2018 IEEE Real-Time and Embedded Technology and Applications Symposium

FD-PaS: A Fully Distributed Packet Scheduling Framework for Handling Disturbances in Real-Time Wireless Networks

Tianyu Zhang[†§*], Tao Gong[‡], Zelin Yun[‡], Song Han[‡], Qingxu Deng[†], Xiaobo Sharon Hu[§]
[†]School of Computer Science and Engineering, Northeastern University, Shenyang, China
[†]Email: tyzhang@stumail.neu.edu.cn, dengqx@mail.neu.edu.cn
[‡]Dept. of Computer Science and Engineering, University of Connecticut, Storrs, CT, 06269
[‡]Email: {tao.gong, zelin.yun, song.han}@uconn.edu
[§]Dept. of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN, 46556
[§]Email: shu@nd.edu

***Abstract*—Along with the rapid growth of Industrial Internet-of-Things (IIoT) applications and their penetration into many industry sectors, real-time wireless networks (RTWNs) have been playing a more critical role in providing real-time, reliable and secure communication services for such applications. A key challenge in RTWN management is how to ensure real-time Quality of Services (QoS) especially in the presence of unexpected external and internal disturbances. Most prior work takes a centralized approach for handling disturbances, which is slow and subject to single-point failure, and does not scale. To overcome these drawbacks, this paper presents a fully distributed packet scheduling framework called FD-PaS. FD-PaS aims to provide guaranteed fast response to unexpected disturbances while dropping a minimum number of packets for meeting the deadlines of all critical tasks. To combat the scalability challenge, FD-PaS incorporates several key advances in both algorithm design and data link layer protocol design to enable individual nodes to make on-line decisions locally without any centralized control. Our extensive simulation and testbed results have validated the correctness of the FD-PaS design and demonstrated its effectiveness in providing fast response for handling disturbances.**

I. Introduction

Real-time wireless networks (RTWNs) are fundamental to many Industrial Internet-of-Things (IIoT) applications in a broad range of fields such as military, civil infrastructure and industrial automation [1]–[3]. These applications have stringent real-time constraints on RTWNs to ensure timely collection of environmental data and proper delivery of control decisions. The Quality of Service (QoS) offered by a RTWN is thus often measured by how well it satisfies the end-to-end (from sensors via controllers to actuators) deadlines of the real-time tasks executed in the RTWN. Packet scheduling in RTWNs plays a critical role in achieving the desired QoS. Though packet scheduling in RTWNs has been studied for a long time, the explosive growth of IIoT applications especially in terms of their scale and complexity has dramatically increased the level of difficulty in tackling this inherently challenging undertaking. The fact that most RTWNs must deal with unexpected disturbances further aggravate the problem.

Unexpected disturbances can be classified into *external* disturbances from the environment being monitored and controlled (*e.g.*, detection of an emergency, sudden pressure or temperature changes) and *internal* disturbances within the network infrastructure (*e.g.*, link failure due to multi-user interference or weather related changes in channel signal to noise ratio (SNR)). The focus of this paper is to develop an effective and distributed packet scheduling solution that can be readily scaled to extremely large RTWNs and can satisfactorily handle unexpected *external disturbances* to meet real-time constraints. In the rest of the paper, we simply refer to external disturbance as disturbance.

The challenge of handling disturbances in RTWNs comes from the unpredictability of disturbance occurrence at run time. Specifically, it is generally unknown when/which disturbance occurs and what is the current network status at that point (*e.g.*, how many packets have been delivered to their destinations). Since it is computationally infeasible to enumerate all possibilities before network starts, people resort to using on-line dynamic scheduling approaches to react to unexpected workload changes incurred by disturbances (*e.g.*, [4]–[9]). The most recent dynamic method appears in [10] in which authors propose a distributed dynamic packet scheduling framework, referred to as D^2-PaS, to handle disturbances in RTWNs. The main idea of D^2-PaS is to rely on a centralized control point in the network, *e.g.* the gateway, to generate a dynamic schedule when a disturbance occurs and propagate a minimum amount of necessary information to the network. After all nodes receive such information, they are able to generate a consistent dynamic schedule in a distributed manner to start handling the disturbance. Though D^2-PaS can achieve good performance in terms of minimizing the number of dropped packets to guarantee real-time deadlines of critical packets when disturbance occurs, it suffers from limited scalability since a centralized control point is required. It becomes a significant limitation as RTWNs start to be deployed over large geographic area (*e.g.*, thousands of devices over an oil field). Moreover, D^2-PaS may need an extremely long response time to handle a disturbance especially for large-scale networks since the system cannot start to handle the disturbance until the dynamic schedule is disseminated to the whole network via broadcast packets. All these drawbacks lead to degraded system performance when disturbances occur.

In this work, we introduce a fully distributed packet scheduling framework, referred to as FD-PaS, to handle disturbances in RTWNs. Same as D^2-PaS, FD-PaS lets each node construct its own schedule using the technique in [10]. However, FD-PaS makes on-line decisions locally without *any* centralized control point (*e.g.*, the gateway in D^2-PaS) when disturbances occur. This is achieved by sending the disturbance

*The first two authors have equal contribution to this work.

978-1-5386-5295-4/18/$31.00 ©2018 IEEE
DOI 10.1109/RTAS.2018.00007

information only to a subset of all nodes via the routing paths of the tasks running in the network. In such a manner, a broadcast task is no longer needed in FD-PaS for notifying all nodes about the disturbance information, which significantly reduces the response time to handle the disturbance. To ensure this partial disturbance propagation scheme works properly, we need to overcome several challenges. For example, to avoid transmission collision among different nodes with inconsistent schedules, we propose a multi-priority wireless packet preemption mechanism called MP-MAC in the data link layer to ensure that high-priority packets can always be delivered by preempting the transmissions of low-priority packets. Further, to minimize the number of dropped packets, we formulate a packet dropping problem to determine a temporary dynamic schedule for individual nodes to handle the disturbance. We prove that the packet dropping problem is NP-hard, and introduce an efficient heuristic to be executed by individual nodes locally. Both the MP-MAC design and the dynamic schedule construction method (they jointly comprise the FD-PaS framework) are implemented on our RTWN testbed. Our extensive performance evaluation validates the correctness of the FD-PaS design and demonstrates its effectiveness in providing fast response for handling disturbances.

The remainder of this paper is organized as follows. Section II describes the system model. Section III presents a motivating example and gives an overview of the FD-PaS framework. We discuss how to propagate disturbances and avoid transmission collisions in Section IV and V, respectively. Section VI introduces the packet dropping problem and presents an efficient heuristic. Performance evaluation are summarized in Section VII. We conclude the paper and discuss future work in Section VIII.

II. System Model

We adopt the system architecture of a typical RTWN, in which multiple sensors and actuators are wirelessly connected to a controller node directly or through relay nodes. (Note that the controller node is for initial network setup and performing control computations. FD-PaS does not need it for making any on-line decision and updating schedules.) We refer to non-controller nodes as device nodes. We assume that all device nodes have routing capability and are equipped with a single omni-directional antenna to operate on a single channel in half-duplex mode. The network is modeled as a directed graph $G = (V, E)$, where the node set $V = \{\{V_0, V_1, \dots\}, V_c\}$ and V_c represents the controller node. A direct link $(V_i, V_j) \in E$ represents a reliable link from node V_i to node V_j.

