DiGS: Distributed Graph Routing and Scheduling for Industrial Wireless Sensor-Actuator Networks

Junyang Shi, Mo Sha
Department of Computer Science
State University of New York at Binghamton
{jshi28, msha}@binghamton.edu

Zhicheng Yang
Department of Computer Science
University of California, Davis
zcyang@ucdavis.edu

***Abstract*—Wireless Sensor-Actuator Networks (WSANs) technology is appealing for use in industrial IoT applications because it does not require wired infrastructure. Battery-powered wireless modules easily and inexpensively retrofit existing sensors and actuators in industrial facilities without running cabling for communication and power. IEEE 802.15.4 based WSANs operate at low-power and can be manufactured inexpensively, which makes them ideal where battery lifetime and costs are important. Almost a decade of real-world deployments of WirelessHART standard has demonstrated the feasibility of using its core techniques including reliable graph routing and Time Slotted Channel Hopping (TSCH) to achieve reliable low-power wireless communication in industrial facilities. Today we are facing the 4th Industrial Revolution as proclaimed by political statements related to the Industry 4.0 Initiative of the German Government. There exists an emerging demand for deploying a large number of field devices in an industrial facility and connecting them through a WSAN. However, a major limitation of current WSAN standards is their limited scalability due to their centralized routing and scheduling that enhance the predictability and visibility of network operations at the cost of scalability. This paper decentralizes the network management in WirelessHART and presents the first *Di*stributed *G*raph routing and autonomous *S*cheduling (DiGS) solution that allows the field devices to compute their own graph routes and transmission schedules. Experimental results from two physical testbeds and a simulation study show our approaches can significantly improve the network reliability, latency, and energy efficiency under dynamics.**

***Index Terms*—Wireless Sensor-Actuator Networks, Industrial Internet of Things, Graph Routing, Transmission Scheduling**

I. Introduction

The Internet of Things (IoT) refers to a broad vision whereby *things* such as everyday objects, places, and environments are interconnected with one another via the Internet [1]. Until recently, most of the IoT infrastructure and applications development work by businesses have focused on smart homes and wearables. However, it is "production and manufacturing" cyber-physical system (CPS), underlying the 4th generation of industrial revolution (or Industry 4.0), that presents one of the largest economic impact potential of IoT [2] – up to $47 trillion in added value globally by 2025 (according to McKinsey's report on future disruptive technologies) [3].

Industrial networks, the underlying support of Industrial IoT (IIoT), typically connect hundreds or thousands of sensors and actuators in industrial facilities, such as steel mills, oil refineries, chemical plants, and infrastructures implementing complex monitoring and control processes. Although the typical process applications have low data rates, they pose unique challenges because of their critical demands for *reliable* and *real-time* communication in harsh industrial environments. Failing to achieve such performance can lead to production inefficiency, safety threats, and financial loss. These requirements have been traditionally met by specifically chosen wired solutions, e.g., Highway Addressable Remote Transducer (HART) [4], where cables connect sensors and forward sensor readings to a control room where a controller sends commands to actuators. However, wired networks are often costly to deploy and maintain in industrial environments and difficult to reconfigure to accommodate new production process requirements.

Wireless Sensor-Actuator Networks (WSANs) technology is appealing for use in industrial process applications because it does not require wired infrastructure. Battery-powered wireless modules easily and inexpensively retrofit existing sensors and actuators in industrial facilities without running cabling for communication and power. IEEE 802.15.4 based WSANs operate at low-power and can be manufactured inexpensively, which makes them ideal where battery lifetime and costs are important. Almost a decade of real-world deployments of WirelessHART standard [5] has demonstrated the feasibility of using its core techniques including reliable graph routing and Time Slotted Channel Hopping (TSCH) to achieve reliable low-power wireless communication in industrial facilities. Under graph routing, a packet is scheduled to reach its destination through multiple redundant paths to enhanced end-to-end reliability. TSCH requires that all devices in the network are time synchronized and hop channels to exploit frequency diversity.

Today we are facing the 4th Industrial Revolution as proclaimed by political statements related to the Industry 4.0 Initiative of the German Government [6]. There exists an emerging demand for deploying a large number of field devices in an industrial facility, e.g., hundreds of devices over an oil field, and connecting them through a WSAN. However, a major limitation of current WSAN standards such as WirelessHART is their *limited scalability* due to their centralized routing and scheduling that enhance the predictability and visibility of network operations at the cost of scalability. For instance, when encountering network dynamics (e.g., node or link failure, topology change), the centralized Network

Manager (a software module) in a WirelessHART network has to regenerate the routes and transmission schedule and then distribute them to all devices, introducing long delay and large overhead.

Recently, there has been an increasing interest in developing new distributed scheduling on top of the distributed tree-based routing protocols proposed in the wireless sensor networks (WSNs) literature (e.g., RPL [7] and CTP [8]) to replace the centralized routing and scheduling in industrial WSANs. For instance, the IETF created the 6TiSCH working group to standardize how to use an IPv6-enabled upper stack on top of IEEE 802.15.4e TSCH networks [9]. Duquennoy et al. developed the Orchestra that allows nodes in the RPL networks to compute their own schedules [10]. Unfortunately, the stringent reliability and real-time requirements of industrial applications distinguish traditional WSNs from industrial WSANs, that packet lost must become an exception and redundant routes between a source and a destination are essential to meet with guaranteed service. Our study shows that the networks relying on the tree-based routing suffer long repair time and insufficient reliability when encountering external interference and node failure.

This paper aims to address the stated scalability and reliability challenges; to our knowledge, it represents the first *Distributed Graph routing and autonomous Scheduling (DiGS)* solution that allows the field devices to compute their own graph routes and transmission schedules. Specifically, this paper makes the following contributions:

- We develop a distributed routing protocol that generates and operates with graph routes by extending RPL, the routing protocol for low-power IPv6 networks standardized by the IETF ROLL working group, with minimal changes;
- We design an autonomous scheduling approach that allows the field devices to compute their own transmission schedule autonomously based on the graph routes;
- We implement our proposed solution and evaluate it on two physical testbeds located in different cities as well as a simulator. Experimental results show our approaches can significantly improve the network reliability and latency under dynamics.

The remainder of the paper is organized as follows. Section II reviews related work and Section III introduces the background of WirelessHART networks. Section IV presents our empirical study and Sections V and VI describe the design of DiGS. Section VII presents the evaluation and Section VIII concludes the paper.

II. Related Works

Routing for wireless mesh networks and WSNs have been studied extensively in the literature. Multipath routing protocols (e.g., [11]–[15]) are proposed to enhance reliability by providing a few either node-disjoint or link-disjoint paths between source and destination. There also exist RPL based multipath routing protocols (e.g, [16]–[20], which are designed to balance the traffic load and energy consumption among nodes in the network. Comparing to these protocols, the graph routing specified in WirelessHART is designed to achieve high reliability by providing a high degree of routing redundancy to the TSCH networks. Its real-world deployments during the last decade have demonstrated the feasibility of achieving reliable low-power wireless communication in industrial facilities. Han et al. [21] and Wu et al. [22] proposed to generate graph routes in a centralized fashion, while Modekurthy et al. developed a protocol that generates graph routes distributedly based on the Bellman-Ford Algorithm [23]. In this paper, we develop the distributed graph routing protocol by extending the widely used RPL, integrate it with an autonomous scheduling approach, and demonstrate their performance when encountering network dynamics through extensive experiments on two physical testbeds.

