. As it will be clear later on, each node locally identifies the condition for which an epoch is finished from its perspective, and acts accordingly. Different nodes "see" in general different epochs. Although some events which affect the epoch duration cannot be predicted and thus controlled, we are interested in the events which could be affected by the data distribution process and which could potentially influence the maximum epoch duration. We observe that an epoch cannot last longer than the time that the next node in the network dies. Consequently, if we manage to maximize the time until a node dies due to energy consumption, we also make a step forward for the maximization of the epoch duration.

We now define the maximum epoch duration, as it can serve as a useful metric for the decision making process of the distributed path reconfiguration. The maximum epoch duration is the time interval between two consecutive node deaths in the network. Specifically, each epoch's duration is bounded by the lifetime of the node with the shortest lifetime in the network, given a specific data forwarding configuration. Without loss of generality, we assume that the duration of the configuration phase equals τ. We define the variables, x_{uv}^{ij} which hold the necessary information regarding the transmission

of the data pieces across the edges of the graph. More specifically, for epoch j, $x_{uv}^{ij} = 1$ when edge (u, v) is activated for data piece D_i. On the contrary, $x_{uv}^{ij} = 0$ when edge (u, v) is inactive for the transmission of data piece D_i. We denote as $a_{uv}^j = \sum_{i=1}^{d} r_i x_{uv}^{ij}$ the aggregate data rate of (u, v) for epoch j. Stacking all a_{uv}^j together, we get $\mathbf{x}_u^j = [a_{uv}^j]$, the data rate vector of node u for every $v \in N_u$. Following this formulation (and if we assumed that $J \to \infty$) the maximum lifetime of u during epoch j can be defined as:

$$T_u(\mathbf{x}_u^j) = \begin{cases} \frac{E_u^j}{\sum_{v \in N_u} \epsilon_{uv} a_{uv}^j} & \text{if } E_u^j > e_u^{\text{cfg}} \\ \tau & \text{if } E_u^j \leq e_u^{\text{cfg}} \\ 0 & \text{if } E_u^j = 0 \end{cases} \tag{17}$$

where e_u^{cfg} is the amount of energy that is needed by u in order to complete the configuration phase. Consequently, given an epoch j, the maximum epoch duration is $J_{\max} = \min_{u \in V} \left\{ T_u(\mathbf{x}_u^j) \mid \sum_{v \in N_u} x_{uv}^{ij} > 0 \right\}$.

In Section 4 we presented methods which can identify, for each data source s_i, the proxy p where its data should be cached, in order to maximize the total lifetime of the network until the first node dies (or, in other words, maximize the duration of the first epoch: $\max \min_{u \in V} \left\{ T_u(\mathbf{x}_u^1) \mid \sum_{v \in N_u} x_{uv}^{i1} > 0 \right\}$), and configure the data forwarding paths accordingly. Reconfigurations can be triggered also when the conditions under which a configuration has been computed, change. Therefore, (i) epoch duration can be shorter than J, and (ii) we do not need any centralized synchronization in order to define the actual epoch duration. We consider the epoch as only an abstraction (but not a working parameter for the functions), which is defined as the time between two consecutive reconfigurations of the network, following the functions presented in Section 5.3.

5.3 Path reconfiguration and data forwarding

The main idea behind our method is the following: the nodes are initially provided with a centralized data forwarding plan. When a significant change in the network occurs, the nodes involved are locally adjusting the paths, using lightweight communication links among them (e.g., 802.15.4) instead of communicating with the central network controller (e.g., LTE, WiFi). The main metric used for the path adjustment is the epoch-related $T_u(\mathbf{x}_u^j)$, as defined in Eq. 17. The functions' pseudocode is presented in the following subsections. The functions are presented in upright typewriter font and the messages which are being sent and received are presented in italics typewriter font. The arguments in the parentheses of the functions are the necessary information that the functions need in order to compute the desired output. The arguments in the brackets of the messages are the destination nodes of the messages and the arguments in the parentheses of the messages are the information carried by the messages. We assume that a node u complies with the following rules: u knows the positions of every $v \in N_u$, u knows the neighborhood N_v of every node v in its own neighborhood N_u, and u stores only local information or temporarily present data pieces in transit.

Distributed data forwarding. The distributed data forwarding function `DistrDataFwd`(u) pseudocode is being ran on every node u of the network and is provided in the body of Alg. 3. At first, if $E_u^0 > 0$, the node communicates its status to the central network controller (which uses the method presented in Section 4 for computing the data distribution parameters (proxy allocation, data forwarding paths) in an initial setup phase of the network), it receives the data forwarding plan and it initiates the first time cycle (lines 1-4). Then, for every time cycle u repeats the following process, until either it is almost dead, or more than half of its associated wireless links spend more energy compared to the previous time cycle, according to the system parameter γ (lines 5-18): u starts the data forwarding process according to the data distribution plan received by C (line 6). Afterwards, it checks if a set of control messages have been received from any $v \in N_u$ and acts accordingly, by calling the necessary functions (lines 7-16).