We use the concept of task to describe packet transmission from sensor nodes to actuator nodes. Specifically, the system runs a fixed set of unicast tasks $\mathcal{T} = \{\tau_0, \tau_1, \dots, \tau_n\}$. Each task τ_i $(0 \le i \le n)$ follows a designated single routing path with H_i hops. It periodically generates a packet which originates at a sensor node, passes through the controller node (not necessary for FD-PaS but to carry out control computations) and delivers a control message to an actuator.

When external disturbances (*e.g.*, sudden change in temperature or pressure) occur, many IIoT applications would require more frequent sampling and control actions, which in turn increase network resource demands. To capture such abrupt

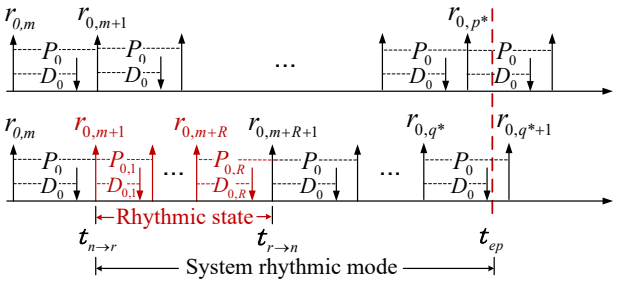

Fig. 1: Timing parameters of the rhythmic task τ_0 in the system rhythmic mode. Top and bottom subfigures denote the nominal and actual release times and deadlines of τ_0 respectively.

increase in network resource demands, we adopt the rhythmic task model [11] which has been shown to be effective for handling disturbances in event-triggered control systems [8]. (Note that our FD-PaS framework is not limited to the rhythmic task model and is applicable to any task model that provides workload changing patterns for handling disturbances.) In the rhythmic task model, each unicast task τ_i has two states: *nominal state* and *rhythmic state*. In the nominal state, τ_i follows nominal period P_i and nominal relative deadline $D_i (\le P_i)$, which are all constants. When a disturbance occurs, τ_i enters the rhythmic state in which its period and relative deadline are first reduced in order to respond to the disturbance, and then gradually return to their nominal values by following some monotonically non-decreasing pattern. We use vectors $\overrightarrow{P_i} = [P_{i,x}, x = 1, \dots, R]^T$ and $\overrightarrow{D_i} = [D_{i,x}, x = 1, \dots, R]^T$ to represent the periods and relative deadlines of τ_i when it is in the rhythmic state. As soon as τ_i enters the rhythmic state, its period and relative deadline adopt sequentially the values specified by $\overrightarrow{P_i}$ and $\overrightarrow{D_i}$, respectively. τ_i returns to the nominal state when it starts using P_i and D_i again.

Here we assume that at most one task can be in the rhythmic state at any time during the network operation. To simplify the notation, we refer to any task currently in the rhythmic state as *rhythmic task* and denote it as τ_0 while task τ_i $(1 \le i \le n)$ is a *periodic task* which is currently not in the rhythmic state. As shown in Fig. 1, when τ_0 enters the rhythmic state, we also say that the system switches to the *rhythmic mode*. The system returns to the *nominal mode* when the disturbance has been completely handled, typically some time after τ_0 returns to the nominal state. Since disturbances may cause catastrophe to the system, the rhythmic task has a hard deadline when the system is in the rhythmic mode while periodic tasks can tolerate occasional deadline misses.

Each task τ_i consists of an infinite sequence of instances. The k-th instance of τ_i, referred to as packet $\chi_{i,k}$, is associated with release time $r_{i,k}$, deadline $d_{i,k}$ and finish time $f_{i,k}$. Without loss of generality, we assume that τ_0 enters the rhythmic state at $r_{0,m+1}$ (denoted as $t_{n\to r}$) and returns to the nominal state at $r_{0,m+R+1}$ (denoted as $t_{r\to n}$). Thus, τ_0 stays in its rhythmic state during $[t_{n\to r}, t_{r\to n})$, and $t_{r\to n} = t_{n\to r} + \sum_{x=1}^{R} P_{0,x}$. Any packet of τ_0 released in the system rhythmic mode is referred to as a *rhythmic packet* while the packets of task τ_i $(1 \le i \le n)$ are *periodic packets*. The delivery of packet $\chi_{i,k}$ at the h-th hop is referred to as a transmission denoted as $\chi_{i,k}(h)$ $(1 \le h \le H_i)$. Following industry practice, we assume a Time Division Multiple Access

(TDMA)-based data link layer. Every node follows a given schedule to transmit or receive packets, and each transmission $\chi_{i,k}(h)$ must be completed in a single time slot.

Based on the above system model, the problem that we aim to solve in this paper is presented as follows.

Problem 1: Assume that for a given RTWN, a static schedule is provided which can deliver all packets by their respective deadlines when there are no disturbances. Upon detection of a disturbance at $r_{0,m}$ (a release time of τ_0's packet[1]), determine the dynamic schedule in the system rhythmic mode such that (i) the system can start handling rhythmic packets no later than $r_{0,m+1} = r_{0,m} + P_0$, (ii) all rhythmic packets are delivered by their deadlines, (iii) minimum number of periodic packets are dropped in the system rhythmic mode, and (iv) the system can safely return to the nominal mode after which all packets meet their nominal deadlines.

Constraint (i) ensures that disturbances can be handled in the earliest possible time (*i.e.*, before the nominal arrival time of the next packet). If Constraint (i) was violated, the corresponding control system could become unstable or suffer from severe performance degradation. The meaning of Constraints (ii), (iii) and (iv) are self explanatory.

III. Overall Framework of FD-PaS

In this section, we first give a motivating example to show the deficiency of centralized packet scheduling approaches for handling rhythmic tasks. We then present an overview of the fully distributed packet scheduling framework, FD-PaS.

A. Drawbacks of Centralized Approaches

In order to properly handle unexpected external disturbances, centralized dynamic scheduling approaches have been proposed, which can adapt to changes in on-line network resource demand. For example, OLS [8] can generate an on-line dynamic schedule based on a dynamic programming approach to handling disturbances modeled as rhythmic events. To further improve the performance in terms of reducing the number of dropped packets, D^2-PaS [10] leverages the network-wide synchronization in RTWNs and the computing capability of individual device nodes to generate consistent schedules locally. By effectively reducing the amount of schedule related information to be broadcast by the gateway, D^2-PaS significantly improves the scalability of the dynamic schedule construction and dissemination processes.

Centralized approaches, however, incur long latency for handling disturbances, especially in large networks. Consider an RTWN (Fig. 2) with 3 tasks (τ_0, τ_1 and τ_2) running on 7 nodes ($V_0, \ldots, V_5$ and V_c) with V_0 and V_2 being sensors, V_4 and V_5 being actuators, V_1 and V_3 being relay nodes, and V_c being the controller node and functioning as the gateway in centralized approaches). Note that since centralized approaches rely on the controller node to disseminate the dynamic schedule, broadcast task τ_3 is needed. The tasks' routing paths, periods and relative deadlines are given in Table I.