There has been increasing interest in studying transmission scheduling for time-critical process monitoring and control applications over WirelessHART networks [24]–[27]. All these scheduling solutions designed to work with graph routing are centralized solutions which are designed to run on the centralized Network Manager. There also exists research on developing distributed scheduling for RPL networks [10], [27]–[32]. For instance, Duquennoy et al. developed the Orchestra that allows nodes in the RPL networks to compute their own schedules [10]. The IETF created the 6TiSCH working group to standardize how to use an IPv6-enabled upper stack on top of IEEE 802.15.4e TSCH networks [9]. However, our study shows that the network running RPL suffers long repair time and unsatisfactory reliability when encountering external interference and node failure. Another recent research direction is synchronous transmissions [33]–[37]. However, synchronous transmissions always require a centralized node to manage the synchronous transmissions. In contrast to the existing work, this paper presents the first autonomous scheduling approach that allows the field devices to compute their own schedule autonomously based on the graph routes.

III. Background of WirelessHART Networks

As shown in Figure 1, a WirelessHART network consists of a gateway, multiple access points, and a set of field devices (i.e., sensors and actuators) forming a multi-hop mesh network. The access points and field devices are equipped with half-duplex omnidirectional radio transceivers compatible with the IEEE 802.15.4 physical layer [38]. The multiple access points are wired to the gateway and provide redundant paths between the wireless network and the gateway. A WirelessHART network is managed by a centralized Network Manager. The Network Manager, a software module running on the gateway, is responsible for collecting the topology information from the devices, determining the routes and transmission schedule of the network, and disseminating them to all devices. WirelessHART adopts the centralized routing and scheduling that enhance the predictability and visibility of network operations at the cost of scalability. When encountering dynamics (e.g., node or link failure, topology

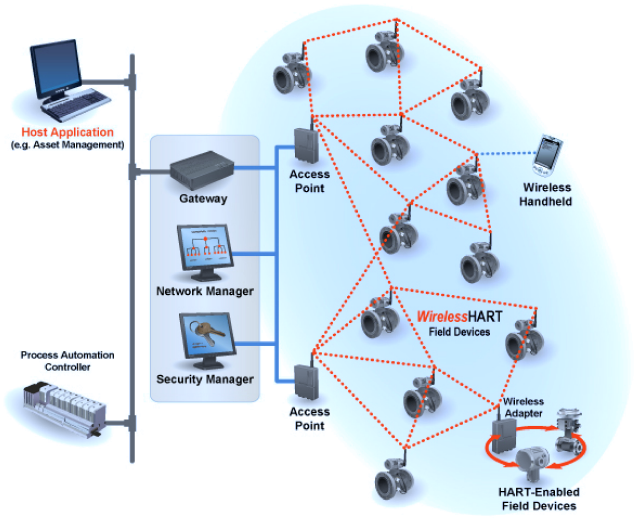


Fig. 1. Architecture of WirelessHART networks (Credit: HART Communication Foundation [4]).

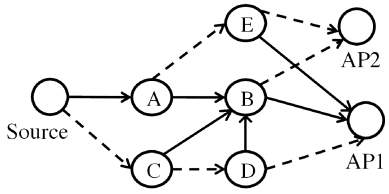


Fig. 2. A graph routing example. The solid lines represent the primary paths and the dashed lines represent the backup paths).

change), the Network Manager must regenerate the routes and transmission schedule and then distribute them to all devices, which introduce long delay and large overhead. To address this problem, our work is to develop a distributed graph routing and autonomous scheduling to enhance the scalability of the network.

Graph Routing: WirelessHART adopts graph routing to enhance end-to-end reliability by taking advantage of the route diversity. Graph routing involves a routing graph consisting of a directed list of paths between the field devices and access points. Graph routing consists of a single primary path and a backup path for each node. As illustrated in Figure 2, the packet may take backup routes (through node C, D, or E) to reach the access points (AP1 and AP2) if the links on the primary path (through nodes A and B) fail to deliver a packet. The graph routing specified by WirelessHART requires each node to have at least two outgoing paths. Based on the graph routes, the Network Manager allocates the time slots and channels to the devices to assure the packet deliveries.

TSCH MAC and Transmission Scheduling: TSCH technology inherits from WirelessHART and has been implemented as a MAC protocol, and was introduced as part of the IEEE 802.15.4e standard in 2012 for the industrial process control and automation [39]. WirelessHART employs the TSCH MAC that offers deterministic and collision-free communication.

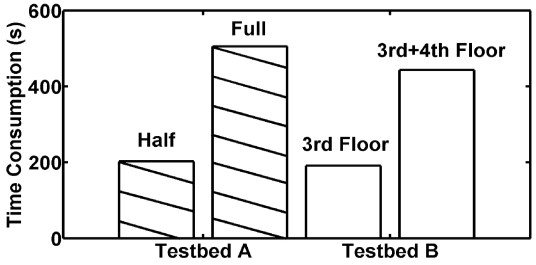


Fig. 3. Time consumed by the Network Manager in WirelessHART to update routes and transmission schedule.

Based on TSCH MAC, all nodes need to be globally time synchronized by exchanging the Enhanced Beacons (EBs) and the time synchronization trickles from the access points to the leaf nodes. Time is divided into 10 ms time slots, which are long enough for packet transmission and its acknowledgement (ACK); several time slots are grouped into one slotframe which appears periodically in every node. A TSCH schedule determines a node what to do in each time slot: transmit, receive, or sleep, and a time slot can either be dedicated or shared. In a dedicated slot, only one transmission is allowed in each channel which is fully contention free, while in a shared slot, two or more senders compete for a transmission in a CSMA/CA fashion. According to the time slot offset in one slotframe, the TSCH scheduling entity (Network Manager in WirelessHART) can determine whether to transmit a packet, receive a packet, or synchronize nodes to global time, etc. With our solution, the network no longer needs a centralized Network Manager to determine the functionality of every time slot. Each node computes its own primary and backup paths toward its destination based on its local topology information and the transmission schedule is automatically determined and updated once the network topology changes.

IV. Empirical Study

In this section, we present our empirical studies on the impact of interference and node failure on the performance of state of the art WSAN solutions (i.e., WirelessHART and Orchestra). Our empirical studies are conducted on two physical testbeds located in different cities: (1) *Testbed A* consisting of 50 TelosB motes [40] deployed in the second floor of a building in the campus of the State University of New York at Binghamton and (2) *Testbed B* featuring 44 TelosB motes spanning two floors of a building in the campus of Washington University in St. Louis (see Figure 8 for testbed deployments). To study the impact at different scales, we perform the measurement using four network topologies of different sizes and locations: (1) Half Testbed A with 20 nodes; (2) Full Testbed A with 50 nodes; (3) Half Testbed B with 19 nodes in one floor; and (4) Full Testbed B with 44 nodes spanning two floors.

Figure 3 shows the time consumed by the Network Manager in WirelessHART to collect topology information, regenerate the routes and transmission schedule, and disseminate them to

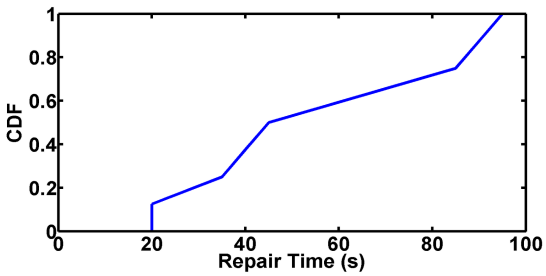


Fig. 4. CDF of repair time when the network encounters interference.

