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

Junyang Shi, *Student Member, IEEE,* Mo Sha, *Member, IEEE,* and Zhicheng Yang

***Abstract*—Wireless Sensor-Actuator Networks (WSANs) technology is appealing for use in industrial Internet of Things (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].

Junyang Shi and Mo Sha are with the Department of Computer Science, State University of New York at Binghamton, Binghamton, NY, 13902 USA (e-mail: jshi28@binghamton.edu; msha@binghamton.edu).

Zhicheng Yang are with the Department of Computer Science, University of California, Davis, CA 95616, 13902 USA (e-mail: zcyang@ucdavis.edu).

Manuscript received XXX XX, 2018; revised XXX XX, 201X. This work was supported by the U.S. National Science Foundation through grant CRII-1657275 (NeTS).

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 approaches, which run on top of the distributed routing protocols developed for wireless sensor networks (WSNs), such as the Collection Tree Protocol (CTP) [7] and the IPv6 Routing Protocol for Low power and Lossy Networks (RPL) [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 abovementioned 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 in a distributed fashion. 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 two autonomous scheduling approaches that allow the field devices to compute their own transmission schedule autonomously based on the graph routes; the first approach provides shorter end-to-end latency, while the later completely eliminates the scheduling conflicts among different types of traffic;
- 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 introduces our conflict deferral scheduling approach. Section VIII presents the evaluation and Section IX concludes the paper.

II. Related Works

Routing for wireless mesh networks and WSNs have been studied extensively in the literature. CTP is a routing protocol that computes anycast routes to a single or a small number of designated sinks in a wireless sensor networkt [7]. CTP has been used in research, teaching, and in commercial products. Experiences with CTP has also informed the design of RPL [8]. However, both CTP and RPL are tree-based routing protocols and cannot generate graph routes which are specified in WirelessHART to achieve high reliability. Thus, they are not suitable for those mission-critical industrial applications, where packet lost must become an exception. In contrast, 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] developed two algorithms to generate graph routes in a centralized fashion, while Modekurthy et al. proposed to use the Bellman-Ford algorithm to generate the graph routes [23] in a distributed fashion. Comparing to these efforts, we developed the first RPL-based distributed routing protocol that generates and operates with graph routes. More important, we developed two transmission scheduling approaches, which run on top of our proposed routing protocol, providing a complete networking solution.

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

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

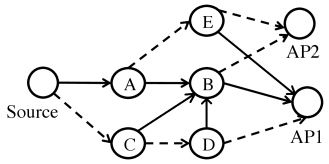


Fig. 1. A graph routing example. The solid lines represent the primary paths and the dashed lines represent the backup paths.

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 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 1, 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. 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[1] and Orchestra[2]). Our empirical studies are conducted on two physical testbeds located in different cities: (1) *Testbed A* consisting of 50 TelosB motes [42] 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. 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. We use graph routing and the rate monotonic scheduling [43] to generate transmission schedules. We set up six data flows for full Testbed A and B and three data flows for half Testbed A and B with different sources, destinations, and data period. The data period of each data flow is selected within the range of $2^{0\sim7}$ seconds. Priorities are assigned inversely to the period of each data flow, giving higher priority to data flows with shorter periods.

Figure 2 shows the time consumed by the Network Manager in WirelessHART to collect topology information, regenerate the routes and transmission schedule, and disseminate them to all devices triggered by the events such as network topology changes and node/link failure. As showed in Figure 2, 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 3 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]We use the WirelessHART implementation provided by Li et al. in our experiments. The implementation is publicly accessible [40].

[2]We use the Orchestra implementation provided by Duquennoy et al. The implementation is publicly accessible [41]

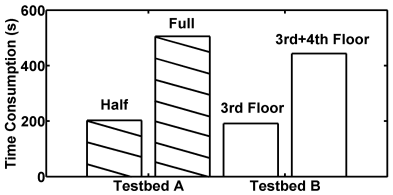


Fig. 2. Time consumed by the Network Manager in WirelessHART to update routes and transmission schedule.

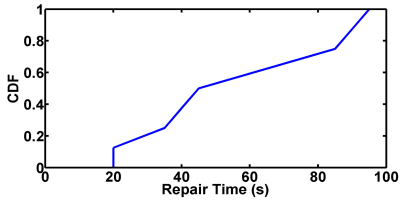


Fig. 3. CDF of repair time when the network encounters interference.

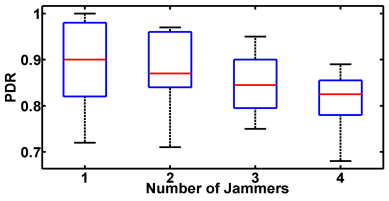


Fig. 4. PDR during the repair when the network encounters interference with different number of jammers.

1~4 jammers running JamLab [44]. 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[3]. 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 4 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 4. 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.

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 [8], 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[4].

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

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

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.

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:

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

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

[4]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.

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.

