Distributed Path Reconfiguration and Data Forwarding in Industrial IoT Networks

Theofanis P. Raptis(✉), Andrea Passarella, and Marco Conti

Institute of Informatics and Telematics, National Research Council, Pisa, Italy
{theofanis.raptis,andrea.passarella,marco.conti}@iit.cnr.it

Abstract. In today's typical industrial environments, the computation of the data distribution schedules is highly centralised. Typically, a central entity configures the data forwarding paths so as to guarantee low delivery delays between data producers and consumers. However, these requirements might become impossible to meet later on, due to link or node failures, or excessive degradation of their performance. In this paper, we focus on maintaining the network functionality required by the applications after such events. We avoid continuously recomputing the configuration centrally, by designing an energy efficient local and distributed path reconfiguration method. Specifically, given the operational parameters required by the applications, we provide several algorithmic functions which locally reconfigure the data distribution paths, when a communication link or a network node fails. We compare our method through simulations to other state of the art methods and we demonstrate performance gains in terms of energy consumption and data delivery success rate as well as some emerging key insights which can lead to further performance gains.

Keywords: Industry 4.0 · Internet of Things · Data distribution

1 Introduction

With the introduction of Internet of Things (IoT) concepts in industrial application scenarios, industrial automation is undergoing a tremendous change. This is made possible in part by recent advances in technology that allow interconnection on a wider and more fine-grained scale [12]. The core of distributed automation systems and networks is essentially the reliable exchange of data. Any attempt to steer processes independently of continuous human interaction requires, in a very wide sense, the flow of data between some kind of sensors, controllers, and actuators [9].

In today's typical industrial configurations, the computation of the data exchange and distribution schedules is quite primitive and highly centralised. Usually, the generated data are transferred to a central network controller or

K. R. Chowdhury et al. (Eds.): WWIC 2018, LNCS 10866, pp. 29–41, 2018.
https://doi.org/10.1007/978-3-030-02931-9_3

intermediate network proxy nodes using wireless links. The controller analyses the received information and, if needed, reconfigures the network paths, the data forwarding mechanisms, the caching proxy locations and changes the behaviour of the physical environment through actuator devices. Traditional data distribution schemes can be implemented over relevant industrial protocols and standards, like IEC WirelessHART and IEEE 802.15.4e.

Those entirely centralised and offline computations regarding data distribution scheduling, 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 [10]. Also, 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.

Our Contribution. In this paper we consider an industrial IoT network comprised of sensor and actuator nodes. Data consumers (actuators) and producers (sensors) are known. A number of intermediate resource-rich nodes act as proxies. 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 maximising the energy lifetime of the network (i.e., the time until the first node in the network exhaust energy resources). 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 utilisation 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.

2 Related Works

There are numerous relevant previous works in the literature, but due to lack of space, we provide some information about the most representative and most related ones to this paper, which are [2,3,5,6,8,11]. 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 [2]. Efficient proxy-consumer assignments are presented in [5]. Data re-allocation methods among proxies for better traffic balancing are presented in [11]. Scheduling of the data distribution process maximising the time until the first network node dies is suggested in [6], respecting end-to-end data access latency constraints of the order of 100 ms, as imposed by the industrial operators [1]. 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 [8]. Reliable routing, improved communication latency and stable real-time communication, at the cost of modest overhead in device configuration, are demonstrated in [3]. Different to those works, in this paper, we present a method which exploits the local knowledge of the network nodes so as to perform distributed, local path reconfiguration computations towards more efficient energy dissipation across the network.

3 The Model

We model an industrial IoT network as a graph $G = (V, E)$. Typically, the network features three types of devices [13]: 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 neighbourhood 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 modelled 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[1] of data piece D_i, and r_i is the data generation rate of D_i.

[1] 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 $s_1 = s_2$.

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 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 [5]. 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. Proxy selection placement strategies have been studied in recent literature [2,5]. We denote as L_{uv} the latency of the multi-hop data propagation of the path Π_{uv}, where $L_{uv} = l_{ui} + l_{i(i+1)} + \ldots + l_{(i+n)v}$. 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 $L_{c_i} = L_{c_i p} + L_{p c_i}$. We assume an existing mechanism of initial centrally computed configuration of the data forwarding paths in the network, e.g., as presented in [6]. In order to meet the industrial requirements the following constraint must be met: $L_{c_i} \leq L_{\max}, \forall c_i \in V$.