Assume all tasks are synchronized and first released at time slot 0, and each node employs an EDF scheduler to construct its local schedule (see Fig. 3). Suppose at time slot 9, an external disturbance is detected and sensor V_2 sends a rhythmic event request via τ_0 to the controller node. V_c then determines the time slot $t_{n\to r}$ when τ_0 is going to enter its rhythmic state. In order to achieve fast response to the disturbance, $t_{n\to r}$ should be set to be as early as possible, but later than the time slot when all nodes in the network receive the dynamic schedule. In this example, V_c has to wait till time slot 26 to broadcast the constructed dynamic schedule. Only after the broadcast packet reaches all nodes at 30, τ_0 can enter its rhythmic state at the nearest release time slot 36. Therefore, for this example, though the disturbance is detected by the sensor at time slot 9, the system cannot enter the rhythmic mode until slot 36, which is three nominal periods later (instead of one nominal period as required in **Problem 1**).

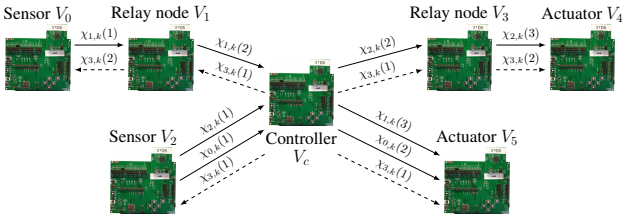

Fig. 2: An example RTWN with three unicast tasks and one broadcast task running on 7 nodes.

TABLE I: Task parameters for the motivational example

Task	Routing Path	P_i (= D_i)
τ_0	$V_2 \to V_c \to V_5$	9
τ_1	$V_0 \to V_1 \to V_c \to V_5$	9
τ_2	$V_2 \to V_c \to V_3 \to V_4$	10
τ_3	$V_c \to *$	18

From the above example, one can readily see that the centralized approaches suffer from a considerably long response time to the disturbances especially for large RTWNs. Moreover, centralized approaches rely on a single point (the gateway) in the network to make on-line packet scheduling decisions. These are the two main roadblocks in scaling up the packet scheduling framework to support handling disturbances in large-scale RTWNs.

B. Overview of FD-PaS

In order to achieve fast response to disturbances in RTWNs, in this work we propose a fully distributed packet scheduling framework, referred to as FD-PaS. The key idea of FD-PaS is to make dynamic, local schedule adaptation at each node along the path of the rhythmic task while avoiding transmission collisions from other nodes that still follow their static schedules in the system rhythmic mode.

Fig. 4 gives an overview of the execution model of FD-PaS. After network initialization, each node generates locally a static schedule, S, using the local schedule generation mechanism in D^2-PaS and follows S to transmit packets. When a disturbance is detected by rhythmic task τ_0 at $t' = r_{0,m}$, a notification is propagated to all the nodes responsible for handling the disturbance. Let these nodes be $V_j \in \mathbf{V}_{rhy}$. Upon receiving the notification, each node in $\mathbf{V}_{rhy}$ determines the time duration of the network being in the rhythmic mode and generates a dynamic schedule $\tilde{S}$ for handling the disturbance.

[1]We assume that disturbances can be detected only at the time when the sensor samples the environment data, *i.e.*, the release time of a certain packet.

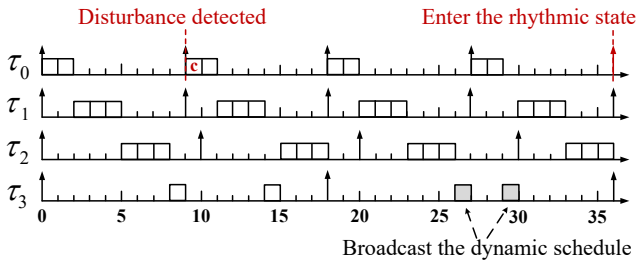

Fig. 3: Local EDF schedules of the tasks in the motivating example. The block with symbol c denotes the transmission of the rhythmic event request. The shaded blocks denote the two transmissions of the broadcast task to propagate the dynamic schedule generated at the controller node to the whole network.

Starting from $r_{0,m+1}$, one nominal period of τ_0 after detecting the disturbance, the nodes in $\mathbf{V}_{rhy}$ follow $\tilde{S}$ while all other nodes keep using static schedule S to transmit periodic packets. Thus, by not relying on a broadcast packet to disseminate the dynamic schedule generated by a centralized point in the network, FD-PaS is able to significantly reduce the response time of reacting to disturbances. For ease of discussion, in the rest of the paper, we refer to *disturbance response time* (DRT) as the time duration from t' to the start time of system rhythmic mode and *disturbance handling latency* (DHL) as the time duration of system rhythmic mode (see Fig. 4).

To ensure that FD-PaS works properly, several challenges need to be tackled. First, when a disturbance occurs, only the sensor node that has detected it knows which task will enter the rhythmic state, while the rest of the nodes in $\mathbf{V}_{rhy}$ that are to handle the disturbance have no knowledge about this. Second, if the nodes in $\mathbf{V}_{rhy}$ follow the dynamic schedule while other nodes follow the static schedule S, transmission collisions would occur which may cause rhythmic packets to miss their deadlines. Third, to properly handle disturbances, efficient methods are needed by the nodes in $\mathbf{V}_{rhy}$ to determine a dynamic schedule in which the minimum number of periodic packets are dropped. We discuss in detail how FD-PaS tackles these challenges in the following sections.

IV. Propagating Disturbance Information

In centralized approaches, all nodes in the RTWN must know the disturbance information since a dynamic schedule must be deployed at each node. However, such a network-wide propagation mechanism does not scale and often violates constraint (i) in **Problem 1** as shown by the motivating example. To overcome this drawback, we propose to disseminate the disturbance information to only a subset of all nodes, denoted as $\mathbf{V}_{rhy}$, to minimize the DRT. This scheme requires the following three questions be answered: (1) which nodes in the network belong to $\mathbf{V}_{rhy}$, (2) how to propagate the disturbance information to nodes in $\mathbf{V}_{rhy}$, and (3) does each node in $\mathbf{V}_{rhy}$ have sufficient time to generate the dynamic schedule before the system enters the rhythmic mode? Below we present our answers to these questions.

Consider questions (1) and (2) above. Recall that when a disturbance occurs, the rhythmic task τ_0 will enter its rhythmic state following reduced periods and deadlines as specified in $\overrightarrow{P_0}$ and $\overrightarrow{D_0}$. An updated schedule is needed to accommodate the increased workload of τ_0. To ensure that each transmission

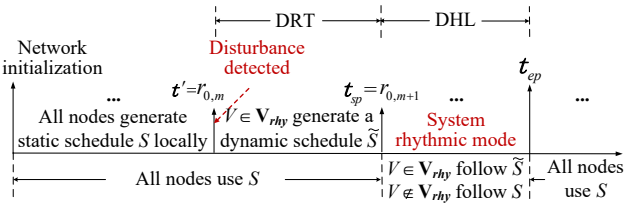

Fig. 4: Overview of the execution model of FD-PaS.