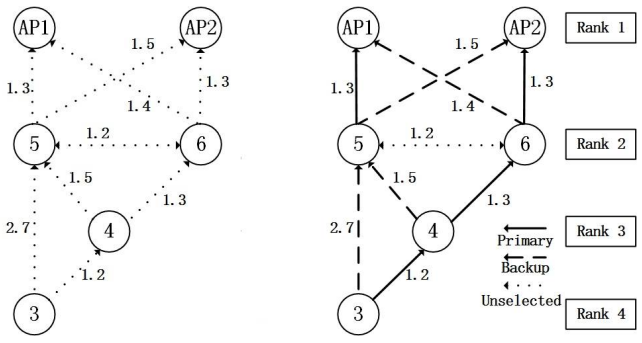
The routing graph building procedure begins from the access points until reaching all leaf nodes. Each node selects its best and second best parents, as required by WirelessHART, towards the access points according to the accumulated ETX values. It is important to note that the initialized ETX between 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), as Eq. 4 shows.

$$ETX = ETX_{old} * \alpha + P * (1 - \alpha) \tag{4}$$

where ETX_{old} is the ETX value before apply the penalty, P is the penalty coefficient, and α is a weighting factor ranging between 0 and 1. We use the values suggested by RPL for P and α, where α equals to 10% and P equals to 16. A node runs the Algorithm 1 when it receives a join-in message. The Trickle algorithm [45] 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} 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 $rank$.