Algorithm 3: `DistrDataFwd`(u)

1 **if** $E_u^0 > 0$ **then**
2 send $\textit{status}(C)(E_u, \epsilon_{uv}, l_{uv})$
3 receive $\textit{plan}(u)$
4 $\tau = 1$
5 **repeat**
6 run `DataForwarding`(τ)
7 **if** $\exists(u,v)$ *with* $\frac{\epsilon_{uv}(\tau)-\epsilon_{uv}(\tau-1)}{\epsilon_{uv}(\tau)} > \gamma$ **then**
8 `Deactivate`$(i,(u,v)), \forall D_i$
9 send $\textit{alert}(previous(i,u))(u,v), \forall D_i$
10 **if** *receive* $\textit{alert}(u)(v, next(i,v))$ **then**
11 `Deactivate`$(i,(u,v))$
12 call `LocalPathConfig`$(i,u,next(i,v))$
13 **if** *receive* $\textit{join}(u)(i,w,v)$ **then**
14 call `JoinPath`(i,w,v)
15 **if** *receive* $\textit{modify\_path}[u](i,w,deleteArg,dirArg)$ **then**
16 call `ModifyPath`$(i,w,deleteArg,dirArg)$
17 $\tau++$
18 **until** $E_u = 0$ *or* $\frac{\epsilon_{uv}(\tau)-\epsilon_{uv}(\tau-1)}{\epsilon_{uv}(\tau)} > \gamma$ *for* $> 50\%$ *of active edges* (u,v) *of* u
19 send $\textit{alert}(previous(i,u))(u,v), \forall D_i, \forall v \in N_u$
20 `Disconnect`(u)

If u detects that a link is consuming too much energy and has to be deactivated, it deactivates this link (by causing a path disconnection for every D_i that is using this link) and notifies the previous node in the path of every D_i that was using this link, $previous(i,u)$, by sending an alert message (lines 7-9). For a given deactivated link (u,v) for data piece D_i, alert messages contain information about D_i and about the two nodes u, v in the path prior to disconnection. Then, u checks whether there has been an alert message received (line 10), and calls function `LocalPathConfig` (displayed in Alg. 4). Through this function the paths can be reconfigured accordingly, for all involved data pieces D_i. Due to the fact that `LocalPathConfig` sends some additional messages regarding joining a new path and modifying an existing one, u then checks for reception of any of those messages (lines 13 and 15) and calls the necessary functions `JoinPath` and `ModifyPath`.

Finally, u sends an alert message to the previous nodes in the existing paths prior to final disconnection due to energy supplies shortage (line 19).

Local path configuration. A node u calls the path configuration function `LocalPathConfig` when it receives an alert which signifies cease of operation of an edge (u,v) due to a sudden significant increase of energy consumption due to interference $\left(\frac{\epsilon_{uv}(\tau)-\epsilon_{uv}(\tau-1)}{\epsilon_{uv}(\tau)} > \gamma\right)$ or a cease of operation of a node v due to heavy interference in all of v's edges or due to low energy supplies (Alg. 3, lines 9 and 19).

`LocalPathConfig` is inherently local and distributed. The goal of this function is to restore a functional path between nodes u and v by replacing the problematic node $previous(v)$ with a better performing node w, or if w does not exist, with a new efficient multi-hop path Π_{uv}. At first, u checks if there are nodes ι in its neighborhood N_u which can directly connect to v and achieve a similar or better one-hop latency than the old configuration (line 1). If there are, then the w selected is the node ι which given the new data piece, will achieve a maximum lifetime compared to the rest of the possible replacements, i.e., $w = \arg\max_{\iota \in N_u} T_\iota(\mathbf{x}_u^j)$, and an acceptable latency $l_{uw} + l_{wv}$ (line 2). u then sends to w a $\textit{join}$ message (line 3).

If such a node does not exist, then u runs `local_aodv+`, a modified, local version of AODV protocol for route discovery, between nodes u and v. `local_aodv+` is able to add

Algorithm 4: `LocalPathConfig`(i, u, v)

1 **if** $\exists \iota \in N_u$ *with* $v \in N_\iota$ *and* $l_{u\iota} + l_{\iota v} \leq l_{upprevious(i,v)} + l_{previous(i,v)v}$ **then**
2 $\quad w = \arg\max_{\iota \in N_u} T_\iota(\mathbf{x}_u^j)$
3 $\quad$ send $join(w)(i, u, v)$
4 $\quad \Pi_{s_i c_i} \leftarrow$ replace v with w
5 **else**
6 $\quad$ run `local_aodv+`(u, v, TTL)