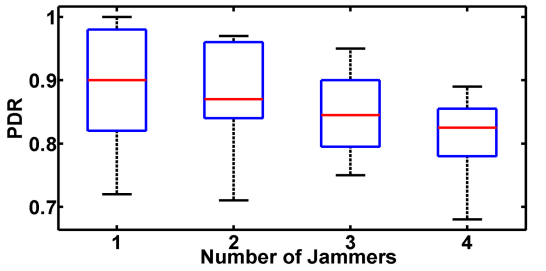


Fig. 5. PDR during the repair when the network encounters interference with different number of jammers.

all devices triggered by the events such as network topology changes and node/link failure. As showed in Figure 3, the Network Manager, running on a Dell Linux laptop with a 2.8 GHz Intel Core E3-1505M, spends 203s and 506s for Half Testbed A and Full Testbed A and 191s and 443s for Half Testbed B and Full Testbed B on reacting to network dynamics. These results illustrate the centralized routing and scheduling adopted by WirelessHART are insufficient for fast response to network dynamics since the network during the update has to operate under compromised routes and schedule leading to degraded performance.

Orchestra runs on top of RPL and schedules the transmissions in a distributed fashion. Figure 4 shows the cumulative distribution function (CDF) of the repair time used by Orchestra to update routes and transmission schedule when the network encounters the controlled interference generated by 1∼4 jammers running JamLab [41]. We repeat the experiments three times under each setting. The network repair time ranges from 20s to 95s (median: 45s) when the jammers generate signals emulating WiFi data streaming traffic[1]. We use the end-to-end packet delivery rate (PDR) as the metric for network reliability. The PDR of a data flow is defined as the percentage of packets that are successfully delivered to their destination. Figure 5 shows the PDRs of 8 data flows during the repair when 1∼4 jammers are present in the network. Low median PDRs (0.9, 0.87, 0.845, and 0.825) and large variations are observed in Figure 5. We observe similar results when using JamLab to generate jamming signals emulating Bluetooth. Orchestra requires much shorter repair time compared to WirelessHART and achieves high averaged delivery rates in clean environments [10], making it a good networking solution for many real-time applications. However, the repair time is still too long and its performance when encountering interference needs to be enhanced for those reliability-critical industrial WSANs that packet lost must become an exception to meet with guaranteed service. Our work is therefore an alternative approach that is complementary to Orchestra for reliability-critical industrial WSANs and further enhances the network reliability under network dynamics by developing new distributed graph routing and autonomous scheduling approaches.

[1]Co-existence of WSAN devices and WiFi is common in industrial deployments since WiFi is often used as backhauls to connect multiple WSANs.

V. Distributed Graph Routing

In this section, we first describe some terminologies and then introduce our distributed graph routing protocol that generates and operates with graph routes. Our protocol is extended from RPL [7], which is an oriented distance-vector routing protocol developed for low-power IPv6 networks and standardized by the IETF ROLL working group. Under RPL, nodes are organized in a Destination-Oriented DAG (DODAG) structure and the DODAG is rooted at the border router node (Internet access point). Each node is attached a rank, i.e., its distance to the root using a cost function (e.g., the expected transmission count (ETX) metric), and sends a packet towards the root by forwarding it to a neighbor node with a smaller rank. The routes generated by RPL are not graph routes since each node only has a single preferred parent in the parent set to which it sends packets. It is to be noted that RPL also allows to use multiple parents if those parents are equally preferred and have identical rank, while our protocol assigns two preferred parents to each node as default routes and builds the routing graph following the specification of WirelessHART[2].

Directed Acyclic Graph (DAG): In a DAG, all links are oriented in such a way that no cycle exists. All links selected for routing orient toward or terminate at the access points. Basically the DAG begins at the leaf nodes and ends at the access points which can ensure messages to be safely delivered to the destination without any cycle. The graph routes generated by our protocol form a DAG.

Best Parent and Second Best Parent: Each node has a best parent and a second best parent. The best parent locates on the primary path from the node to the access points with the smallest accumulated ETX. The path through the second best parent has the second smallest accumulated ETX and serves as a backup route.

Rank: Each node has a rank. All access points set their ranks to 1 and a field device sets its rank by increasing its best parent's rank by 1.

[2]In this paper, we focus on illustrating the generation of the uplink graph (from the field devices to the access points). Other graphs such as downlink graph and broadcast graph can be generated following the same method.

Weighted ETX: The weighted ETX (ETX_w) of a node is a cost function quantifying the distance to the access points through two routes:

$$ETX_w = \omega_1 * ETX_{abp} + \omega_2 * ETX_{asbp} \quad (1)$$

where ETX_{abp} is the accumulated ETX to the access point through the best parent and ETX_{asbp} is the accumulated ETX through the second best parent. ω_1 and ω_2 are two weighting factors defined as:

$$\omega_1 = 1 - (1 - 1/ETX_{bp})^2 \quad (2)$$

$$\omega_2 = (1 - 1/ETX_{bp})^2 \quad (3)$$

where ETX_{bp} denotes the ETX between the node and its best parent. According to WirelessHART, the transmission and first retransmission of a packet are scheduled through the primary route, while the second retransmission is scheduled through the backup route. Therefore, ω_1 represents the probability of a successful packet delivery during the first two transmission attempts and ω_2 represents the probability of the first two attempts fail.

Join-in Message: All nodes in the network broadcast the join-in messages periodically allowing new nodes to join the network. The join-in message contains the $rank$ and ETX_w of the node.

Joined-callback Message: Once a node selects its best or second best parent, it sends a joined-callback message to the selected node to inform the selection.

Our distributed graph routing algorithm is presented in Algorithm 1 which runs on the access points and field devices to construct the routing graph towards the access points. When a network starts, all access points initialize their $rank$ to 1 and ETX_w to 0 and then begin to broadcast the join-in messages. The rest nodes set their $rank$ and ETX_w to infinity. When a node receives the join-in messages from other nodes, it selects its best parent and second best parent based on the accumulated ETX values and then sets its rank by increasing its best parent's rank by 1. After joining the network, the node begins to broadcast the join-in messages.

The routing graph building procedure begins from the access points until reaching all leaf nodes. Each node selects its best and second best parents, as required by WirelessHART, towards the access points according to the accumulated ETX values. It is important to note that the initialized ETX between two nodes are determined by the Received Signal Strength (RSS). We empirically set $RSS_{min} = -90dBm$ and $RSS_{max} = -60dBm$. If the RSS value is larger than -60 dBm, the ETX is set to 1. If the RSS value is smaller than -90 dBm, the ETX is set to 3. The ETX in between is scaled proportionally between 1 and 3. The ETX value gets penalized if a transmission error occurs (e.g., no ACK).

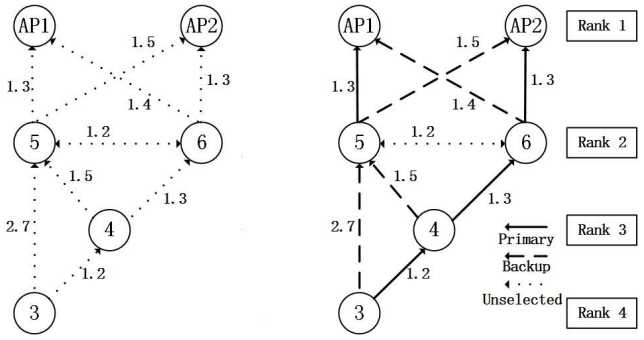
A node runs the Algorithm 1 when it receives a join-in message. The Trickle algorithm [42] is used to control the generation of the join-in messages. A timer varying from I_{min} to I_{max} is used to control the internal between two consecutive join-in messages. Specifically, the Trickle algorithm uses I_{min}