$\chi_{0,k}(h)$ can be successful, both the sender and the receiver of $\chi_{0,k}(h)$ must follow the same schedule. Thus, all nodes along the routing path of τ_0 must know the disturbance information to generate a consistent dynamic schedule, and should be included in $\mathbf{V}_{rhy}$. For example, $\mathbf{V}_{rhy} = \{V_2, V_c, V_5\}$ for the motivating example in Fig. 3 when τ_0 enters the rhythmic state. When a disturbance is detected at $r_{0,m}$, its information can be piggybacked onto $\chi_{0,m}$ and transmitted to all nodes in $\mathbf{V}_{rhy}$. Propagating disturbance information in this manner guarantees that all nodes in $\mathbf{V}_{rhy}$ receive the disturbance information within one nominal period of τ_0, *i.e.*, P_0, since the static schedule ensures that each task gets the needed number of transmissions along its routing path within P_0 in order to meet the end-to-end deadline.

Now consider question (3). As required in Constraint (i) of **Problem 1**, the system should start handling the rhythmic packets from $r_{0,m+1}$ after the disturbance is detected at $r_{0,m}$. This requires that (i) the disturbance information be successfully propagated to the relevant nodes before τ_0 enters its rhythmic state at $r_{0,m+1}$, and (ii) each node in $\mathbf{V}_{rhy}$ completes the construction of the dynamic schedule before it starts receiving/transmitting the first rhythmic packet. The propagation scheme discussed above ensures that condition (i) is met. Regarding condition (ii), our prior work showed that one idle slot (10ms) is sufficient for a typical device node in RTWNs (*e.g.*, TI CC2538 SoC) to complete its local schedule computation [10]. The theorem below establishes that such an idle slot indeed exists within the time frame specified in condition (ii).

Theorem 1 *If an RTWN system is schedulable under a given static schedule, any node V_j ($V_j \neq V_c$) in $\mathbf{V}_{rhy}$ has at least one idle slot (neither receiving nor sending any transmission) between time t_1 ($t_1 \geq r_{0,m}$) when it receives the disturbance information and time t_2 ($t_2 \geq r_{0,m+1}$) when it is involved in the transmission of the first rhythmic packet after τ_0 enters its rhythmic state at $r_{0,m+1}$.*

Proof: We first recall the following lemma from [10].

Lemma 1 *If an RTWN system is schedulable under a given static schedule,* i.e. *each packet completes all its transmissions before the deadline, for any node $V_j \neq V_c$ and task τ_i passing through V_j, there exists at least one idle slot at V_j among any three consecutive transmissions of τ_i passing V_j.*

Since in our system model, sensors and actuators are connected via the controller node, every task follows a routing path with at least two hops corresponding to two transmissions. Suppose $\chi_{0,m}(h)$ occurs at t_1 and is the transmission from

which V_j receives the disturbance information[2]. There exists at least one transmission between $\chi_{0,m}(h)$ and $\chi_{0,m+1}(h)$ (the first transmission that V_j is involved in the dynamic schedule, occurring at t_2). Then, according to Lemma 1, V_j has at least one idle slot between $\chi_{0,m}(h)$ and $\chi_{0,m+1}(h)$ (*i.e.*, between t_1 and t_2). Thus, the theorem holds. □

Based on Theorem 1 and the disturbance propagation time bound, the proposed partial disturbance propagation scheme guarantees that any disturbance can be promptly responded within one nominal period of the rhythmic task and Constraint (i) in **Problem 1** can be satisfied.

V. Avoiding Transmission Collisions

According to the disturbance propagation mechanism presented in Section IV, only the nodes on the path of the rhythmic task are included in $\mathbf{V}_{rhy}$. Nodes in $\mathbf{V}_{rhy}$ construct their local schedules individually and employ them in the system rhythmic mode. All other nodes in the network follow the static schedule. With this execution model, unless the disturbance information is propagated to the entire RTWN, inconsistencies between the dynamic and static schedules in the system rhythmic mode may easily arise, which would result in transmission collisions. To ensure that the disturbances are handled appropriately, in the FD-PaS framework, the transmissions of rhythmic packets need to be always successful even in the presence of collision with other periodic packets.

In conventional RTWNs such as WirelessHART [12] and 6TiSCH [13], TDMA-based data link layer are widely adopted to provide synchronized and collision-free channel access. In addition, most of those protocols employ the Clear Channel Assessment (CCA) operation at the beginning of each transmission for collision avoidance. CCA, however, cannot prioritize packet transmissions. When multiple transmissions happen in the same time slot sharing the same destination, it cannot guarantee the more important packets (*e.g.*, rhythmic packets) are granted the access to the channel.

To tackle this challenge, we propose an enhancement to the IEEE 802.15.4e standard [14], called Multi-Priority MAC (MP-MAC), to support prioritization of packet transmissions in RTWNs. Several attempts have been made in the literature towards supporting this feature. For example, the PriorityMAC was proposed in [15] to prioritize critical traffic in RTWNs. It introduces the concept of subslots, in which the transmitter does a very short transmission to indicate the priority of the packet to be transmitted in the following time slot. By adding two subslots before each time slot, PriorityMAC is able to create three priority levels. Different from PriorityMAC, the design of the MP-MAC aims to be lightweight and scalable. In MP-MAC, the transmitter does not explicitly conduct a short transmission to indicate the priority. Instead it implicitly indicates the priority of the transmission by adjusting the Start-Of-Frame (SOF) time offset. Compared with PriorityMAC, MP-MAC is more energy efficient (by avoiding transmissions in the subslots), and able to support more priority levels.

Fig. 5 gives a comparison of the slot timing of 802.15.4e (top) and MP-MAC (bottom). In a 802.15.4e time slot, the sender transmits a packet and the receiver responds with an acknowledgement (ACK) if the packet is successfully received[3]. The packet transmission starts at *TxOffset* after the start of the time slot, while the ACK starts at *TxAckDelay* after the completion of the packet transmission. A long Guard Time (LGT) and a short Guard Time (SGT) are used by the receiver and sender respectively to tolerate clock drift and radio/CPU operation delays. With this standard design of 802.15.4e, if multiple senders transmit packets in the same time slot, they are not aware of the other transmissions, and thus will cause interference. The slot timing of MP-MAC is presented at the bottom of Fig. 5. In MP-MAC, instead of being set as a constant, *TxOffset* is varied to implicitly indicate the priority of the packet (shown as red dashed lines). A packet with a higher priority is associated with a shorter *TxOffset* to start the transmission earlier. In addition, a CCA operation will be performed before each transmission to ensure that there is no higher priority packet transmission present in the channel. This enhancement ensures that only the highest priority packet (with the shortest *TxOffset*) is transmitted, and all lower priority transmissions yield to it.