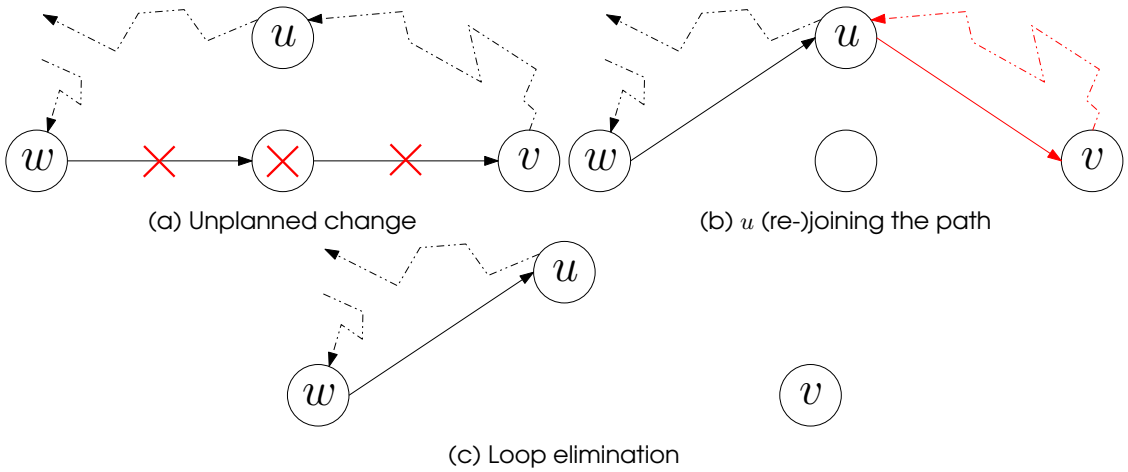
more than one replacement node in the path. The main modification of `local_aodv+` with respect to the traditional AODV protocol is that `local_aodv+` selects the route which provides the maximum lifetime $T_w(\mathbf{x}_u^j)$ for the nodes w which are included in the route. Specifically, this modification with respect to the classic AODV is implemented as follows: The nodes piggyback in the route request messages the minimum lifetime $T_w(\mathbf{x}_u^j)$ that has been identified so far on the specific path. Then when the first route request message arrives at v, instead of setting this path as the new path, v waits for a predefined timeout for more route request messages to arrive. Then, v selects the path which provided the $\max\min_{w \in N_u} T_w(\mathbf{x}_u^j)$. The reader can find more details about the AODV protocol in (45).

Joining new paths, modifying existing paths and avoiding loops. In this subsection, we briefly describe the functions regarding joining a new path and modifying an already existing path for loop elimination. `JoinPath`(i, w, v) is the function which regulates how, for data piece D_i, a node u will join an existing path between nodes w and v and how u will trigger a path modification and avoid potential loops which could result in unnecessary traffic in the network. Due to the fact that the reconfigurations do not use global knowledge, we can have three cases of u joining a path: (i) u is not already included in the path ($u \notin \Pi_{s_i c_i}$), (ii) u is already included in the path ($u \in \Pi_{s_i c_i}$), and w is preceding u in the new path ($previous(i, u) = w$) with a new link (w, u), and (iii) u is already included in the path ($u \in \Pi_{s_i c_i}$), and u is preceding w in the new path ($previous(i, w) = u$) with a new link (u, w). In all three cases, `JoinPath` sends a modification message to the next node to join the path, with the appropriate arguments concerning the deletion of parts of the paths, and the direction of the deletion, for avoidance of potential loops (see (46)). This message triggers the function `ModifyPath` (see (46)). In case (i) it is apparent that there is no danger of loop creation, so there is no argument for deleting parts of the path. In order to better understand cases (ii) and (iii) we provide Figures 15 and 16. In those Figures we can see how the function `ModifyPath` eliminates newly created loops on u from path reconfigurations which follow unplanned network changes.

Following the loop elimination process, loop freedom is guaranteed for the cases where there are available nodes $w \in N_u$ which can directly replace v. In the case where this is not true and `LocalPathConfig` calls `local_aodv+` instead (Alg. 4, line 6), then the loop freedom is guaranteed by the AODV path configuration process, which has been proven to be loop free (47).