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) = ω1 * ETX_a(NodeID, bestparent) +
      ω2 * 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
```



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

Fig. 5. Example of the route generation.

A. Routing Example

Figure 5 shows an example with two access points and four field devices. The dash lines in Figure 5(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 5(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. The synchronization slotframe period is determined by the time drift of the device's clock. The routing slotframe period is determined by the intervals of routing updates which are required by the routing protocol. The application slotframe period is determined by the application traffic needs. Our scheduling approach first assigns time slots in those three slotframes for transmissions and then combines them into a single one for runtime execution. 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 by using its carried timestamp and learn the transmission schedule currently used in the network. The APs in the network are wired to the gateway and time synchronized through the gateway. 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 and compete the slot in a CSMA fashion. In our implementation, we use the first time slot in the routing slotframe to exchange routing packets.

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 ($NodeID$[5].) and parent-child relationship. Under our scheduling approach, a node uses the sth time slot in the application slotframe for the pth transmission attempt:

$$s = A * (NodeID - N_{AP}) - A + p \tag{5}$$

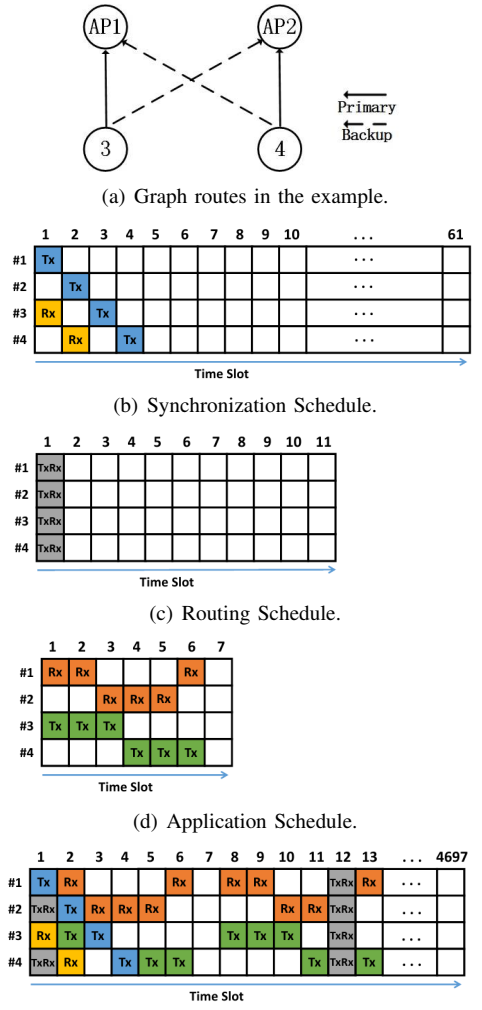
where A denotes the total number of transmission attempts for each packet and N_{AP} denotes the number of access points. Each node can learn its neighbors' $NodeID$s from the routing table and N_{AP} and A are global information shared by all nodes in the network and can be carried by EBs.

Algorithm 2 shows the pseudocode on how a node assigns time slots for the synchronization, routing, and application traffic. Based on the abovementioned policy, the node i uses the dedicated ith slot in the synchronization slotframe to exchange EB and uses the first time slot (shared) in the routing slotframe to exchange routing information. The node i uses sth slot (according to Eq. 5) in the application slotframe to send or receive application traffic.

Algorithm 2: Algorithm that assigns time slots for the synchronization, routing, and application traffic

Input : $NodeID = i$, N_{AP}, A, p, transmission x
Output: Time slot assignments for three slotframes
if *x belongs to synchronization traffic* **then**
  Assign ith slot in the synchronization slotframe to send or receive x;
end
if *x belongs to routing traffic* **then**
  Assign the first slot in the routing slotframe to send or receive x;
end
if *x belongs to application traffic* **then**
  Assign sth slot in the application slotframe to send or receive x (s is computed by Eq.5);
end

[5]A lookup table is used to map the MAC address of a node to its node id. We use one byte integer to store the node ID, which supports up to 255 devices in the network. More bytes can be used to store the node id if there are more devices.



(a) Graph routes in the example.

(b) Synchronization Schedule.

(c) Routing Schedule.

(d) Application Schedule.

(e) Combined Schedule

Fig. 6. Three schedules for different traffic and combined schedule.

Schedule Combination: After assigning time slots in those three slotframes, the node combines them to a single schedule for runtime execution. 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 priority. The slotframe for traffic with lower priority gives up its transmission in the slot in which a scheduling conflict happens during the combination and the schedule for traffic with higher priority occupies the slot. It is important to note that no traffic is constantly blocked since the three slotframes use different prime numbers as their lengths.

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

A. Scheduling Example

Figure 6 illustrates our scheduling approach. Figure 6(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 6(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 6(c) shows the routing schedule which assigns the first slot for routing and Figure 6(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 6(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[6]. 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:

$$p_c(routing) = \begin{cases} 1 - e^{-T*L/N}, & \text{if } L \geq N \\ 1 - e^{-T}, & \text{otherwise} \end{cases} \quad (6)$$

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.

Let us assume that the slotframe B has B_{len} slots and B_{slot} slots among them are scheduled for transmissions. Let $conf_{A,B}$ denote the event of a given slot in the slotframe A conflicting with any slot scheduled for transmission in the slotframe B. The probability for the slot in the slotframe A to conflict with the slotframe B is:

$$p(conf_{A,B}) = \frac{1}{B_{len}/B_{slot}} \quad (7)$$

When such a slot scheduling conflict occurs, the slotframe with higher priority takes precedence and all other slotframes give up its transmissions. So the probability of a slot in the slotframe A to be skipped due to a conflict with any other slotframe during the combination is:

$$p_{skip(A)} = 1 - (\prod_{\forall B \in SF, B_{pri} > A_{pri}} (1 - p(conf_{A,B}))) \quad (8)$$

where SF denotes the set of all slotframes in the network and B_{pri} denotes the priority of B. The multiplication of $(1 - p(conf_{A,B}))$ denotes the probability of lower priority traffic A without any conflict when it is combined with higher priority traffic. Based on the multiplication of $(1-p(conf_{A,B}))$, we can calculate the $p_{skip(A)}$. As reported in Orchestra, the probability of an application or routing slotframe to be skipped is expected to be low in practice since the synchronization period determined by the hardware clock drift is much longer

[6]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.

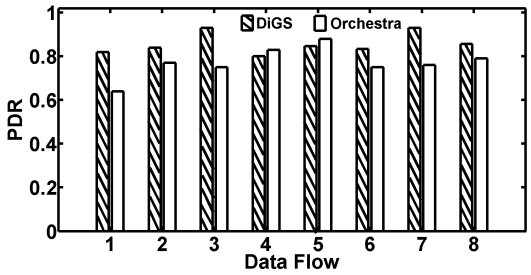


Fig. 7. Average PDR of eight data flows.

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. Conflict Deferral Scheduling

DiGS assigns different priorities to different types of traffic and forces the traffic with lower priority to give up its transmission when any scheduling conflict happens during the combination of individual schedules. This design simplifies the scheduling process at the cost of a potential risk of performance deterioration caused by the scheduling conflicts (as the probability analysis in Section VI-B shows).

The potential risk of performance deterioration is not acceptable to some mission-critical industrial applications, especially when operating in the presence of external interference and node failure. We perform an empirical study using the Cooja simulator to investigate the effect of scheduling conflicts in a worst-case scenario. We place 50 TelosB motes in a $150mX150m$ area in Cooja [46] based on the deployment of Testbed A and randomly deploy 4 jammers to generate inconstant interference, causing frequent routing updates in the network. We conduct two sets of experiments: one set under DiGS and the other under Orchestra. In each set, we run experiments with 300 flow sets, each of which contains 8 data flows that have different sources and destinations. Figure 7 shows the average PDRs of 8 data flows when the network runs DiGS and Orchestra. 87.5% and 100% of data flows have average PDRs lower than 90% under DiGS and Orchestra, respectively. While Figure 7 shows promising results (DiGS significantly outperforming Orchestra), it also highlights the significant performance deterioration caused by the scheduling conflicts.

To address this issue, we enhance DiGS (reported in our conference paper [47]) by introducing a conflict deferral scheduling policy that defers the traffic with lower priority at conflict instead of forcing it to give up its slot. The enhanced DiGS is named DiGS-CD, where "CD" stands for conflict deferral. The design goal of DiGS-CD is to equip the network devices with the capability of detecting scheduling conflicts and deriving available future time slots for conflicted traffic rapidly and autonomously without performing any handshake. DiGS-CD completely eliminates the three different conflicts (routing traffic blocked by synchronization traffic, application traffic blocked by synchronization traffic, and application traffic blocked by routing traffic) at the cost of slightly increasing the latency. A node running DiGS-CD computes the available future time slot for a deferred transmission completely based on its local information such as its node id ($NodeID$), the number of nodes in the network (N_{node}), the number of access points (N_{AP}), the slotframe length of synchronization, routing and application (L_s, L_r, L_a), and ASN. Those parameters (N_{node}, N_{AP}, L_s, L_r) are global information shared by all nodes in the network and can be carried by the EBs.

A naive way to defer conflicts is to generate three individual schedules first and then detect the conflicts during combination. However, this method introduces significant computation overhead and memory usage, since each node has to detect conflicts from the first slot to the least common multiple (LCM) of three slotframe lengths (L_s, L_r, and L_a) in order to detect all potential conflicts and then defer the lower priority traffic. Each node also has to store the ASN and the deferral offset after capturing a conflict. To reduce the overhead, we develop a fast proactive conflict detection and offset calculation algorithm that automatically captures conflicts and computes offsets when generating the individual schedules.

Algorithm 3: Conflict Detection and Offset Calculation Algorithm