Algorithm 1: Distributed Graph Routing Algorithm

```
//Table I shows the notations
Input  : RootID, NodeID
Output: RouteTable
RouteTable ← NULL;
ETX_w(NodeID) = Rank(NodeID) = ∞;
if NodeID == RootID then
    //access point
    Set Rank = 1 and ETX_w = 0;
    Broadcast join-in messages;
end
if Rank(NodeID) == ∞ and NodeID! = RootID
 then
    //field device receives the first join-in message from i
    Set ETX_a(NodeID, i) =
      ETX(NodeID, i) + ETX_w(i);
    Set message sender as its best parent;
    Set ETX_min = ETX_a(NodeID, i);
    Set Rank(NodeID) = Rank(i) + 1;
    Send joined-callback messaget;
end
if Rank(NodeID)! = ∞ and NodeID! = RootID then
    //field device receives the non-first join-in message
      from i
    Set ETX_a(NodeID, i) = ETX(NodeID, i) +
      ETX_w(i);
    if ETX_a(NodeID, i) < ETX_min then
        Set its best parent as the second best parent;
        Set message sender as its best parent;
        Set ETX_min = ETX_a(NodeID, i)
        Set Rank(NodeID) = Rank(i) + 1;
        Send joined-callback message;
    end
    if ETX_a(NodeID, secondbestparent) >
      ETX_a(NodeID, i) >= ETX_min and
      Rank(i) < Rank(NodeID) then
        Set message sender as second best parent;
        Send joined-callback message;
    end
    ETX_w(NodeID) =
      ω_1 * ETX_a(NodeID, bestparent) + ω_2 *
      ETX_a(NodeID, secondbestparent);
    Broadcast join-in message;
end
if Receive joined-callback message then
    Update RouteTable and add the message sender as a
      child;
end
```

TABLE I
NOTATIONS USED IN ALGORITHM 1.

Symbol	Description
$ETX_w(i)$	Weighted ETX from node i to access points
$ETX_a(i,j)$	Accumulated ETX from node i to access points through node j
$ETX(i,j)$	ETX between node i and j
$ETX_{min}(i)$	Min accumulated ETX from node i to access points
$Rank(i)$	Rank of node i



(a) Network Topology. (b) Graph Routes.

Fig. 6. Example of the route generation.

as the first interval and then doubles the size of the interval until it reaches I_{max}. If a node detects a change of its own best parent or second best parent, it resets its Trickle timer to I_{min} to quickly update its ETX_w and $rank$ to its neighbors. The Trickle algorithm dynamically scales the interval length to enable fast yet low cost updates on ETX_w and $rank$.

A. Routing Example

Figure 6 shows an example with two access points and four field devices. The dash lines in Figure 6(a) denotes the links with the ETX values. When the network starts, AP1 and AP2 broadcast their $rank$ and ETX_w. #5 selects AP1 as its best parent and AP2 as its second best parent since $ETX_a(5, AP1)$ is smaller than $ETX_a(5, AP2)$. Similarly, #6 selects AP2 as its best parent and AP1 as its second best parent. Both #5 and #6 set their ranks to 2 and begin to broadcast the join-in messages. The link between #5 and #6 is not selected for routing since #5 and #6 have the same rank. This design is used to avoid loops. #4 selects #6 as its best parent since $ETX_a(4,6)$ has the smallest value and sets its rank to 3. #3 compares $ETX_a(3,4)$ with $ETX_a(3,5)$ to determine the best and second best parents. Figure 6(b) shows the generated graph routes. The solid lines represents the primary paths (#3→#4→#6→AP2 and #5→AP1) and the dash lines represents the backup routes (#3→#5, #4→#5, #5→AP2, and #6→AP1).

VI. AUTONOMOUS SCHEDULING

In this section, we introduce our autonomous transmission scheduling approach that allows the field devices to compute their own transmission schedule autonomously based on the graph routing presented in Sections V. Our scheduling approach has the salient feature that requires no schedule negotiation or sharing among neighboring nodes, which significantly reduces the communication overhead.

Following the suggestion in Orchestra, we separate the network traffic into three types: synchronization traffic, routing traffic, and application traffic. The EBs are used for time synchronization thus belong to the synchronization traffic. The join-in and joined-callback messages used to select parents are part of the routing traffic. The packets containing application data belong to the application traffic. Three slotframes with different periods are designed to carry different types of traffic. Under our scheduling approach, each node first generates three schedules following the slotframes based on its node id (e.g., MAC address), traffic demand, and routing table and then combines them into a single schedule to execute at runtime. Here are the key scheduling rules of our approach:

Use of Dedicated and Shared Slots: To achieve deterministic behavior, the synchronization and application traffic uses the contention-free dedicated slots, while the routing traffic employs the shared slots to accommodate network topology changes.

Assigning Slots for Synchronization: When a node attempts to join the network, it first snoops the channel to capture an EB from its neighbors. A captured EB allows a joining node to synchronize its clock and learn the transmission schedule currently used in the network. After the synchronization, the node selects its best and second best parents as presented in Sections V. Under our scheduling approach, the node i uses the ith slot in the synchronization slotframe to broadcast EB and jth slot to receive EB from its best parent (node j).

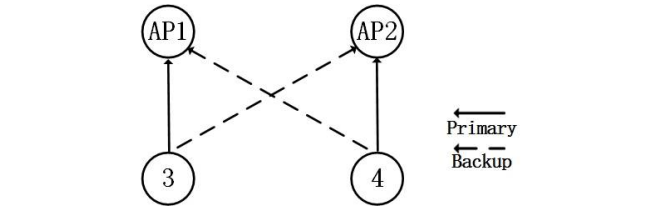
Assigning Slots for Routing: A fixed, shared slot in the routing slotframe is assigned for all nodes to exchange routing related packets including the join-in and joined-callback messages. All nodes in the network use the same time slot offset for the routing traffic.

Assigning Slots for Application: According to WirelessHART, multiple transmission attempts are scheduled for each packet through its primary and backup routes. The node's packet transmission and reception schedules are determined by its unique node id ($NodeID$) and parent-child relationship. Under our scheduling approach, a node uses the sth time slot in the application slotframe for the pth transmission attempt:

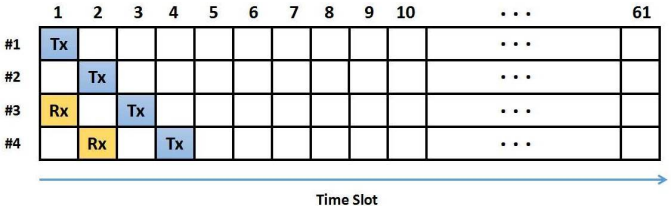
$$s = A * (NodeID - N_{AP}) - A + p \tag{4}$$

where A denotes the total number of transmission attempts for each packet and N_{AP} denotes the number of access points.

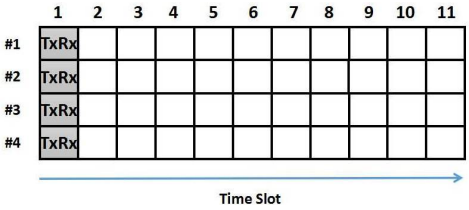
Schedule Combination: After generating the three individual schedules, the node then combines them to a single one for execution at runtime. To resolve slot assignment conflict during the combination, we assign different priorities to different types of traffic. The most critical synchronization traffic has the highest priority, while the application traffic has the lowest



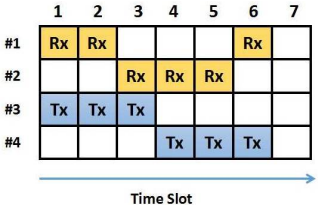
(a) Graph routes in the example.