5.4 Performance evaluation

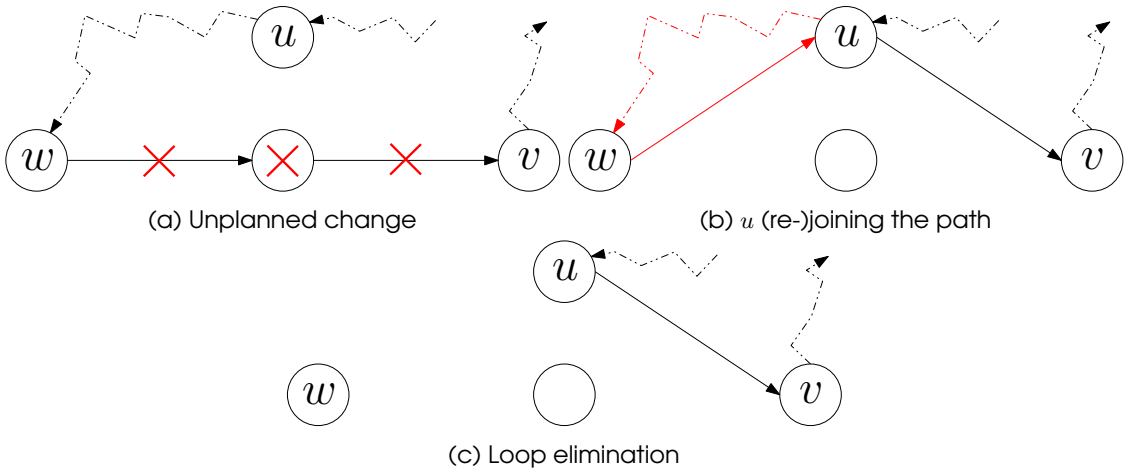
We implemented `DistrDataFwd` method and we conducted simulations in order to demonstrate its performance. We configured the simulation environment (Matlab) according to realistic parameters and assumptions. Table 3 presents the parameter configuration in detail. Briefly, we assume an industrial IoT network, comprised of devices equipped with ultra low-power MCUs like MSP430 and IEEE 802.15.4 antennae like CC2420, able to support industrial IoT standards and protocols like WirelessHART and IEEE 802.15.4e. We assume a structured topology (as in usual controlled industrial settings) of 18 nodes with 4 proxies which form a 2D grid with dimensions of 7.5 m $\times$ 16.0 m. We set the transmission power of the nodes for multi-hop communication to -25 dBm (typical low-power) which results in a transmission range of 3 m. For the more expensive communication with



(a) Unplanned change (b) u (re-)joining the path

(c) Loop elimination

Figure 15: Loop avoidance - forward loop



(a) Unplanned change (b) u (re-)joining the path

(c) Loop elimination

Figure 16: Loop avoidance - backward loop

Table 3: Simulation parameters.

Parameter	Value
Topology	
deployment dimensions (2D grid)	7.5 m $\times$ 16.0 m
$\|V\|$, $\|E\|$, $\|P\|$, $\delta(u,v)$	$18, 47, 4, 2.5 - 2.8$
transmission range ρ_u, neighborhood N_u	3 m, v, with $d(u,v) \leq 3$ m
Time	
τ, L_{max}, TTL (`local_aodv+`)	1 sec, 100 ms, 2 hops
experiment duration	2000 hours
Data	
percentage of consumers c_i	$0.05 - 45\%$
data piece generation rate r_i	$1 - 8\ D_i/\tau$
data piece size (incl. headers and CRC)	9 bytes
Hardware assumptions	
MCU (e.g., MSP430), antenna (e.g., CC2420)	ultra low-power, IEEE 802.15.4
max. battery capacity, initial energies E_u^0, E_p^0	830 mAh / 3.7 V, $0 - 1$ and 3 Wh
transmission power for e_{uv}, for e_{cc}	-25 dBm, 15 dBm

the network controller, we set the transmission power to 15 dBm, typical of wireless LAN settings. We set the time cycle $\tau = 1$ second, the percentage of consumers over the population $0.05 - 45\%$ and we produce $1 - 8\ D_i/\tau$ per consumer. In order to perform the simulations in the most realistic way, we align the L_{max} value with the official requirements of future network-based communications for Industry 4.0 (38), and set the latency threshold to $L_{\text{max}} = 100$ ms. We set $\gamma = 50\%$, the TTL argument of `local_aodv+` equal to 2, we assume a maximum battery capacity of 830 mAh (3.7 V) and equip the nodes with energy supplies of $E_u^0 = 0 - 1$ Wh and $E_p^0 = 3$ Wh. Last but not least, in order to have a realistic basis for the values of the one-hop latencies l_{uv} used in the simulations, we aligned the different l_{uv} values to one-hop propagation measurements with real devices, for different pairs of transmitting and receiving nodes. Specifically, we used the measurements provided in Fig. 3b of (44).