4 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 neighbours 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 synchronisation. 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 maximise the time until a node dies due to energy consumption, we also make a step forward for the maximisation 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:

$$T_u(\mathbf{x}_u^j) = \begin{cases} \frac{E_u^j}{\sum_{v \in N_u} \epsilon_{uv} a_{uv}^j} & \text{if } E_u^j > e_u^{\text{cfg}} \\ \tau & \text{if } E_u^j \leq e_u^{\text{cfg}} \\ 0 & \text{if } E_u^j = 0 \end{cases}, \tag{1}$$

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 $J_{\max} = \min_{u \in V} \left\{ T_u(\mathbf{x}_u^j) \mid \sum_{v \in N_u} x_{uv}^{ij} > 0 \right\}$.

There have already been works in the literature which identify, for each data source s_i, the proxy p where its data should be cached, in order to maximise the total lifetime of the network until the first node dies [6] (or, in other words, maximise 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 centralised synchronisation 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 Sect. 5.

5 Path Reconfiguration and Data Forwarding

The main idea behind our method is the following: the nodes are initially provided with a centralised 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. 1. The functions' pseudocode is presented in the following subsections. Due to lack of space we omit the presentation of some functions' pseudocodes, but those can be found in the extended version of the paper [7]. 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 neighbourhood N_v of every node v in its own neighbourhood 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 Algorithm 1. At first, if $E_u^0 > 0$, the node communicates its status to the central network controller (which uses the method presented in [6] 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 7–15).

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 7–7). 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 Algorithm 2). 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

Algorithm 1. `DistrDataFwd`(u)

```
1  if E_u^0 > 0 then
2      send status[C](E_u, ε_uv, l_uv)
3      receive plan[u]
4      τ = 1
5      repeat
6          run DataForwarding(τ)
7          if ∃(u,v) with (ε_uv(τ) − ε_uv(τ−1)) / ε_uv(τ) > γ then
8              Deactivate(i, (u,v)), ∀D_i
9              send alert[previous(i,u)](u,v), ∀D_i
10         if receive alert[u](v, next(i,v)) then
11             Deactivate(i, (u,v))
12             call LocalPathConfig(i, u, next(i,v))
13         if receive join[u](i, w, v) then
14             call JoinPath(i, w, v)
15         if receive modify_path[u](i, w, deleteArg, dirArg) then
16             call ModifyPath(i, w, deleteArg, dirArg)
17         τ++
18     until E_u = 0 or (ε_uv(τ) − ε_uv(τ−1)) / ε_uv(τ) > γ for > 50% of active edges (u,v) of u
19 send alert[previous(i,u)](u,v), ∀D_i, ∀v ∈ N_u
20 Disconnect(u)
```

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 (Algorithm 1, lines 7 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 neighbourhood 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 *join* message (line 3).

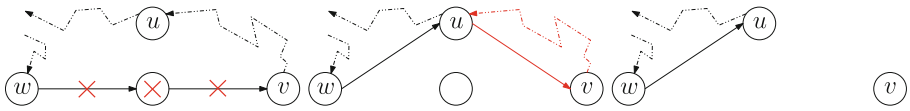
Algorithm 2. `LocalPathConfig`(i, u, v)

if $\exists \iota \in N_u$ *with* $v \in N_\iota$ *and* $l_{u\iota} + l_{\iota v} \leq l_{upreviously(i,v)} + l_{previous(i,v)v}$ **then**
  $w = \arg\max_{\iota \in N_u} T_\iota(\mathbf{x}_u^j)$
  send $\boldsymbol{join}[w](i, u, v)$
  $\Pi_{s_i c_i} \leftarrow$ replace v with w
else
  run `local_aodv+`(u, v, TTL)

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 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 [4].

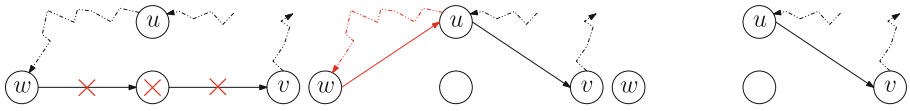
Joining New Paths, Modifying Existing Paths and Avoiding Loops. In this subsection, we briefly describe the functions regarding joining a new path and modifying an already existing path for loop elimination. Due to lack of space, we do not include the pseudocode of those functions; however, they can be found at the extended version of this paper [7]. `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 [7]). This message triggers the function `ModifyPath` (see [7]). 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 Figs. 1 and 2. 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 (Algorithm 2, line 6), then the loop freedom is guaranteed by the AODV path configuration process, which has been proven to be loop free [14].