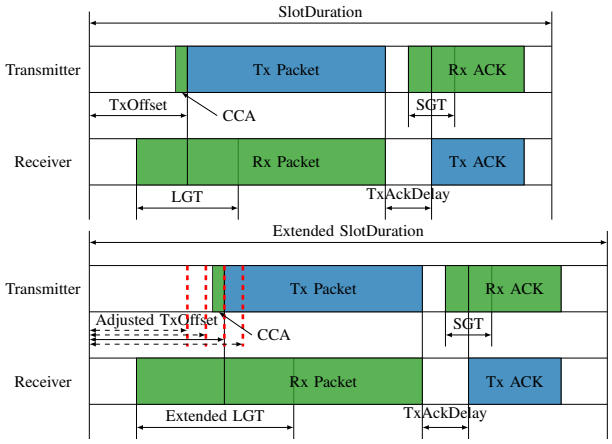

Fig. 5: Slot timing of 802.15.4e (top) and MP-MAC (bottom)

Similar to the guard times, the *TxOffset* values for different priorities need to be set sufficiently apart so that different senders and receivers have consensus on the priorities. In MP-MAC, we define *PriorityTick* as the difference between two consecutive *TxOffset*s. To support k different priorities in MP-MAC, the length of the time slot, compared to the standard design, needs be extended by $(k-1)\times PriorityTick$. A longer *PriorityTick* can ensure successful packet prioritization, but either leads to longer SlotDuration and reduced network throughput, or smaller number of supported priorities if the size of the time slot is fixed. Since *PriorityTick* is a hardware-dependent parameter, we will elaborate the selection of *PriorityTick* in our testbed experiments and demonstrate the effectiveness of MP-MAC in Section VII-B.

VI. Constructing Local Dynamic Schedule

MP-MAC ensures that once the dynamic schedules are generated locally, the nodes in $\mathbf{V}_{rhy}$ can follow those schedules to handle the disturbance without transmission collisions with other nodes in the network. Since all the nodes in

[2]If V_j is the sensor, it detects the disturbance at $r_{0,m}$.

[3]No acknowledgement is provided for broadcast and multicast packets.

$\mathbf{V}_{rhy}$ receive the same disturbance information, the dynamic schedules generated locally at these nodes are all consistent. The construction of a dynamic schedule must guarantee that 1) all rhythmic packets meet their deadlines, 2) minimum number of periodic packets are dropped, and 3) the system can reuse the static schedule after the rhythmic mode ends and all packets meet their nominal deadlines. In this section, we describe how FD-PaS generates the dynamic schedule at each node in $\mathbf{V}_{rhy}$.

A. Problem Formulation

In FD-PaS, the network starts operation by following a static schedule which guarantees that all tasks meet their nominal deadlines if no disturbance occurs. The static schedule is generated at each node locally using the local schedule generation technique proposed in [10]. We denote the static schedule as $S = \{(t, i, h)\}$, where t is the slot ID, i is the task ID and h is the hop index. For any given time slot t, we have $S[t] = (i, h)$ if t is assigned to the h-th transmission of τ_i. Otherwise, $S[t] = (-1, -1)$ to indicate an idle slot.

As shown in Fig. 1, when a disturbance is detected at $r_{0,m}$, τ_0 requires to enter its rhythmic state from the next release time $r_{0,m+1}$, *i.e.*, $t_{n\to r} = r_{0,m+1}$. Then, the system enters the rhythmic mode with an increased workload induced by τ_0. A dynamic schedule $\tilde{S}$ is thus needed before the system switches back to the nominal mode and reuses static schedule S. $\tilde{S}$ starts from $t_{n\to r}$ and ends at a carefully chosen end point t_{ep} of the system rhythmic mode. To achieve guaranteed fast disturbance handling, we further define t^u_{ep} as a user specified parameter which bounds the maximum allowed DHL, and is often application dependent. Though it is natural to use idle slots in $S[t_{n\to r}, t_{ep})$ to accommodate the increased rhythmic workload, they are not always sufficient to guarantee the deadlines of all rhythmic packets. In this case, some periodic transmissions have to be dropped. Since any node $V_j \notin \mathbf{V}_{rhy}$ keeps following the static schedule S to transmit periodic packets, periodic transmissions cannot be adjusted in the dynamic schedule[4]. Therefore if any periodic transmission $\chi_{i,k}(h)$ in S is replaced by a rhythmic transmission in $\tilde{S}$, we define packet $\chi_{i,k}$ as being dropped. Then, the question is which periodic transmissions should be replaced by rhythmic transmissions to generate dynamic schedule $\tilde{S}[t_{n\to r}, t_{ep})$ such that (i) all rhythmic packets meet their deadlines and (ii) minimum number of periodic packets are dropped.

Formally, to satisfy Constraints (ii), (iii) and (iv) in **Problem 1**, we aim to solve the following two subproblems.

Problem 1.1 – End Point Selection: Given task set $\mathcal{T}$, $t_{n\to r}$, t^u_{ep} and static schedule S, this subproblem determines the end point t_{ep} that satisfies the following two constraints.

Constraint 1 $f_{0,m+R} \le t_{ep} \le t^u_{ep}$

Here, $f_{0,m+R}$ is the finish time of the last packet released in τ_0's rhythmic state. $f_{0,m+R} \le t_{ep}$ ensures that the current rhythmic event can be completely handled before the system switches back to the nominal mode.

[4]Some periodic tasks may share common nodes with τ_0 on their routing paths, which indicates that the periodic transmissions at these nodes can be adjusted in the dynamic schedule. Due to page limit, we leave this discussion to our future work and focus on the case that all periodic transmissions should not be adjusted in the dynamic schedule.

Constraint 2 *The system can switch back to the nominal mode and reuse the static schedule from t_{ep} and all packets after t_{ep} meet their nominal deadlines.*

Problem 1.2 – Dynamic Schedule Generation: this subproblem generates the dynamic schedule $\tilde{S}[t_{n\to r}, t_{ep})$ such that the number of dropped packets, denoted as $\rho[t_{n\to r}, t_{ep})$, is minimized and the following two constraints are satisfied.

Constraint 3 *All rhythmic packets meet their deadlines.*

Constraint 4 *In the dynamic schedule $\tilde{S}[t_{n\to r}, t_{ep})$, any periodic transmission slot $S[t] = (i, h)(1 \le i \le n)$ can only either be replaced by a rhythmic transmission slot $S[t] = (0, h)$ or kept unchanged.*

Below we discuss how FD-PaS solves the two problems above.

B. End Point Selection

Determining the right end point for the dynamic schedule is vital since it impacts not only the DHL but also the number of dropped periodic packets. A concept similar to the end point is used by OLS and is referred to as switch point [8]. Since both OLS and FD-PaS require the system to reuse the static schedule after t_{ep}, to select the end point in FD-PaS, we borrow some ideas in OLS including aligning the actual release time of τ_0 to its nominal one and reducing the number of end point candidates by only considering the actual release times of τ_0.

FD-PaS and OLS have two key differences for end point selection. First, to satisfy Constraint 2, we need to determine which packets must be completed before the system reuses the static schedule at end point t_{ep}. Since OLS must obey a user-specified bound on the number of adjusted transmissions in dynamic schedule $\tilde{S}$, a *transmission set* containing all transmissions to be scheduled in $\tilde{S}[t_{n\to r}, t_{ep})$ must be constructed. However, FD-PaS has no such requirement (due to its distributed nature), thus only needs to construct an *active packet set* containing all packets to be scheduled. Second, according to Constraint 4, transmissions of periodic packets must not be adjusted and can only be replaced by rhythmic transmissions in the dynamic schedule. Thus, for the active packet set, we only need to consider rhythmic packets to be scheduled by $\tilde{S}[t_{n\to r}, t_{ep})$. These differences require modifications to the end point selection process, which are detailed below.