```
//Table II shows the notations
Input  : N_node, ASN_R, ASN_A, ASN_Af, L_s, L_r, Pivot
Output: SO_sync_routing, SO_sync_app, SO_routing_app
SO_sync_routing = SO_sync_app = SO_routing_app = 0;
if 1 ≤ ASN_R % L_s && ASN_R % L_s ≤ N_node then
    SO_sync_routing = N_node − ASN_R % L_s + 1;
    //A Type 1 conflict detected
end
if ASN_Af % L_s ≤ N_node || ASN_Af % L_s >
  L_s − 3 ∗ (N_node − N_AP) + 1 then
    SO_sync_app = N_node − Pivot % L_s + 1;
    //A Type 2 conflict detected
end
if ASN_A % L_r == SO_sync_routing + 1 then
    SO_routing_app = 1;
    //A Type 3 conflict detected
end
```

We define three types of slot usage conflicts: the conflict between routing traffic and synchronization traffic (Type 1), the conflict between application traffic and synchronization traffic (Type 2), and the conflict between application traffic and routing traffic (Type 3). Since the synchronization traffic has the highest priority, the routing and application traffic needs to yield to it when conflicting. Algorithm 3 presents our conflict detection and offset calculation algorithm and Table II shows the notations used in the algorithm. When a node is generating the routing schedule at runtime, it checks whether there is any (Type 1) conflict between routing traffic and synchronization traffic. A Type 1 conflict occurs when the following condition is satisfied:

$$1 \leq ASN_R \ \% \ L_s \leq N_{node} \tag{9}$$

All nodes in the network transmit synchronization beacons using the first N_{node} slots in each synchronization slotframe (see Section VI). When Condition 9 is satisfied, the routing

TABLE II
NOTATIONS USED IN ALGORITHM 3.

Symbol	Description
ASN_R	ASN of the slot scheduled for routing in the following slotframe
ASN_{A_i}	ASN of the slot scheduled for application in the following slotframe
ASN_{A_f}	ASN of the first slot scheduled for application in the following slotframe
L_s	Length of synchronization traffic slotframe
L_r	Length of routing traffic slotframe
N_{node}	Number of nodes in the network
$Pivot$	ASN of the first conflicted slot between application and synchronization
$SO_{sync\_routing}$	Shift offset for routing traffic yielding to synchronization traffic
SO_{sync_app}	Shift offset for application traffic yielding to synchronization traffic
$SO_{routing\_app}$	Shift Offset for application traffic yielding to routing traffic

traffic uses a slot conflicting with the synchronization traffic. After capturing a Type 1 conflict, the node computes the shift offset ($SO_{sync\_routing}$) as below:

$$SO_{sync\_routing} = N_{node} - ASN_R \ \% \ L_s + 1 \tag{10}$$

The routing traffic is then deferred by $SO_{sync\_routing}$ slots, outside the block of slots ($[1, N_{node}]$) scheduled for the synchronization traffic. It also guarantees that the sender and receiver will both move their schedule with the same offset to prevent conflicts. Similarly, a Type 2 conflict occurs when one of the following conditions is satisfied:

$$ASN_{A_f} \ \% \ L_s \leq N_{node} \tag{11}$$

or

$$ASN_{A_f} \ \% \ L_s > L_s - 3 * (N_{node} - N_{AP}) + 1 \tag{12}$$

When Condition 11 or 12 is satisfied, the application traffic uses a slot conflicting with the synchronization traffic. Please note that the access points do not transmit any application packet, so we deduct N_{AP} in Condition 12. After capturing a Type 2 conflict, the node computes the SO_{sync_app} based on the ASN of the first conflicted slot ($Pivot$) and the number of nodes in the network:

$$SO_{sync\_app} = N_{node} - Pivot \ \% \ L_s + 1 \tag{13}$$

The application traffic is then deferred by SO_{sync_app} slots, outside the block of slots ($[1, N_{node}]$) scheduled for the synchronization traffic. A Type 3 conflict occurs when the following condition is satisfied:

$$ASN_{A_i} \ \% \ L_r = SO_{syn\_routing} + 1 \tag{14}$$

Since the routing traffic may have already been deferred by the synchronization traffic, we add $SO_{syn\_routing}$ to Condition 14. It should be noted that ASN_{A_i} denotes one of the time slot i for the application traffic in the following slotframe which collides with routing traffic. After capturing a Type 3 conflict, the node defers the application traffic by one slot. With $SO_{sync\_routing}$, SO_{sync_app}, and $SO_{routing\_app}$, each node continues to assign slots.

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 $1 + SO_{sync\_routing}$ for the routing traffic.

Assigning Slots for Application: SO_{sync_app} and $SO_{routing\_app}$ are added into Eq. 5. Under DiGS-CD, a node uses the sth time slot in the application slotframe for the pth transmission attempt:

$$s = A * (NodeID - N_{AP}) - A + p + SO_{sync\_app} + SO_{routing\_app} \tag{15}$$

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

Schedule Combination: After assigning time slots in those three slotframes, the node combines them to a single conflict-free schedule for runtime execution.

A. Scheduling Example

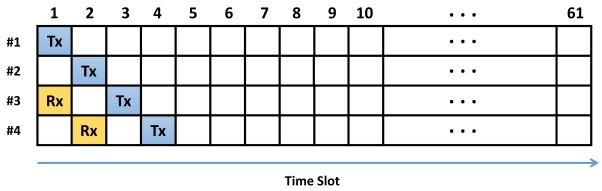
We again use the network presented in Figure 6(a) as an example. The periods of the synchronization, routing, and application schedules (slotframe length) are assumed to be 61, 11, and 12 time slots, respectively. The combined schedule has $61*11*12 = 8052$ time slots in total. Figure 8(a) shows the first four slots scheduled for the synchronization traffic. A Type 1 conflict is detected (according to Condition 9), since $1 \ \% \ 61$ is within the range of $[1, 4]$. So $SO_{sync\_routing} = 4 - 1 \ \% \ 61 + 1 = 4$. As Figure 8(b) showed, all four node defer their routing schedule from the first slot to the fifth slot to resolve the Type 1 conflict. In the next routing cycle, there is no conflict in the 12th slot, so the routing traffic is still scheduled on the 12th slot. Similarly, a Type 2 conflict is detected (according to Condition 11), since $1 \ \% \ 61$ is within the range of $[1, 4]$ and $Pivot = 1$. So $SO_{sync\_app} = 4 - 1 \ \% \ 61 + 1 = 4$. A Type 3 conflict is also detected (according to Condition 14), So $SO_{routing\_app} = 1$. All four nodes defer their application traffic by ($SO_{sync\_app} + SO_{routing\_app} = 5$) slots. Figure 8(c) shows the application schedule and Figure 8(d) shows the combined conflict-free schedule.

B. Delay Analysis

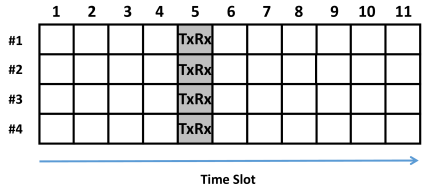
Under DiGS-CD scheduling approach, each slotframe repeats at a constant period and defers its transmission whenever it detects a scheduling conflict with a slotframe with higher priority. Here, for simplicity, we assume that the packet can be delivered from the source to the destination within a single application slotframe. The synchronization and routing traffic may block the application traffic. The maximum increase on the end-to-end latency caused by the synchronization traffic (D_{sync_app}) is:

$$D_{sync\_app} = T_{slot} * N_{node} \tag{16}$$

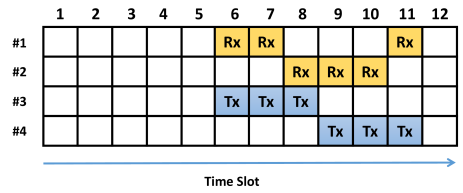
where T_{slot} denotes the time duration of one time slot and N_{node} denotes the number of nodes in the network. The



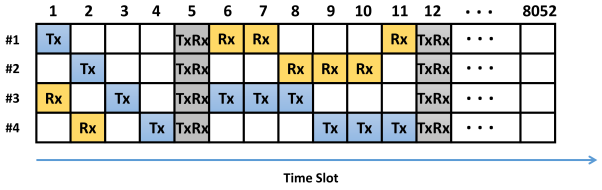
(a) Synchronization Schedule.



(b) Routing Schedule.



(c) Application Schedule.



(d) Combined Schedule

Fig. 8. Three schedules for different traffic and combined schedule.

maximum increase on the end-to-end latency caused by the routing traffic ($D_{routing\_app}$) is:

$$D_{routing\_app} = T_{slot} * \lfloor \frac{3N_{node}}{L_r} \rfloor \quad (17)$$

where Lr is the length of the routing slotframe. So the maximum increase on the end-to-end latency ($L_{DiGS-CD}$) is:

$$L_{DiGS-CD} = L_{DiGS} + T_{slot} * (N_{node} + \lfloor \frac{3N_{node}}{L_r} \rfloor) \quad (18)$$

where L_{DiGS} and $L_{DiGS-CD}$ denote the end-to-end latency under DiGS and DiGS-CD, respectively.

DiGS-CD increases the end-to-end latency by $[0, T_{slot} * (N_{node} + \lfloor \frac{3N_{node}}{L_r} \rfloor)]$ compared to DiGS. Please note that the synchronization slotframe period is much longer than the application slotframe period in practice. Thus, the probability of introducing significant latency increases is low. The average latency differences between DiGS and DiGS-CD which we observed in our experiments range between 3ms and 33ms (see Section VIII-D).

VIII. Evaluation

We have implemented our solution (DiGS) in Contiki [48], 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 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 [49]; and (2) *Testbed B* featuring 44 TelosB motes spanning two floors of a building [50]. 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.

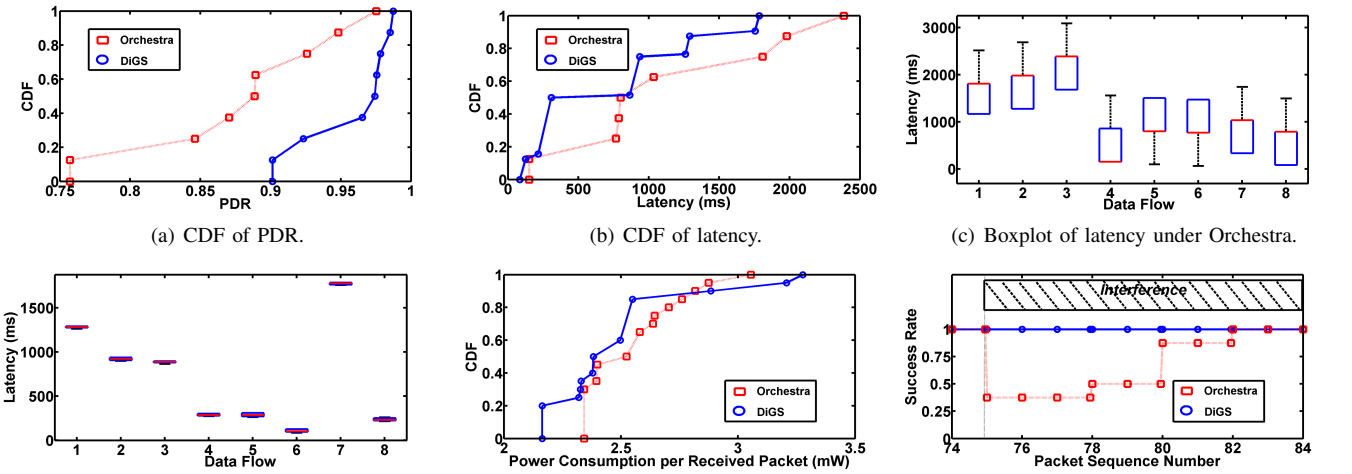
We observe that Orchestra significantly outperforms WirelessHART under network dynamics (see Section IV), therefore compare our solution against Orchestra[7] instead of WirelessHART and examine their performance under two scenarios: one under interference (Section VIII-A) and the other with node failure (Section VIII-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 VIII-C), examine the effectiveness of conflict deferral scheduling (Section VIII-D), and perform a simulation study with 150 nodes in the Cooja simulator [46] (Section VIII-E).