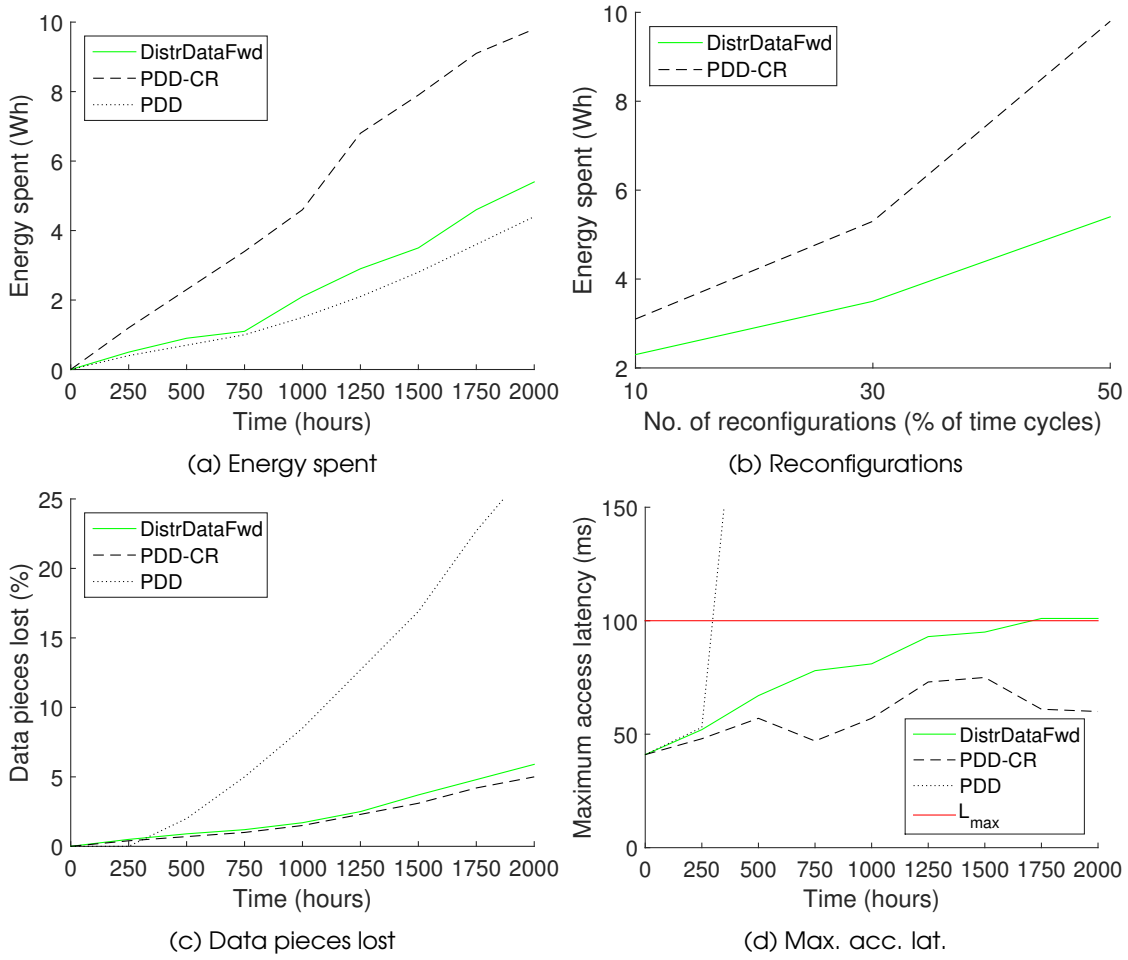
In order to have a benchmark for our method, we compared its performance to the performance of the PDD data forwarding method which was provided in Section 4. Due to the fact that PDD was designed for static environments without significant network parameter changes, we also compare to a modified version of PDD, which incorporates central reconfiguration when needed (we denote this version as PDD-CR). Specifically, PDD-CR runs PDD until time t, when a significant change in the network happens, and then, all network nodes communicate their status (E_u^t, e_{uv}, l_{uv}) to the network controller C by spending e_{cc} amount of energy. C computes centrally a new (near-optimal as shown in Section 4) data forwarding plan and the nodes run the new plan. In our case, we run the PDD-CR reconfigurations for each case where we would do the same if we were running `DistrDataFwd`. As noted before, the conditions that trigger a change of the forwarding paths are either node related (a node dies) or link related (change of interference which results in $\frac{\epsilon_{uv}(\tau) - \epsilon_{uv}(\tau-1)}{\epsilon_{uv}(\tau)} > \gamma$)[8].

Energy efficiency. The energy consumption over the entire network during 2000 hours of operation is depicted in Fig. 17a. The energy consumption values include the energy consumed for both the data distribution process and the reconfiguration. Our method achieves comparable energy consumption as PDD, despite being a local, adaptive method. This is explained by the following facts: PDD-CR requires more energy than `DistrDataFwd` for the path reconfiguration process, as during each epoch alteration every node has to spend e_{cc} amount of energy for the configuration phase. On the contrary, in the `DistrDataFwd` case, only some of the nodes have to participate in a new configuration phase (usually the nodes in the neighborhood of the problematic node), and spend significantly less amounts of energy. In the case of PDD, the nodes do not participate in configuration phases, so they save high amounts of energy. In Fig. 17b, we can also see the energy consumption of `DistrDataFwd` and PDD-CR for different percentages of reconfigurations (w.r.t. the number of time cycles τ). It is clear that the more the reconfigurations that we have in the network, the more the gap between the performance of `DistrDataFwd` and PDD-CR increases.