Let $\Psi(t_{ep})$ denote the active packet set containing all rhythmic packets to be scheduled within $[t_{n\to r}, t_{ep})$. Naturally, any rhythmic packet with both release time and deadline in $[t_{n\to r}, t_{ep})$ must be included in $\Psi(t_{ep})$. The question is how to treat the rhythmic packet released before t_{ep} with a deadline after t_{ep}. As shown in Fig. 1, let χ_{0,q^*} be such a packet. To ensure the system can reuse the static schedule from t_{ep}, the actual release time of τ_0 must be aligned to its nominal release time after t_{ep}. Same as OLS, we shorten the time interval between r_{0,q^*} and r_{0,q^*+1} by shifting r_{0,q^*+1} backward to the closest nominal release time of τ_0, denoted as r_{0,p^*}. The more challenging part is adjusting the deadline and execution time of χ_{0,q^*} since the number of transmissions may vary depending on which hop occurs after t_{ep}. We construct χ_{0,q^*} by adjusting its execution time and deadline according to the position of t_{ep} by considering the following two cases.

Case 1: If $t_{ep} < r_{0,p^*}$, d_{0,q^*} is adjusted to t_{ep}. Suppose the first transmission of τ_0 after t_{ep} is at t_{h_0} in the static schedule and $S[t_{h_0}]$ is reserved for the h_0-th hop of τ_0, *i.e.*, $S[t_{h_0}] = (0, h_0)$. If $t_{h_0} \geq r_{0,p^*}$, it indicates that $S[t_{h_0}]$ is reserved for the first hop of χ_{0,p^*}, *i.e.*, $S[t_{h_0}] = (0, 1)$. Then the execution time of χ_{0,q^*} is set to H_0. If $t_{h_0} < r_{0,p^*}$, the execution time is set to $h_0 - 1$ correspondingly.

Case 2: If $t_{ep} \geq r_{0,p^*}$, suppose the first hop of χ_{0,p^*} is transmitted at t_1 in the static schedule, *i.e.*, $S[t_1] = (0, 1)(r_{0,p^*} \leq t_1 < d_{0,p^*})$. d_{0,q^*} is adjusted to $\min(t_{ep}, t_1)$ to guarantee that the deadline of χ_{0,q^*} is smaller than or equal to the first transmission of χ_{0,q^*+1}. Also the execution time of χ_{0,q^*} is set to be equal to H_0.

Given t^u_{ep}, any time slot within $[f_{0,m+R}, t^u_{ep}]$ can be selected as end point t_{ep}. However, to avoid checking every time instant which is time consuming, we only need to consider the actual release times of τ_0 within $[f_{0,m+R}, t^u_{ep}]$ as end point candidates, denoted as t^c_{ep} [5]. That is,

$$\{t^c_{ep}\} = \{r_{0,k}, \forall r_{0,k} \in [f_{0,m+R}, t^u_{ep}]\} \quad (1)$$

Then the dynamic schedule generation subproblem **Problem 1.2** can be refined as follow.

Problem 1.2: Given the end point candidate t^c_{ep}, active packet set $\Psi(t^c_{ep})$ and static schedule $S[t_{n\to r}, t^c_{ep})$, determine the dynamic schedule $\tilde{S}[t_{n\to r}, t^c_{ep})$ in which the number of dropped periodic packets is minimized, *i.e.*,

$$\min |\rho[t_{n\to r}, t^c_{ep})| \quad (2)$$

and Constraint 3 and Constraint 4 are satisfied.

C. Dynamic Schedule Generation

Determining the dropped periodic packet set (*i.e.*, solving the problem defined in (2) is a non-trivial problem. Lemma 2 below states the nature of the problem.

Lemma 2 *Given end point t_{ep}, an active packet set $\Psi(t_{ep})$ containing all rhythmic packets of τ_0 to be scheduled and a static schedule $S[t_{n\to r}, t_{ep})$, the packet dropping problem to determine the dropped packet set $\rho[t_{n\to r}, t_{ep})$ with the minimum number of dropped packets and satisfying both Constraint 3 and Constraint 4 is NP-hard.*

Proof: We prove the lemma by reducing the set cover problem [16] to a special case of the packet dropping problem.

The set cover problem is defined as follows: Given a set of n elements $X = \{x_1, x_2, \ldots, x_n\}$ and a collection $C = \{C_1, C_2, \ldots, C_m\}$ of m nonempty subsets of X where $\cup_{i=1}^{m} C_i = X$. The set cover problem is to identify a subcollection $C_s \subseteq C$ whose union equals X such that $|C_s|$ is minimized.

Given a set cover problem, we can construct a special case of the packet dropping problem in polynomial time as follows:

(1) Suppose that after utilizing the original transmission slots of τ_0 and the idle slots in $S[t_{n\to r}, t_{ep})$ to accommodate rhythmic transmissions in $\Psi(t_{ep})$, there still remain n packets of τ_0, denoted as $\{x_1, x_2, \ldots, x_n\}$, to be scheduled. Each packet x_i only needs one slot to transmit.

(2) In the static schedule $S[t_{n\to r}, t_{ep})$, there are m periodic packets, denoted as $\{C_1, C_2, \ldots, C_m\}$. For each packet C_j, if there exists a transmission of C_j falls into the time window of rhythmic packet x_i (*i.e.*, $[r_{x_i}, d_{x_i})$), we have $x_i \in C_j$.

Thus, one can determine the minimum number of dropped packet set $\rho[t_{n\to r}, t_{ep})$ that can accommodate all the rhythmic packets if and only if the smallest sub-collection C_s whose union equals X can be identified. The Lemma is proved. □

After the dropped packet set is determined, the dynamic schedule can be obtained in linear time by assigning the transmissions of the rhythmic packets to the static schedule $S[t_{n\to r}, t_{ep})$ using both idle slots and transmission slots of the dropped packets. Thus Lemma 2 readily leads to Theorem 2 and the proof is omitted.

Theorem 2 *The dynamic schedule generation problem,* **Problem 1.2***, is NP-hard.*

Below we focus on solving the packet dropping problem. Given that the packet dropping algorithm is to be deployed on resource-constrained device nodes and the sizes of both the rhythmic packet set ($|\Psi|$) and periodic packet set ($|\Phi|$) can become large as the network grows, we propose a greedy heuristic to solve the packet dropping problem which is time- and space-efficient to be deployed in practical RTWNs. The key idea of the greedy heuristic is to drop the periodic packet which contributes maximum number of slots to all rhythmic packets. We first introduce the following notation.

$\Psi = \{\chi_{0,i} | 1 \leq i \leq n\}$ denotes the active packet set containing all rhythmic packets to be scheduled. The execution time of $\chi_{0,i}$ is denoted as $H_{0,i}$.

$\Phi = \{\chi_j | 1 \leq j \leq m\}$ denotes the periodic packet set in which each packet χ_j has at least one transmission in the static schedule $S[t_{n\to r}, t_{ep})$.

$\Lambda_j = [\lambda^1_j, \lambda^2_j, \ldots, \lambda^n_j] (1 \leq j \leq m)$ denotes the transmission vector of periodic packet χ_j where each λ^i_j is the number of transmissions from χ_j in the static schedule that can be replaced by transmissions of rhythmic packet $\chi_{0,i}$ in the dynamic schedule. Specifically, transmission $\chi_j(h)$ of χ_j can be replaced by $\chi_{0,i}$ if $S[t] = (j, h)$ and $r_{0,i} \leq t < d_{0,i}$.