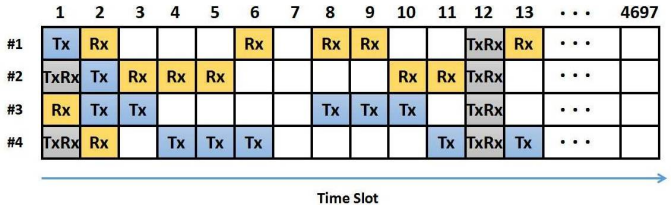
(b) Synchronization Schedule.



(c) Routing Schedule.



(d) Application Schedule.



(e) Combined Schedule

Fig. 7. Three schedules for different traffic and combined schedule.

priority. The schedule for traffic with lower priority yields during the combination. It is important to note that no traffic is constantly blocked since the three slotframes have different periods.

Section VI-A uses an example to illustrate the scheduling process and Section VI-B analyzes the performance.

A. Scheduling Example

Figure 7 illustrates our scheduling approach. Figure 7(a) shows the graph routes (primary paths: #3→#1, #4→#2; backup paths: #3→#2, #4→#1). In the example, the periods of the synchronization, routing, and application schedules (slotframe lengths) are assumed to be 61, 11, and 7 time slots, respectively. The combined schedule has $61 * 11 * 7 = 4697$ time slots in total. As Figure 7(b) showed, node #3 uses the third time slot to transmit its EB and receive the EB from its best parent in the first slot. Figure 7(c) shows the routing schedule which assigns the first slot for routing and Figure 7(d) shows the application schedule which deliveries a packet from #3 and a packet from #4 to the access points in every 7 time slots. Figure 7(e) shows the combined schedule. There exist conflicts during the combination. Each node resolves the conflicts locally. For example, #1 and #3 use the first slot for the synchronization traffic with highest priority in their combined schedule, while #2 and #4 use the slot for routing[3]. It is important to note that each node generates its combined schedule only based on local information requiring no schedule negotiation or sharing from its neighbors, which represents an important feature of our approach.

B. Performance Analysis

Under our scheduling approach, each slotframe repeats at a constant period and the transmission behavior is equivalent to Orchestra. The synchronization and application traffic using dedicated slots is by design contention-free, while the routing traffic utilizing shared slots has a contention probability:

$$p_c(routing) = \begin{cases} 1 - e^{-T*L/N}, & \text{if } L \geq N \\ 1 - e^{-T}, & \text{otherwise} \end{cases} \quad (5)$$

where T, N, and L denotes the average traffic load on the slot under a Poisson distribution, the number of nodes in the network, and the slotframe length. Here, for simplicity, we assume a simple network of N nodes, all connected to each other, and a single slotframe.

The probability of a slotframe A to be skipped due to a conflict with any other slotframe during the combination is:

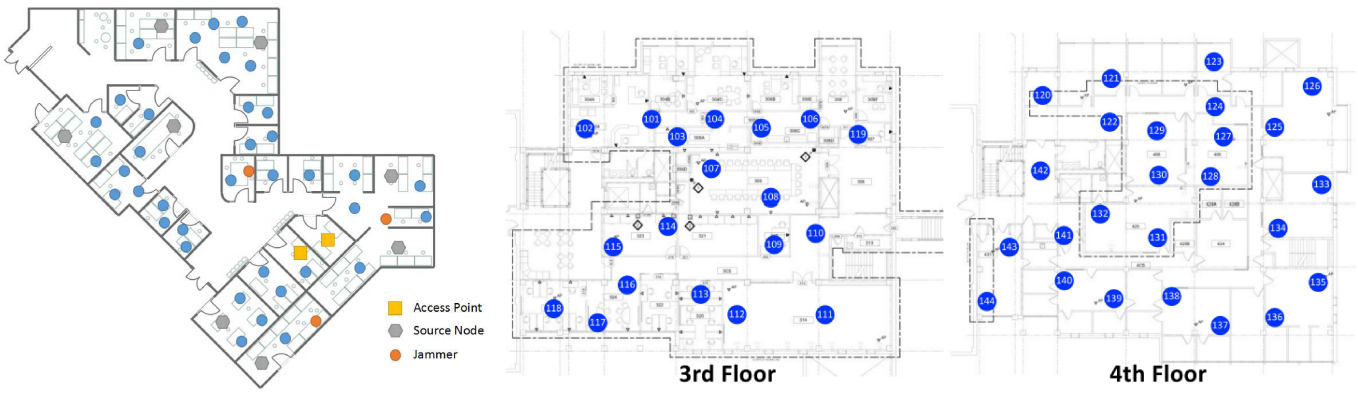
$$p_{skip(A)} = 1 - (\prod_{\forall B \in SF, B_{pri} > A_{pri}} (1 - p(conf_{A,B}))) \quad (6)$$

where SF denotes the set of all slotframes in the network, B_{pri} denotes the priority of B, and $p(conf_{A,B})$ denotes the event of a given slot in A conflicting with any slot in B. As reported in Orchestra, the probability of an application or routing slotframe to be skipped is expected to be very low in practice since the synchronization period determined by the hardware clock drift is much longer than the routing and application periods and the routing traffic is actually controlled by the Trickle algorithm. Our experimental results also confirm this and show high PDRs.

VII. Evaluation

We have implemented our solution (DiGS) in Contiki [43], an open source operating system for IoT, and evaluated it in three aspects: end-to-end reliability, end-to-end latency, and the energy consumption per received packet. To demonstrate the feasibility of our solution, we repeat the experiments on two physical testbeds located in the campuses of the

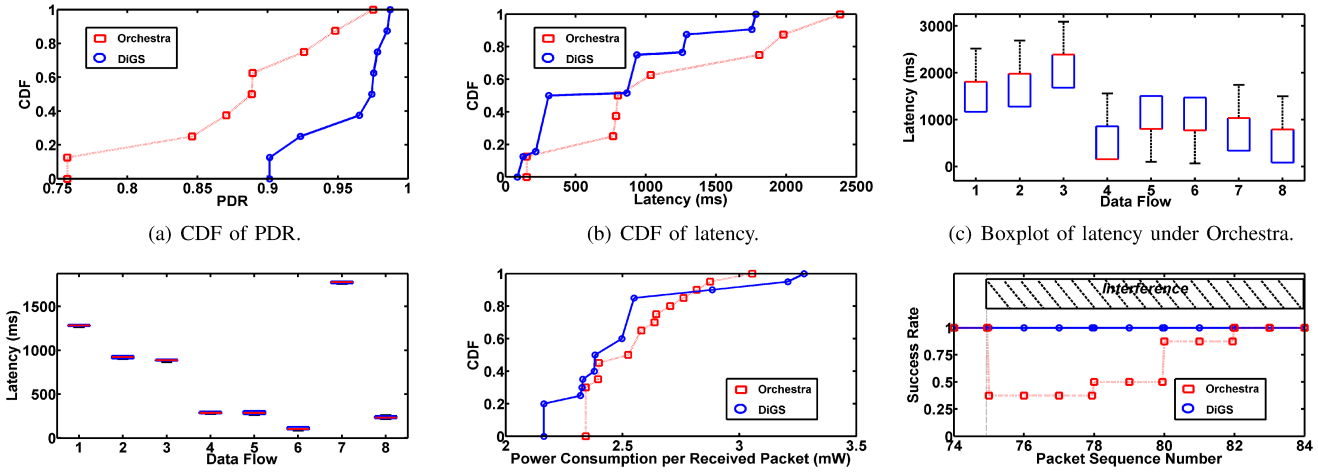
[3]Although the slot is assigned for routing, whether using it or not at runtime is controlled by the Trickle algorithm as discussed in Section V.



(a) Testbed A with 50 nodes.

(b) Testbed B with 44 nodes spanning two floors (Access Points: 130, 128; Source nodes: 144, 126, 136, 142, 115, 106; Jammers: 124, 141, 138.).