Data delivery rate. The data pieces lost during 2000 hours of operation are depicted in Fig. 17c. We consider a data piece as lost when the required nodes or path segments are not being available anymore so as to achieve a proper delivery. When a data piece is delivered, but misses the deadline L_{max}, it is not considered as lost, but we measure the high delivery latency instead. We can see that the low energy consumption of the PDD method comes at a high cost: it achieves a significantly lower data delivery rate than the PDD-CR and the `DistrDataFwd` methods. This is natural, because as noted before, PDD computes an initial centralized paths configuration and follows it throughout the entire data distribution process. The performance of the `DistrDataFwd` method stays very close to the performance of the PDD-CR method, which demonstrates the efficiency of `DistrDataFwd` in terms of successfully delivering the data pieces.

Maximum data access latency. The maximum data access latency during 2000 hours of operation is depicted in Fig. 17d. The measured value is the maximum value observed

[8]The qualitative behavior would not change in case of additional reconfiguration events, which simply increase the number of reconfigurations.



(a) Energy spent

(b) Reconfigurations

(c) Data pieces lost

(d) Max. acc. lat.

Figure 17: Performance results.

among the consumers, after asynchronous data requests to the corresponding proxies. `PDD` does not perform well, due to the fact that it is prone to early disconnections without reconfiguration functionality. The fluctuation of `PDD-CR`'s curve is explained by the re-computation from scratch of the data forwarding paths which might result in entirely new data distribution patterns in the network. `DistrDataFwd` respects the $L_{\max}$ threshold for most of the time, however at around 1700 hours of network operation it slightly exceeds it for a single proxy-consumer pair. On the contrary, `PDD-CR` does not exceed the threshold. This performance is explained by the fact that `DistrDataFwd`, although efficient, does not provide any strict guarantee for respecting $L_{\max}$, for all proxy-consumer pairs, mainly due to the absence of global knowledge on the network parameters during the local computations. `PDD-CR`, with the expense of additional energy for communication, is able to centrally compute near optimal paths and consequently achieve the desired latency. There are two simple ways of improving `DistrDataFwd`'s performance in terms of respecting the $L_{\max}$ threshold: (i) insert strict latency checking mechanisms in the `local_aodv+` function, with the risk of not finding appropriate (in terms of latency) path replacements, and thus lowering the data delivery ratio due to disconnected paths, and (ii) increase the TTL argument of `local_aodv+`, with the risk of circulating excessive amounts of route discovery messages, and thus increasing the energy consumption in the network. Including those mechanisms is left for future work.

6 Conclusions and future directions

D2.4 describes the work carried out in Task T2.4 with respect to the proposed Smart Data Management and Distribution mechanisms within AUTOWARE, until month 18 of the project. Solutions which determine the locations which are appropriate to move data to were presented, as well as the novel concept of a distributed Data Management Layer, whereby nodes can cooperate so as to store data within the network. Emphasis was given to the maximization of the network lifetime, the data access latency constraints and the maintenance of the network configuration in a way such that application requirements are met, even after important network operational parameters change due to some unplanned events. Finally, performance results will be also provided, coming from both experiments with real devices and large-scale simulations.

Within Task 2.4, the next steps include further development and improvement of the Smart Data Distribution mechanisms. The main target objective will be the precise adaptation to the application requirements of diverse automation processes. The developments can range from the design of additional heuristics which capture different aspects of the automation processes to the introduction of self-adaptive data management and distribution proposals which also contribute towards the design of processes that are more capable to dynamically reconfigure. Also, we plan to further extend the adaptive path reconfiguration and data forwarding process, so that it takes into account not only "negative" changes in the network, but also "positive" changes, such as addition of nodes in the topology. Finally, we opt for exploring the scalability of the proposed schemes, especially in the cases of larger networks and more dynamic topologies and parameters, which might affect the performance of the designed algorithms.

References

(1) L. D. Xu, W. He, and S. Li, "Internet of things in industries: A survey," *IEEE Transactions on Industrial Informatics*, vol. 10, pp. 2233–2243, Nov 2014.

(2) M. Nobre, I. Silva, and L. Guedes, "Routing and scheduling algorithms for WirelessHART networks: A survey," *Sensors*, vol. 15, p. 9703Ű9740, Apr 2015.

(3) E. Baccelli, M. Philipp, and M. Goyal, "The P2P-RPL routing protocol for IPv6 sensor networks: Testbed experiments," in *SoftCOM 2011, 19th International Conference on Software, Telecommunications and Computer Networks*, pp. 1–6, Sept 2011.