$A = [a_1, a_2, \ldots, a_n]$ denotes the available slot vector where each a_i represents the total number of idle slots and rhythmic transmission slots in the static schedule that can be used by rhythmic packet $\chi_{0,i}$.

Alg. 1 describes how the greedy heuristic drops periodic packets. Given the static schedule $S[t_{n\to r}, t_{ep})$, a periodic packet set $\Phi = \{\chi_j | 1 \leq j \leq m\}$ in which each χ_j maintains a transmission vector $\Lambda_j = [\lambda^1_j, \lambda^2_j, \ldots, \lambda^n_j]$ is constructed (Lines 1−2). Considering the idle slots and rhythmic transmission slots in $S[t_{n\to r}, t_{ep})$, the greedy heuristic constructs a demand vector $[v_1, v_2, \ldots, v_n]$ for all rhythmic packets in $\Psi(t_{ep})$ where v_i captures the number of additional slots required by $\chi_{0,i}$ ($v_i = H_{0,i} - a_i$) (Line 3). If all elements in the demand vector equal 0, which means that the idle slots and rhythmic transmission slots in the static

[5] Such a space reduction scheme is safe and can be proved in a similar way as Lemma 2 in [8] which is thus omitted due to page limit.

Algorithm 1 Greedy Heuristic for Dropping Packets

Input: $\Psi(t_{ep})$, $S[t_{n\to r}, t_{ep})$
Output: $\rho[t_{n\to r}, t_{ep})$

1: $\Phi \leftarrow$ periodic packet set $\{\chi_j | 1 \le j \le m\}$;
2: $\Lambda_j \leftarrow$ transmission vector $[\lambda_j^1, \lambda_j^2, \ldots, \lambda_j^n]$ for each periodic packet χ_j;
3: Construct $\Psi(t_{ep})$'s demand vector $[v_1, v_2, \ldots, v_n]$ considering the idle slots and rhythmic transmission slots in $S[t_{n\to r}, t_{ep})$;
4: **if** each v_i equals 0 **then**
5: **return** $\emptyset$;
6: **end if**
7: **while** true **do**
8: Add the periodic packet χ_{max} with the maximum $\sum_{i=1}^{n} \lambda_j^i$ in Φ to $\rho[t_{n\to r}, t_{ep})$;
9: $\Phi \leftarrow \Phi \setminus \{\chi_{max}\}$;
10: **for** $i \in \{1, \ldots, n\}$ **do**
11: $v_i \leftarrow \max(0, v_i - \lambda_{max}^i)$;
12: **end for**
13: **if** each v_i equals 0 **then**
14: **return** $\rho[t_{n\to r}, t_{ep})$;
15: **end if**
16: Update Λ_j for each $\chi_j \in \Phi$;
17: **end while**

Algorithm 2 Dynamic Schedule Generation

Input: t_{ep}^u, $S[t_{n\to r}, t_{ep}^u)$
Output: $\tilde{S}[t_{n\to r}, t_{ep})$

1: Construct the end point candidate set $\{t_{ep}^c\}$ according to (1);
2: **for** $(\forall t_{ep}^c \in \{t_{ep}^c\})$ **do**
3: Construct $\Psi(t_{ep}^c)$; // active packet set
4: Generate a dropped packet set $\rho[t_{n\to r}, t_{ep}^c)$ based on $\Psi(t_{ep}^c)$ using the greedy heuristic;
5: $\mathcal{S} \leftarrow \mathcal{S} \bigcup \{\rho[t_{n\to r}, t_{ep}^c)\}$;
6: **end for**
7: Select the $\rho[t_{n\to r}, t_{ep}^*)$ with the minimum number of dropped packet in $\mathcal{S}$;
8: Generate dynamic schedule $\tilde{S}[t_{n\to r}, t_{ep}^*)$ based on $\rho[t_{n\to r}, t_{ep}^*)$ and $S[t_{n\to r}, t_{ep}^*)$;
9: **return** $\tilde{S}[t_{n\to r}, t_{ep}^*)$;

schedule are sufficient to accommodate all rhythmic packets in $\Psi(t_{ep})$, no packet needs to be dropped and an empty set is returned (Lines 4−6). Otherwise, the heuristic drops packets in a greedy fashion as follows. In each iteration, periodic packet χ_{max} with the maximum $\sum_{i=1}^{n} \lambda_j^i$ in Φ is added into the dropped packet set and removed from Φ (Line 8−9). Then the algorithm updates $\Psi(t_{ep})$'s demand vector by subtracting λ_{max}^i for each v_i (Lines 10−12). If all rhythmic packets are schedulable, *i.e.*, each v_i equals 0, after dropping χ_{max}, the dropped packet set $\rho[t_{n\to r}, t_{ep})$ is returned (Lines 13−15). Otherwise, the transmission vector of each periodic packet χ_j is updated according to the status of rhythmic packets (Line 16). Specifically, if rhythmic packet $\chi_{0,i}$ is already schedulable, *i.e.*, $v_i = 0$, λ_j^i is set to 0. If $0 < v_i < \lambda_j^i$ which means dropping χ_j is redundant to schedule $\chi_{0,i}$, we have $\lambda_j^i = v_i$. This process repeats until all rhythmic packets are schedulable and a dropped packet set is returned.

Finally, Alg. 2 summarizes how nodes in $\mathbf{V}_{rhy}$ generate the dynamic schedule to solve **Problem 1.2**. The time complexity of our dynamic schedule generation algorithm is $O(n \cdot m \cdot \beta)$ where n and m are the numbers of rhythmic and periodic packets in the dynamic schedule, respectively. β (end point scaling factor) is a user specified parameter which determines the length of the dynamic schedule. According to our testbed experiments in Sec. VII-B, all nodes have runtime less than 1ms (within one time slot of 10ms) to complete the dynamic schedule generation.

VII. Performance Evaluation

In this section, we present key performance results from both simulation studies and testbed experiments to evaluate the performance of the FD-PaS framework in RTWNs.

A. Simulation Studies

1) Simulation Setup: In the simulation studies, we compare FD-PaS with both OLS and D^2-PaS approaches that are able to handle unexpected external disturbances in RTWNs. For FD-PaS, we use the greedy heuristic presented in Alg. 1 to determine the dropped packet set. The following two key performance metrics are used in the studies.

Success Ratio (SR): SR is defined as the fraction of feasible task sets over all the generated task sets. A task set is feasible only if a specified DRT can be achieved.

Drop Rate (DR): DR is defined as the ratio between the number of dropped packets and the total number of generated packets in the system rhythmic mode.