A. Performance under Interference

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 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

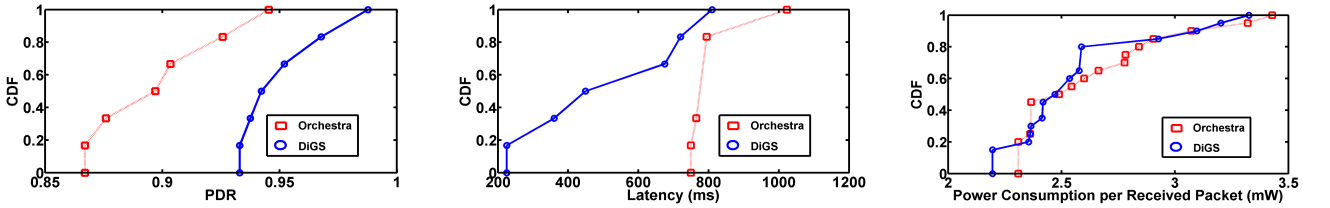
[7]We use the Orchestra implementation in Contiki provided by the authors in [10].



(a) CDF of PDR. (b) CDF of latency. (c) Boxplot of latency under Orchestra.

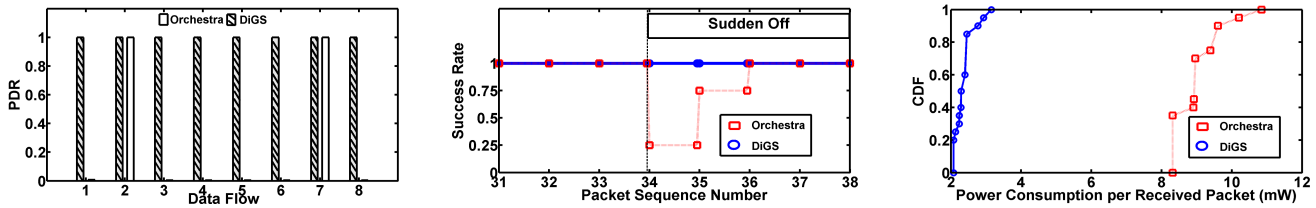
(d) Boxplot of latency under DiGS. (e) CDF of power consumption per received packet (f) A micro-benchmark measurement on the packet delivery success rate among 8 data flows.

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



(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.

per received packet under DiGS and Orchestra[8]. 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.

[8]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 [51]

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 randomly turning off four nodes located on the routing graph one by one. We repeat the experiments for 34 times with different set of failing nodes. Figure 11 shows the performance comparison between DiGS and Orchestra when the network encounters node failure on Testbed A. As Figure 11(a) showed,

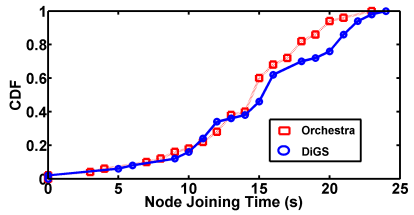
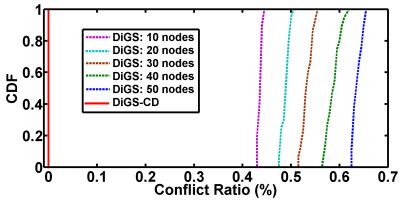
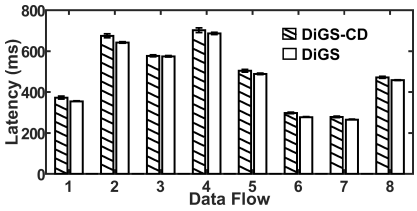


Fig. 12. Network initialization time comparison between DiGS and Orchestra.



(a) CDF of scheduling conflict ratio.



(b) Latency of each data flow with 50 nodes.

Fig. 13. Performance comparison between DiGS-CD and DiGS when the network has different number of nodes.

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 Orchestra. As Figure 11 shows, 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 12 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. Conflict Deferral Scheduling

To examine the effectiveness of conflict deferral scheduling, we run three sets of experiments to compare DiGS-CD against DiGS under various operating conditions. We first evaluate the performance of DiGS-CD and DiGS when operating in a network with different size. Figure 13(a) shows the CDF of scheduling conflict ratio of DiGS-CD and DiGS when the network consists of 10, 20, 30, 40 and 50 nodes. The scheduling conflict ratio under DiGS increases with the increasing number of nodes in the network, while the ratio under DiGS-CD is always zero. The median conflict ratios under DiGS are 0.44%, 0.49%, 0.54%, 0.59%, and 0.64% when the network has 10, 20, 30, 40, and 50 nodes. Figure 13(b) shows the histogram of average latency with 95% confidence interval. DiGS-CD increases the average latency by $[3, 33]$ms among eight data flows, each of which delivers 300 packets. The confidence interval half-width values are 6.79, 9.62, 5.09, 11.30, 6.79, 3.39, 3.42, and 4.87 under DiGS-CD, and 1.70, 3.39, 3.85, 5.09, 3.67, 1.70, 2.15, and 2.42 under DiGS, respectively. The conflict deferral scheduling successfully mitigates the scheduling conflict at the cost of slightly increasing the latency.

We then investigate the effect of interference on the scheduling conflict ratio of DiGS-CD and DiGS. Figure 14 shows the CDF of scheduling conflict ratio of a 50-node network when 1∼3 jammers exist in the network. The median conflict ratio under DiGS increases from 1.01% to 1.39%, and then to 1.96% when more jammers are introduced into the network. The increase on the conflict ratio is caused by the more frequent routing updates which aggravate the conflicts between routing traffic and application traffic. DiGS-CD always provides conflict-free schedules.

Finally, we evaluate DiGS-CD and DiGS under different data rates. Figure 15(a) shows the CDF of scheduling conflict ratio of a 50-node network when the source nodes generate packets with different intervals. The scheduling conflict ratio under DiGS increases slowly when the data rates increase (the packet interval decreases from 20s to 5s). The median conflict ratios under DiGS are 0.59%, 0.62%, 0.63% and 0.64% when the data generation intervals are 20s, 15s, 10s, and 5s, respectively. This is because the conflicts mainly depend on the frequency of routing updates. Figure 15(b) shows the histogram of average latency with 95% confidence interval. DiGS-CD increases the average latency by $[7, 24]$ms among eight data flows, each of which delivers 300 packets. The confidence interval half-width values are 3.96, 4.52, 8.09, 8.92, 10.05, 5.87, 6.73, and 5.92 under DiGS-CD, and 3.32, 4.11, 4.88, 4.76, 34.42, 5.67, 4.12, and 3.13 under DiGS, respectively. From the above results, we conclude that DiGS-CD is more suitable to operate in dynamic and noisy environments, while DiGS is a better choice for clean environments.