(4) P. Gaj, J. Jasperneite, and M. Felser, "Computer communication within industrial distributed environment - survey," *IEEE Transactions on Industrial Informatics*, vol. 9, pp. 182–189, Feb 2013.

(5) W. Steiner and S. Poledna, "Fog computing as enabler for the industrial internet of things," *e & i Elektrotechnik und Informationstechnik*, vol. 133, pp. 310–314, Nov 2016.

(6) "IEEE standard for local and metropolitan area networks–part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs) amendment 1: MAC sublayer," *IEEE Std 802.15.4e-2012 (Amendment to IEEE Std 802.15.4-2011)*, pp. 1–225, April 2012.

(7) D. Chen, M. Nixon, and A. Mok, *WirelessHART^{TM}*. Springer, 2010.

(8) R. Mahmud, R. Kotagiri, and R. Buyya, *Fog Computing: A Taxonomy, Survey and Future Directions*, pp. 103–130. Singapore: Springer Singapore, 2018.

(9) P. Rodriguez and S. Sibal, "SPREAD: Scalable platform for reliable and efficient automated distribution," *Computer Networks*, vol. 33, no. 1âĂŞ6, pp. 33 – 49, 2000.

(10) C. Adjih, E. Baccelli, E. Fleury, G. Harter, N. Mitton, T. Noel, R. Pissard-Gibollet, F. Saint-Marcel, G. Schreiner, J. Vandaele, and T. Watteyne, "FIT IoT-LAB: A large scale open experimental IoT testbed," in *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, pp. 459–464, Dec 2015.

(11) F. Shrouf, J. Ordieres, and G. Miragliotta, "Smart factories in Industry 4.0: A review of the concept and of energy management approached in production based on the internet of things paradigm," in *2014 IEEE International Conference on Industrial Engineering and Engineering Management*, pp. 697–701, Dec 2014.

(12) T. Watteyne, V. Handziski, X. Vilajosana, S. Duquennoy, O. Hahm, E. Baccelli, and A. Wolisz, "Industrial wireless ip-based cyber-physical systems," *Proceedings of the IEEE*, vol. 104, pp. 1025–1038, May 2016.

(13) D. D. Guglielmo, S. Brienza, and G. Anastasi, "IEEE 802.15.4e: A survey," *Computer Communications*, vol. 88, pp. 1 – 24, 2016.

(14) Y. Chen, R. H. Katz, and J. Kubiatowicz, "SCAN: A dynamic, scalable, and efficient content distribution network," in *Proceedings of the First International Conference on Pervasive Computing*, Pervasive '02, (London, UK, UK), pp. 282–296, Springer-Verlag, 2002.

(15) B. Li, M. J. Golin, G. F. Italiano, X. Deng, and K. Sohraby, "On the optimal placement of web proxies in the internet," in *INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 3, pp. 1282–1290 vol.3, Mar 1999.

(16) H. S. Bassali, K. M. Kamath, R. B. Hosamani, and L. Gao, "Hierarchy-aware algorithms for CDN proxy placement in the internet," *Computer Communications*, vol. 26, no. 3, pp. 251 – 263, 2003. Scalability and Traffic Control in IP Networks.

(17) K.-L. Wu, P. S. Yu, and J. L. Wolf, "Segmentation of multimedia streams for proxy caching," *IEEE Transactions on Multimedia*, vol. 6, pp. 770–780, Oct 2004.

(18) J. Heo, J. Hong, and Y. Cho, "Earq: Energy aware routing for real-time and reliable communication in wireless industrial sensor networks," *IEEE Transactions on Industrial Informatics*, vol. 5, pp. 3–11, Feb 2009.

(19) D. Kim, W. Wang, N. Sohaee, C. Ma, W. Wu, W. Lee, and D.-Z. Du, "Minimum data-latency-bound k-sink placement problem in wireless sensor networks," *IEEE/ACM Trans. Netw.*, vol. 19, pp. 1344–1353, Oct. 2011.

(20) C. Antonopoulos, C. Panagiotou, G. Keramidas, and S. Koubias, "Network driven cache behavior in wireless sensor networks," in *2012 IEEE International Conference on Industrial Technology*, pp. 567–572, March 2012.

(21) C. Panagiotou, C. Antonopoulos, and S. Koubias, "Performance enhancement in wsn through data cache replacement policies," in *Proceedings of 2012 IEEE 17th International Conference on Emerging Technologies Factory Automation (ETFA 2012)*, pp. 1–8, Sept 2012.

(22) A. Saifullah, Y. Xu, C. Lu, and Y. Chen, "End-to-end communication delay analysis in industrial wireless networks," *IEEE Transactions on Computers*, vol. 64, pp. 1361–1374, May 2015.

(23) J. Li, X. Zhu, X. Gao, F. Wu, G. Chen, D. Z. Du, and S. Tang, "A novel approximation for multi-hop connected clustering problem in wireless sensor networks," in *2015 IEEE 35th International Conference on Distributed Computing Systems*, pp. 696–705, June 2015.