Fig. 8. Locations of the access points, field devices, and jammers in two Testbeds.



(a) CDF of PDR.

(b) CDF of latency.

(c) Boxplot of latency under Orchestra.

(d) Boxplot of latency under DiGS.

(e) CDF of power consumption per received packet

(f) A micro-benchmark measurement on the packet delivery success rate among 8 data flows.

Fig. 9. Performance under DiGS and Orchestra when the network encounters interference on Testbed A.

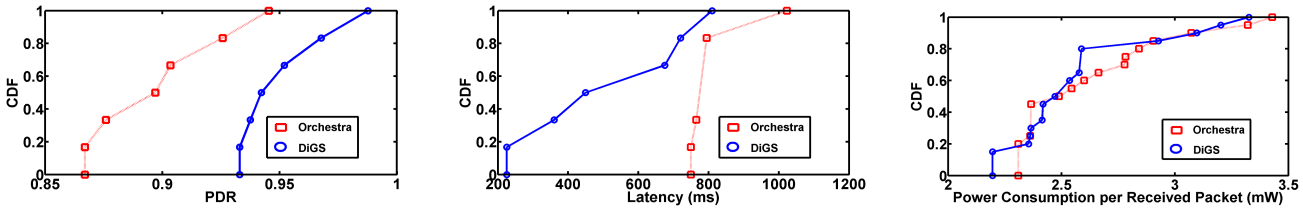
State University of New York at Binghamton and Washington University in St. Louis: (1) *Testbed A* consisting 50 TelosB motes deployed in the 2th floor of a building [44]; and (2) *Testbed B* featuring 44 TelosB motes spanning two floors of a building [45]. Figure 8 shows the testbed deployments. We run experiments on Testbed A with 300 flow sets, each of which contains 8 data flows that have different sources and destinations and repeats the experiments on Testbed B with 220 flow sets, each of which contains 6 flows. Two access points are configured on each testbed. Each source node generates a packet in every 5 seconds. We set the length of synchronization, routing, and application slotframes to 557, 47, and 151 time slots, respectively, for all experiments.

We observe that Orchestra significantly outperforms WirelessHART under network dynamics (see Section IV), therefore compare our solution against Orchestra[4] instead of WirelessHART and examine their performance under two scenarios: one under interference (Section VII-A) and the other with node failure (Section VII-B). JamLab is used to generate controlled interference with different strength and pattern. We also measure the efficiency of DiGS to initialize the network (Section VII-C) and perform a simulation study with 150 nodes in the Cooja simulator [46] (Section VII-D).

A. Performance under Interference

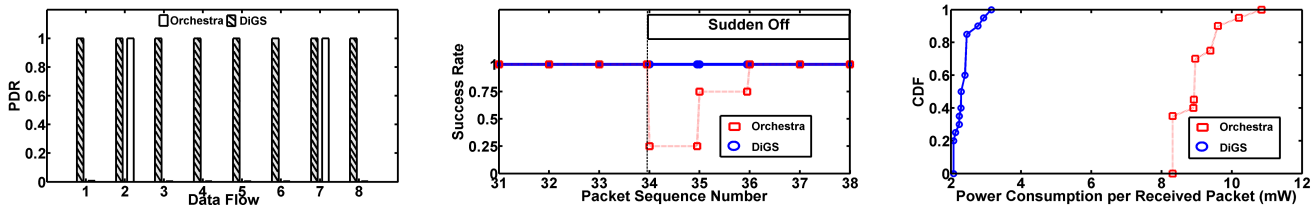
As Figure 8 showed, we configure three nodes to run JamLab and generate signals emulating WiFi data streaming traffic. To create a larger interference range and emulate the higher transmission power employed by 802.11, we configure the nodes running JamLab to transmit at higher transmission powers. Figure 9 shows the performance under DiGS and Orchestra when the network encounters interference. Figure 9(a) plots the CDF of PDR. On average, DiGS achieves 8.3% higher PDR than Orchestra. In addition, 75.0% of the flow sets under DiGS achieve PDRs higher than 95.0%, while

[4]We use the Orchestra implementation in Contiki provided by the authors in [10].



(a) CDF of PDR. (b) CDF of latency. (c) CDF of power consumption per received packet.

Fig. 10. Performance under DiGS and Orchestra when the network encounters interference on Testbed B.



(a) PDR of each data flow. (b) A micro-benchmark measurement on the packet delivery success rate among 8 flows on Testbed A. (c) CDF of power consumption per received packet.

Fig. 11. Performance under DiGS and Orchestra when the network encounters node failure on Testbed A.

only 12.5% under Orchestra provide that. More importantly, DiGS delivers a significant improvement over Orchestra in the worst-case PDR (from 76.0% to 90.3%), which represents a significant advantage in industrial applications that demand high reliability in harsh industrial facilities. The higher PDRs provided by DiGS under interference benefit from the route diversity offered by the graph routing.

As Figure 9(b) showed, DiGS reduces the median latency from 917.5ms to 601.3ms and averaged latency from 1214.1ms to 649.5ms compared to Orchestra. The reduced latency provided by DiGS represents a significant advantage in industrial applications allowing it to employ control loops with tighter deadlines. Moreover, as shown in boxplots Figure 9(c) and Figure 9(d), DiGS achieves a smaller variation of latency than Orchestra, which represents another significant advantage in industrial applications that demand predictable performance. This result shows that DiGS employing the distributed graph routing is indeed more resilient to interference thanks to route diversity. Figure 9(e) shows the CDF of power consumption per received packet under DiGS and Orchestra[5]. DiGS provides an average of 0.056mW decrease in power consumption per received packet compared to Orchestra. Although the idle listening overhead introduced by DiGs leads to moderate increases in total energy consumption, the slight increases in power consumption are in exchange for a significant improvement on reliability, resulting in an overall reduction on power consumption per received packet. Figure 9(f) plots a micro-benchmark measurement on the packet delivery success rate among 8 data flows between the $74th$ and $84th$ packets are forwarded in the network. When encountering the controlled interference, 3 flows lose the $75th$, $76th$, and $77th$ packets when running Orchestra. Those flows recover from the packet lost and successfully deliver the $78th$, $80th$, and $82th$ packets, respectively. Orchestra consumes 35s to recover from interference by updating the routing and scheduling, while DiGS provides seamlessly packet delivery during the process.

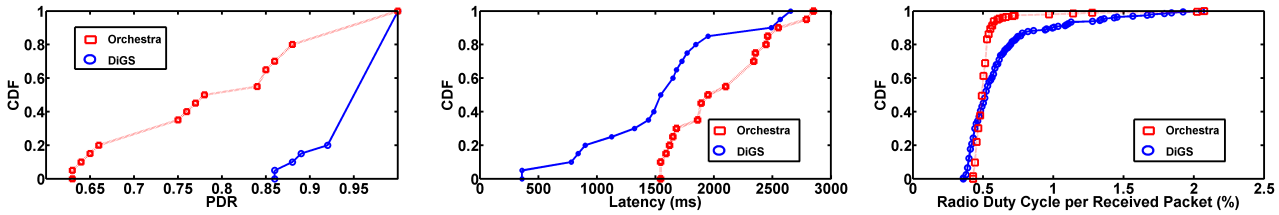
Similar gains are seen for DiGS on Testbed B. As Figure 10(a) showed, under the configuration of 6 data flows, DiGS achieves a worst-case PDR of 93.2%, a median PDR of 94.5%, and a $90th$ percentile PDR of 97.7%, outperforming Orchestra by 7.6%, 5.2%, and 4.7%, respectively. As shown in Figure 10(b), the improvements offered by DiGS in worst-case latency and median latency are 213.0ms and 232.7ms, respectively. As Figure 10(c) showed, DiGS also provides higher energy efficiency when encountering interference over Orchestra (i.e., 0.057mW decrease in the power consumption per received packet), resulting from the significant improvement on reliability.