E. 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 $300mX300m$ 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.

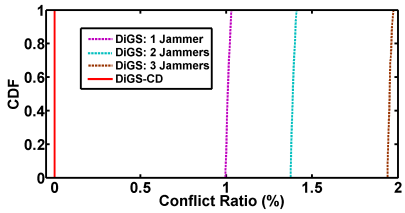
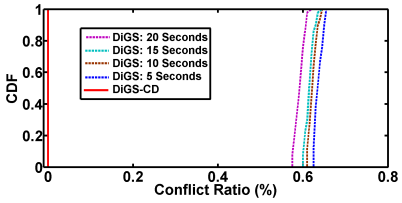
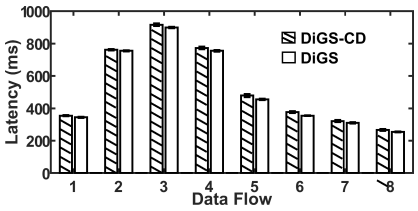


Fig. 14. CDF of scheduling conflict ratio under DiGS-CD and DiGS with different number of jammers. The network consists of 50 nodes.

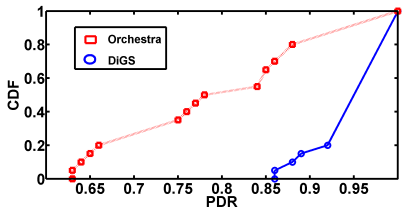


(a) CDF of scheduling conflict ratio.

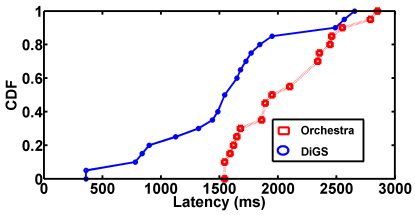


(b) Latency of each data flow with a packet per ten seconds.

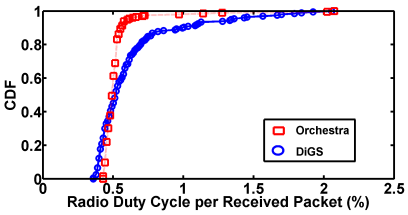
Fig. 15. Performance comparison between DiGS-CD and DiGS when the data packets are generated at different rates. The network consists of 50 nodes.



(a) CDF of PDR.



(b) CDF of latency.



(c) CDF of radio duty cycle per received packet.

Fig. 16. Simulation with 150 nodes in Cooja Simulator.

TABLE III
COMPARISON OF DIGS/DIGS-CD, WIRELESSHART AND ORCHESTRA.

Method Name	Centralized	Distributed	Graph Routing
DiGS/DiGS-CD	×	√	√
WirelessHART	√	×	√
Orchestra	×	√	×

Figure 16 shows the performance under DiGS and Orchestra when the network encounters interference. Figure 16(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 16(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 16(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.

IX. CONCLUSIONS

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

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] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis, "Collection Tree Protocol," in *Sensys*, 2009.
[8] 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.
[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] WCPS Simulator. [Online]. Available: http://wsn.cse.wustl.edu/index.php/WCPS:_Wireless_Cyber-Physical_Simulator
[41] Orchestra. [Online]. Available: https://github.com/contiki-os/contiki/tree/master/apps/orchestra
[42] TelosB: Telosb Mote Platform, Datasheet Provided by MEMSIC Inc. [Online]. Available: http://www.memsic.com/userfiles/files/Datasheets/WSN/telosb%5fdatasheet.pdf
[43] J. W. S. Liu, "Real-time systems," 2000.
[44] C. A. Boano, T. Voigt, C. Noda, K. Romer, and M. Zuniga, "JamLab: Augmenting sensornet testbeds with realistic and controlled interference generation," in *IPSN*, 2011.
[45] P. Levis, T. Clausen, J. Hui, O. Gnawali, and J. Ko, "RFC 6206," in *The Trickle Algorithm*, 2011.
[46] "Cooja Simulator." [Online]. Available: http://anrg.usc.edu/contiki/index.php/Cooja_Simulator
[47] J. Shi, M. Sha, and Z. Yang, "DiGS: Distributed Graph Routing and Scheduling for Industrial Wireless Sensor-Actuator Networks," in *ICDCS*, 2018.
[48] Contiki: The Open Source OS for the Internet of Things. [Online]. Available: http://www.contiki-os.org/
[49] "Testbed at the State University of New York at Binghamton." [Online]. Available: http://www.cs.binghamton.edu/\%7emsha/testbed
[50] "Testbed at the Washington University in St. Louis." [Online]. Available: http://cps.cse.wustl.edu/index.php/Testbed
[51] 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

Junyang Shi is a PhD student in the Department of Computer Science at the State University of New York at Binghamton. He received her B.S. degree in Electrical and Electronic Engineering from the Huazhong University of Science and Technology in 2016. His research focuses on industrial wireless sensor-actuator networks and Internet of Things.

Mo Sha is an Assistant Professor in the Department of Computer Science at the State University of New York at Binghamton. He received his Ph.D. degree in Computer Science from Washington University in St. Louis in 2014. Prior to his PhD, he received a M.Phil. degree from City University of Hong Kong in 2009 and a B.Eng. degree from Beihang University in 2007. His research interests include wireless networks, Internet of Things, embedded and real-time systems, and Cyber-Physical Systems.

Zhicheng Yang is a PhD student in the Department of Computer Science at the University of California, Davis. He received his M.S. degree in Computer Science from the Washington University in St. Louis in 2012. Prior to his Master degree, he received her B.S. Degree from Beijing University of Posts and Telecommunications in 2010. His research interests include wireless networking and sensing, 60 GHz WLANs, and mobile computing.