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

Fig. 1. Loop avoidance - forward loop



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

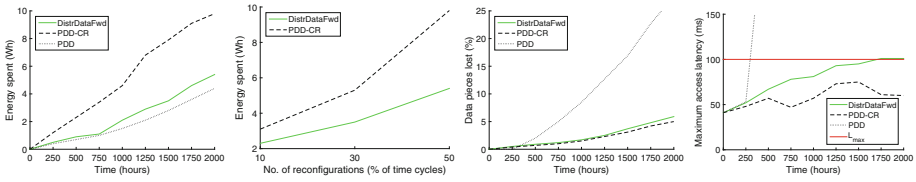
Fig. 2. Loop avoidance - backward loop

6 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. A table presenting the parameter configuration in details can be found in the extended version of the paper [7]. 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 the network controller, we set the transmission power to 15 dBm, typical of wireless LAN settings. We set the time cycle $\tau = 1$ s, the percentage of consumers over the population $0.05 - 45\%$ and we produce $1 - 8$ D_i/τ 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 [1], and set the latency threshold to $L_{\max} = 100$ 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 [6].

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 [6]. 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 [6]) 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 $\frac{\epsilon_{uv}(\tau)-\epsilon_{uv}(\tau-1)}{\epsilon_{uv}(\tau)} > \gamma$)[2].



(a) Energy spent (b) Reconfigurations (c) Data pieces lost (d) Max. acc. lat.

Fig. 3. Performance results.

Energy Efficiency. The energy consumption over the entire network during 2000 h of operation is depicted in Fig. 3a. 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

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

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 neighbourhood 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. 3b, 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 h of operation are depicted in Fig. 3c. 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 centralised 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 h of operation is depicted in Fig. 3d. The measured value is the maximum value observed 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 h 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.

7 Conclusion

We identified the need for a distributed reconfiguration method for data forwarding paths in industrial IoT networks. Given the operational parameters the network, we provided several efficient algorithmic functions which 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 is implemented, ensuring that there will be no loops. We demonstrated the performance gains of our method in terms of energy consumption and data delivery success rate compared to other state of the art solutions.

Acknowledgments. This work was funded by the European Commission through the FoF-RIA Project *AUTOWARE: Wireless Autonomous, Reliable and Resilient Production Operation Architecture for Cognitive Manufacturing* (No. 723909).

References

1. Network-based communication for Industrie 4.0. Publications of Plattform Industrie 4.0 (2016). www.plattform-i40.de. Accessed 01 Jan 2018
2. Ha, M., Kim, D.: On-demand cache placement protocol for content delivery sensor networks. In: 2017 International Conference on Computing, Networking and Communications (ICNC), pp. 207–216, January 2017
3. Han, S., Zhu, X., Mok, A.K., Chen, D., Nixon, M.: 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
4. Perkins, C., Belding-Royer, E., Das, S.: Ad hoc on-demand distance vector (AODV) routing. IETF RFC (3561), July 2003
5. Raptis, T.P., Passarella, A.: 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, October 2017
6. Raptis, T.P., Passarella, A., Conti, M.: 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
7. Raptis, T.P., Passarella, A., Conti, M.: Distributed path reconfiguration and data forwarding in industrial IoT networks. CoRR abs/1803.10971 (2018). http://arxiv.org/abs/1803.10971
8. Saifullah, A., Xu, Y., Lu, C., Chen, Y.: End-to-end communication delay analysis in industrial wireless networks. IEEE Trans. Comput. **64**(5), 1361–1374 (2015)
9. Sauter, T., Soucek, S., Kastner, W., Dietrich, D.: The evolution of factory and building automation. IEEE Ind. Electron. Mag. **5**(3), 35–48 (2011)
10. Shrouf, F., Ordieres, J., Miragliotta, G.: 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, December 2014
11. Sun, X., Ansari, N.: Traffic load balancing among brokers at the IoT application layer. IEEE Trans. Netw. Serv. Manag. **15**(1), 489–502 (2018)

12. Wollschlaeger, M., Sauter, T., Jasperneite, J.: The future of industrial communication: automation networks in the era of the internet of things and Industry 4.0. IEEE Ind. Electron. Mag. **11**(1), 17–27 (2017)
13. Xu, L.D., He, W., Li, S.: Internet of things in industries: a survey. IEEE Trans. Ind. Inform. **10**(4), 2233–2243 (2014)
14. Zhou, M., Yang, H., Zhang, X., Wang, J.: The proof of AODV loop freedom. In: 2009 International Conference on Wireless Communications Signal Processing, pp. 1–5, November 2009