B. Performance with Node Failure

We also explored DiGS's performance with node failure by turning off 4 nodes on the routing graph in turn. We repeat the experiments for 34 times. Figure 11 shows the performance comparison between DiGS and Orchestra when the network encounters node failure on Testbed A. As Figure 11(a) showed, 6 of the total 8 data flows becomes completely disconnected under Orchestra after the nodes fail, while all flows still achieve a 100% PDR under DiGS.

Figure 11(b) plots a micro-benchmark measurement on the packet delivery success rate among 8 data flows when a node suddenly fails. 6 data flows are affected and lose the $34th$ packet and then recover after 10s when running Orchestra, while DiGS successfully delivers all packets through backup routes. As Figure 11(c) showed, DiGS survives node failure without losing any packet and achieves a 9.01mW decrease on power consumption per received packet compared to Or-

[5]We only consider the power consumed by the radio and estimate it based on the timestamps of radio activities and the radio's power consumption in each state according to the CC2420 data sheet [47]



(a) CDF of PDR. (b) CDF of latency. (c) CDF of radio duty cycle per received packet.

Fig. 12. Simulation with 150 nodes in Cooja Simulator.

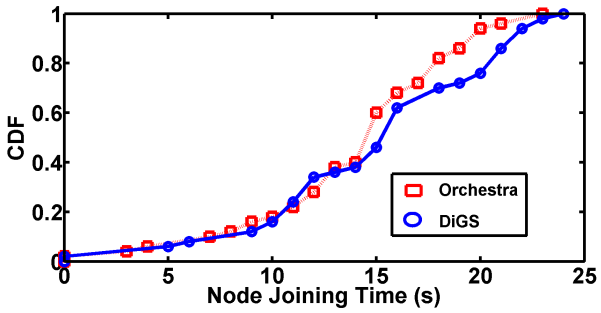


Fig. 13. Network initialization time comparison between DiGS and Orchestra.

chestra. As shown in Figure 11, DiGS provides significant improvements on failure tolerance and energy efficiency over Orchestra, which are critical properties for industrial applications.

C. Network Initialization

To study the efficiency of DiGS to initialize the network, we measure the time duration of each node joining the network (i.e., between the network start and each node synchronizing with the network and setting its preferred parents). Figure 13 shows the CDF of joining time of 50 nodes on Testbed A under DiGS and Orchestra. DiGS does result in a slight increase in network initialization time (from 23.0s to 24.1s) compared to Orchestra as a result of one more preferred parent selected by each node to construct the network. The averaged joining times of 50 nodes are 15.4s and 14.3s under DiGS and Orchestra, respectively. The slight increases in network initialization are in exchange for moderately enhancing the reliability and latency when the network encounters interference and node failure. This tradeoff makes DiGS well-suited for industrial applications running in dynamic environments with critical performance demands.

D. Simulation Study with 150 Nodes

To explore DiGS's performance at a larger scale, we perform a simulation study using the Cooja simulator. In the simulations, 150 nodes and two access points are placed in a 300m X 300m area. We run simulations with 300 flow sets, each of which contains 20 data flows that have different sources and destinations. Each source node generates a packet in every 10 seconds. 5 Cooja disturber nodes are configured to turn on and off in every 5 minutes to interfere nearby links.

Figure 12 shows the performance under DiGS and Orchestra when the network encounters interference. Figure 12(a) presents the CDF of PDR. On average, DiGS achieves 16.3% higher PDR than Orchestra. In addition, 53.0% of the flow sets under DiGS achieve PDRs higher than 95.0%, while only 11.0% under Orchestra provide that. Moreover, DiGS delivers a significant improvement over Orchestra in the worst-case PDR (from 86.7% to 63.0%). As Figure 12(b) showed, DiGS reduces the median latency from 1950.0ms to 1560.0ms and averaged latency from 2068.6ms to 1565.7ms compared to Orchestra. DiGS improves the reliability and latency under interference at the cost of slight increases on the radio duty cycle. As shown in Figure 12(c), DiGS suffers an average of 0.056% increase on radio duty cycle per received packet over Orchestra. The slight increases in duty cycle per received packet are in exchange for a critical improvement on reliability and latency.

VIII. Conclusions

The "production and manufacturing" CPS, underlying the Industry 4.0, that presents one of the largest economic impact potential of IoT. IEEE 802.15.4 based WSANs are appealing for use in industrial IoT applications since they operate at low-power and can be manufactured inexpensively. Almost a decade of real-world deployments of WirelessHART standard has demonstrated the feasibility of using its core techniques including reliable graph routing and TSCH to achieve reliable low-power wireless communication in industrial facilities. However, a major limitation of current WSAN standards is their limited scalability due to their centralized routing and scheduling that enhance the predictability and visibility of network operations at the cost of scalability. This paper decentralizes the network management in WirelessHART and presents the first distributed graph routing and autonomous scheduling solution that allows the field devices to compute their own graph routes and transmission schedules. Experimental results from two physical testbeds and a large-scale simulation show our solution provides significant improvement on network reliability, latency, energy efficiency, and failure tolerance under dynamics, critical properties for industrial applications, over state of the art at the cost of slightly higher power consumption and longer network initialization.

Acknowledgment

The authors thank Dr. Chenyang Lu and Dolvara Gunatilaka at Washington University in St. Louis for facilitating the

experiments on their testbed. This work was supported by the NSF through grant CRII-1657275 (NeTS).

REFERENCES

[1] M. E. Porter and J. E. Heppelmann, "How smart, connected products are transforming competition," *Harvard Business Review*, vol. 92, no. 11, pp. 64–88, 2014.

[2] A. Thierer and A. Castillo, "Projecting the growth and economic impact of the internet of things," Jun 2015. [Online]. Available: https://www.mercatus.org/publication/projecting-growth-and-economic-impact-internet-things

[3] J. Manyika, M. Chui, J. Bughin, R. Dobbs, P. Bisson, and A. Marrs, "Disruptive technologies: Advances that will transform life, business, and the global economy," May 2013. [Online]. Available: http://www.mckinsey.com/business-functions/digital-mckinsey/our-insights/disruptive-technologies

[4] HART Communication Protocol and Foundation (Now FieldComm Group). [Online]. Available: http://www.hartcomm.org/

[5] WirelessHART. [Online]. Available: https://fieldcommgroup.org/technologies/hart

[6] H. Kagermann, W. Wahlster, and J. Helbig. (April 2013) Recommendations for Implementing the Strategic Initiative Industrie 4.0. [Online]. Available: http://www.acatech.de/fileadmin/user%5fupload/Baumstruktur%5fnach%5fWebsite/Acatech/root/de/Material%5ffuer%5fSonderseiten/Industrie%5f4.0/Final%5freport%5f%5fIndustrie%5f4.0%5faccessible.pdf

[7] T. W. (Ed.), P. T. (Ed.), A. Brandt, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, J. Vasseur, and R. Alexander, "RFC 6550," in *RPL: IPv6 Routing Protocol for Low power and Lossy Networks*, 2012.

[8] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis, "Collection Tree Protocol," in *Sensys*, 2009.

[9] IETF 6TiSCH working group. [Online]. Available: https://datatracker.ietf.org/wg/6tisch/

[10] S. Duquennoy, B. Al Nahas, O. Landsiedel, and T. Watteyne, "Orchestra: Robust Mesh Networks Through Autonomously Scheduled TSCH," in *Sensys*, 2015.