(24) M. Ha and D. Kim, "On-demand cache placement protocol for content delivery sensor networks," in *2017 International Conference on Computing, Networking and Communications (ICNC)*, pp. 207–216, Jan 2017.

(25) V. Shah-Mansouri, A. H. Mohsenian-Rad, and V. W. S. Wong, "Lexicographically optimal routing for wireless sensor networks with multiple sinks," *IEEE Transactions on Vehicular Technology*, vol. 58, pp. 1490–1500, March 2009.

(26) V. Shah-Mansouri and V. W. S. Wong, "Bounds for lifetime maximization with multiple sinks in wireless sensor networks," in *2007 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, pp. 82–85, Aug 2007.

(27) L. Mottola and G. P. Picco, "Muster: Adaptive energy-aware multisink routing in wireless sensor networks," *IEEE Transactions on Mobile Computing*, vol. 10, pp. 1694–1709, Dec 2011.

(28) X. Sun and N. Ansari, "Traffic load balancing among brokers at the IoT application layer," *IEEE Transactions on Network and Service Management*, vol. 15, pp. 489–502, March 2018.

(29) S. Han, X. Zhu, A. K. Mok, D. Chen, and M. Nixon, "Reliable and real-time communication in industrial wireless mesh networks," in *2011 17th IEEE Real-Time and Embedded Technology and Applications Symposium*, pp. 3–12, April 2011.

(30) S. Zats, R. Su, T. Watteyne, and K. S. J. Pister, "Scalability of time synchronized wireless sensor networking," in *IECON 2011 - 37th Annual Conference of the IEEE Industrial Electronics Society*, pp. 3011–3016, Nov 2011.

(31) L. Krishnamurthy, R. Adler, P. Buonadonna, J. Chhabra, M. Flanigan, N. Kushalnagar, L. Nachman, and M. Yarvis, "Design and deployment of industrial sensor networks: Experiences from a semiconductor plant and the north sea," in *Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems*, SenSys '05, (New York, NY, USA), pp. 64–75, ACM, 2005.

(32) E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

(33) T. Winter, P. Thubert, A. Brandt, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, J. Vasseur, and R. Alexander, "RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks," RFC 6550, RFC Editor, March 2012.

(34) M. Sepulcre, J. Gozalvez, and B. Coll-Perales, "Multipath qos-driven routing protocol for industrial wireless networks," *Journal of Network and Computer Applications*, vol. 74, pp. 121 – 132, 2016.

(35) P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, "The many faces of publish/subscribe," *ACM Comput. Surv.*, vol. 35, pp. 114–131, June 2003.

(36) "CC2420 datasheet." `http://www.ti.com/lit/ds/symlink/cc2420.pdf`. Accessed: 14-11-2017.

(37) R. Kotian, G. Exarchakos, and A. Liotta, "Data driven transmission power control for wireless sensor networks," in *Proceedings of the 8th International Conference on Internet and Distributed Computing Systems - Volume 9258*, IDCS 2015, pp. 75–87, Springer-Verlag New York, Inc., 2015.

(38) "Network-based communication for Industrie 4.0." Publications of Plattform Industrie 4.0, `www.plattform-i40.de`, 2016. Accessed: 14-11-2017.

(39) A. Müller, "Contribution to the discussion session on "Radio communication for Industrie 4.0"." on May, 28th, 2015 at the ITG-expert comittee 7.2 radio systems, 2015.

(40) A. Willig, K. Matheus, and A. Wolisz, "Wireless technology in industrial networks," *Proceedings of the IEEE*, vol. 93, pp. 1130–1151, June 2005.

(41) T. P. Raptis and A. Passarella, "A distributed data management scheme for industrial IoT environments," in *2017 IEEE 13th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, pp. 196–203, Oct 2017.

(42) M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1979.

(43) D. Eppstein, "Finding the k shortest paths," *SIAM Journal on Computing*, vol. 28, no. 2, pp. 652–673, 1998.

(44) T. P. Raptis, A. Passarella, and M. Conti, "Maximizing industrial IoT network lifetime under latency constraints through edge data distribution," in *1st IEEE International Conference on Industrial Cyber-Physical Systems, (ICPS)*, May 2018.

(45) C. Perkins, E. Belding-Royer, and S. Das, "Ad hoc on-demand distance vector (AODV) routing," *IETF RFC*, July 2003.

(46) T. P. Raptis, A. Passarella, and M. Conti, "Distributed path reconfiguration and data forwarding in industrial IoT networks," *CoRR*, vol. abs/1803.10971, 2018.

(47) M. Zhou, H. Yang, X. Zhang, and J. Wang, "The proof of AODV loop freedom," in *2009 International Conference on Wireless Communications Signal Processing*, pp. 1–5, Nov 2009.