[11] Z. Ye, S. V. Krishnamurthy, and S. K. Tripathi, "A Framework for Reliable Routing in Mobile Ad Hoc Networks," in *INFOCOM*, 2003.

[12] D. Ganesan, R. Govindan, S. Shenker, and D. E. Highlyresilient, "Energy-Efficient Multipath Routing in Wireless Sensor Networks," in *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 5, no. 4, 2001.

[13] M. Radi, B. Dezfouli, K. A. Bakar, S. A. Razak, and T. Hwee-Pink, "Im2pr: Interference-Minimized Multipath Routing Protocol for Wireless Sensor Networks," in *Wireless Networks*, vol. 20, no. 7, 2014.

[14] K. X. J. Zhang and H. J. Chao, "Load Balancing in IP Networks Using Generalized Destination-Based Multipath Routing," in *IEEE/ACM Transactions on Networking*, vol. 23, no. 6, 2015.

[15] H. Geng, X. Shi, X. Yin, Z. Wang, and H. Zhang, "Algebra and Algorithms for Efficient and Correct Multipath QoS Routing in Link State Networks," in *IWQoS*, 2015.

[16] Q. Le, T. Ngo-Quynh, and T. Magedanz, "Rpl-based multipath routing protocols for internet of things on wireless sensor networks," in *International Conference on Advanced Technologies for Communications*, 2014, pp. 424–429.

[17] B. Pavkovi, F. Theoleyre, and A. Duda, "Multipath opportunistic rpl routing over ieee 802.15.4," in *The 14th ACM international conference on Modeling, analysis and simulation*, 2011, pp. 179–186.

[18] O. Landsiedel, E. Ghadimi, S. Duquennoy, and M. Johansson, "Low power, low delay: Opportunistic routing meets duty cycling," in *ACM/IEEE 11th International Conference on Information Processing in Sensor Networks*, 2012, pp. 185–196.

[19] O. Iova, F. Theoleyre, and T. Noel, "Exploiting multiple parents in rpl to improve both the network lifetime and its stability," in *IEEE International Conference on Communications*, 2015, pp. 610–616.

[20] Z. Wang, L. Zhang, Z. Zheng, and J. Wang, "An optimized rpl protocol for wireless sensor networks," in *IEEE 22nd International Conference on Parallel and Distributed Systems*, 2016, pp. 294–299.

[21] S. Han, X. Zhu, A. K. Mok, D. Chen, and M. Nixon, "Reliable and Real-Time Communication in Industrial Wireless Mesh Networks," in *RTAS*, 2011.

[22] C. Wu, D. Gunatilaka, A. Saifullah, M. Sha, P. B. Tiwari, C. Lu, and Y. Chen, "Maximizing Network Lifetime of WirelessHART Networks under Graph Routing," in *IoTDI*, 2016.

[23] V. Modekurthy, A. Saifullah, and S. Madria, "Distributed graph routing for wirelesshart networks," in *International Conference on Distributed Computing and Networking (ICDCN)*, 2018.

[24] A. Saifullah, Y. Xu, C. Lu, and Y. Chen, "End-to-end Communication Delay Analysis in Industrial Wireless Networks," in *IEEE Transactions on Computers*, vol. 64, no. 5, 2014.

[25] C.Wu, M. Sha, D. Gunatilaka, A. Saifullah, C. Lu, and Y. Chen, "Analysis of EDF Scheduling for Wireless Sensor-Actuator Networks," in *IWQoS*, 2014.

[26] S. Zhang, G. Zhang, A. Yan, Z. Xiang, and T. Ma, "A Highly Reliable Link Scheduling Strategy for WirelessHART Networks," in *ATC*, 2013.

[27] X. Zhu, P.-C. Huang, S. Han, A. Mok, D. Chen, and M. Nixon, "RoamingHART: A Collaborative Localization System on WirelessHART," in *RTAS*, 2012.

[28] N. Burri, P. V. Rickenbach, and R. Wattenhofer, "Dozer: Ultra-Low Power Data Gathering in Sensor Networks," in *IPSN*, 2007.

[29] A. Tinka, T. Watteyne, K. S. J. Pister, and A. M. Bayen, "A Decentralized Scheduling Algorithm for Time Synchronized Channel Hopping," in *EAI Endorsed Transactions on Mobile Communications and Applications*, vol. 11, no. 1, 2011.

[30] M. R. Palattella, N. Accettura, M. Dohler, L. A. Grieco, and G. Boggia, "Traffic Aware Scheduling Algorithm for Reliable Low-power Multi-hop IEEE 802.15.4e Networks," in *PIMRC*, 2012.

[31] A. Morell, X. Vilajosana, J. L. Vicario, and T. Watteyne, "Label Switching over IEEE 802.15.4e Networks," in *Transactions on Emerging Telecommunications Technologies*, vol. 24, no. 5, 2013.

[32] P. Zand, A. Dilo, and P. Havinga, "D-MSR: A Distributed Network Management Scheme for Real-Time Monitoring and Process Control Applications in Wireless Industrial Automation," in *D-MSR: A Distributed Network Management Scheme for Real-Time Monitoring and Process Control Applications in Wireless Industrial Automation*, vol. 13, no. 7, 2013.

[33] F. Ferrari, M. Zimmerling, L. Mottola, and L. Thiele, "Low-Power Wireless Bus," in *SenSys*, 2012.

[34] F. Ferrari, M. Zimmerling, L. Thiele, and O. Saukh, "Efficient Network Flooding and Time Synchronization with Glossy," in *SenSys*, 2013.

[35] O. Landsiedel, F. Ferrari, and M. Zimmerling, "Chaos: Versatile and Efficient All-to-All Data Sharing and In-Network Processing at Scale," in *SenSys*, 2013.

[36] M. Doddavenkatappa, M. C. Chan, and B. Leong, "Splash: Fast Data Dissemination with Constructive Interference in Wireless Sensor Networks," in *NSDI*, 2013.

[37] M. Doddavenkatappa and M. C. Chan, "P3: A Practical Packet Pipeline using Synchronous Transmissions for Wireless Sensor Networks," in *IPSN*, 2014.

[38] *Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs)*, IEEE, 2006.

[39] "IEEE 802.15.4e WPAN Task Group." [Online]. Available: http://www.ieee802.org/15/pub/TG4e.html

[40] TelosB: Telosb Mote Platform, Datasheet Provided by MEMSIC Inc. [Online]. Available: http://www.memsic.com/userfiles/files/Datasheets/WSN/telosb%5fdatasheet.pdf

[41] C. A. Boano, T. Voigt, C. Noda, K. Rmer, and M. Ziga, "JamLab: Augmenting sensornet testbeds with realistic and controlled interference generation," in *IPSN*, 2011.

[42] P. Levis, T. Clausen, J. Hui, O. Gnawali, and J. Ko, "RFC 6206," in *The Trickle Algorithm*, 2011.

[43] Contiki: The Open Source OS for the Internet of Things. [Online]. Available: http://www.contiki-os.org/

[44] "Testbed at the State University of New York at Binghamton." [Online]. Available: http://www.cs.binghamton.edu/\%7emsha/testbed

[45] "Testbed at the Washington University in St. Louis." [Online]. Available: http://cps.cse.wustl.edu/index.php/Testbed

[46] Cooja Simulator. [Online]. Available: https://github.com/contiki-os/contiki/wiki/An-Introduction-to-Cooja

[47] CC2420: 2.4 GHz IEEE 802.15.4 ZigBee-ready RF Transceiver, Datasheet Provided by TI Inc. [Online]. Available: http://www.ti.com/lit/ds/symlink/cc2420.pdf