For fair comparison, we use randomly generated task sets. Each random task set is generated according to a target nominal utilization U^* and by incrementally adding random periodic tasks to an initially empty set $\mathcal{T}$. The generation of each random task τ_i is controlled by the following parameters: (i) the number of hops H_i drawn from the uniform distribution over $\{2, 3, \ldots, 16\}$, (ii) nominal period P_i drawn from the uniform distribution over $\{H_i, \ldots, 500\}$, and (iii) nominal relative deadline D_i equal to period P_i[6]. After all tasks in $\mathcal{T}$ are generated, we randomly select one of them as the rhythmic task τ_0 and assume that the disturbance is detected at the k-th instance of τ_0 where k is randomly selected from $\{1, \ldots, 20\}$. The period vector $\overrightarrow{P_0}$ ($\overrightarrow{D_0} = \overrightarrow{P_0}$) is generated by controlling the following parameters: (i) the number of elements in $\overrightarrow{P_0}$, R, and (ii) the initial rhythmic period ratio, $\gamma = P_{0,1}/P_0$. To better control the workload of the rhythmic task, we fix γ to 0.2 and tune R which can be any integer in the set of $\{4, 6, \ldots, 16\}$. Given γ and R, the value of each rhythmic period $P_{0,k}$ can be computed by $P_{0,k}\,(1 \le k \le R) = \lfloor P_0 \times (\gamma + (k-1) \times \frac{1-\gamma}{R}) \rfloor$.

Additional parameters needed are summarized as follows: 1) the maximum allowed DRT α which is some integer multiple of the nominal period of the rhythmic task P_0; 2) the end point scaling factor β which determines the upper bound of the end point t_{ep}^u where $t_{ep}^u = t_{r\to n} + (\beta - 1) \times P_0$. Naturally, a larger β will lead to better performance in terms of a smaller number of dropped packets but may cause longer DHL. To keep β as small as possible without performance degradation, we set $\beta = 4$ which means the disturbance must be completely handled within 3 nominal periods after the rhythmic task returns to its nominal state. Other parameters used in OLS and D^2-PaS, *e.g.* the payload size of a broadcast packet, are set to the same as that in [10] for fair comparison.

[6] The unit of P_i and D_i values is one time slot and the range of the parameters are determined according to realistic RTWN applications.

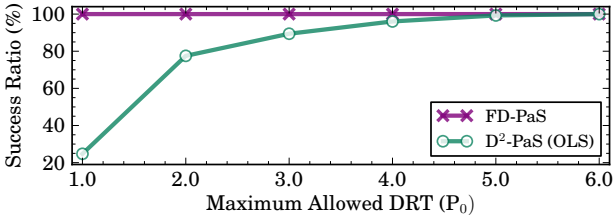

Fig. 6: Comparison of SR with a nominal utilization $U^* = 0.5$.

2) Simulation Results: In the first set of experiments, we compare the SR of OLS, FD-PaS and D^2-PaS for randomly generated task sets with a nominal utilization $U^* = 0.5$ (see Fig. 6) by varying the maximum allowed DRT, α, from P_0 to $6P_0$, with a step size of P_0. The results with other target nominal utilization show similar behavior and thus are omitted. In the experiments, each data point is based on 10,000 randomly generated task sets. As can be observed from Fig. 6, D^2-PaS and OLS have exactly the same SR because they both rely on a broadcast packet to propagate the disturbance information to the entire network. Under D^2-PaS and OLS, the task sets are all feasible only when $\alpha = 6P_0$, *i.e.*, the maximum allowed DRT is set to be 6 nominal periods of the rhythmic task. However, in most practical settings, the RTWN is required to provide fast response to the disturbance within one nominal period, *i.e.*, $\alpha = P_0$. In this case, the SR of both D^2-PaS and OLS drops to 25%. On the other hand, FD-PaS can always achieve 100% SR since the RTWN can start handling disturbance from the beginning of the next nominal period as required by Constraint (i) of **Problem 1** in FD-PaS.

In the second set of experiments, we compare the average DR of OLS, FD-PaS and D^2-PaS by varying the nominal utilization U^* of the randomly generated task sets and the number of rhythmic periods R of the selected rhythmic task. Fig. 7 summarizes the simulation results, where each data point is the average value of $1,000$ trials. From Fig. 7, we can observe that FD-PaS has significantly lower average DR over OLS (53% on average cases and 82% in the best case). The much higher DR of OLS is due to OLD's large broadcast overhead resulted from the centralized approach. Compared to D^2-PaS, FD-PaS drops around 12% more periodic packets on average since FD-PaS makes local packet dropping decisions thus tends to drop more packets. However, with the significant improvement in the average success ratio (75% when the maximum allowed DRT is set to be P_0), this degradation in DR is acceptable.

One somewhat non-intuitive observation from Fig. 7 is that the average DR of FD-PaS first drops when R increases from 4 to 10, and then increases when R keeps increasing from 10 to 16. One would expect that the average DR should monotonically increase when both U^* and R increase (as the case for both OLS and D^2-PaS). Through extensive simulation studies (detailed results are omitted due to page limit), we observe that the average DR of FD-PaS is highly dependent on *slot utilization* (fraction of number of slots contributed by the dropped periodic packets, used by rhythmic packets). When R increases from a small value (*e.g.*, 4), the slot utilization also increases[7]. That is, though we need to drop more packets when the workload of the rhythmic task increases, the number of dropped packets grows more slowly than the number of packets in the system rhythmic mode. This explains why the DR of FD-PaS decreases when R increases from 4 to 10. On the other hand, we found that when R keeps increasing, the slot utilization starts decreasing, and the number of dropped packets grows faster than the total number of packets in the system rhythmic mode. This leads to the observation that the DR of FD-PaS starts to increase from 10 to 16.

TABLE II: Slot Timing Information of MP-MAC

Parameters	Value (μs)	Parameters	Value (μs)
SlotDuration	10,000	LongGT	2,200
TxOffset	2,120	ShortGT	1,000
TxAckDelay	1,000	PriorityTick	30 to 400
Extended SlotDuration	10,800	Extended LongGT	3,000

B. Testbed Implementation and Evaluation

Our testbed is based on OpenWSN stack [17]. OpenWSN is an open source implementation of the 6TiSCH protocol [18]. OpenWSN enables IPv6 network over the TSCH mode of IEEE 802.15.4e MAC and PHY layers. A typical OpenWSN network consists of an OpenWSN Root and several OpenWSN devices, as well as an optional OpenLBR (Open Low-Power Border Router) to connect to IPv6 Internet. It serves as a perfect platform to experiment our proposed FD-PaS framework on both the data link and application layers of the stack.

We implemented FD-PaS on our RTWN testbed to validate the correctness of the design and evaluate its effectiveness for ensuring prompt response to unexpected disturbances. The MP-MAC was implemented by enhancing the data link layer of the OpenWSN stack and the dynamic schedule generation algorithm (using the same code as in the simulation studies) was implemented in the application layer. In the following, we first present the implementation of MP-MAC and its performance evaluation, and then validate the correctness of FD-PaS in a multi-task multi-hop RTWN.

As shown in Fig.10, our testbed consists of 7 wireless devices (TI CC2538 SoC + SmartRF evaluation board). One of them is configured as the root node (controller node) and the rest are device nodes to form a multi-hop RTWN. A CC2531 sniffer is used to capture the packet. A 8-Channel Logic Analyzer is used to record device activities by physical pins, in order to accurately measure the timing information among different devices. Fig. 11 shows the experiment setup for the measurement of application layer performance.

1) Implementation and Evaluation of MP-MAC: For fair comparison with PriorityMAC [15], we used the 10ms slot timing of 802.15.4e in the MP-MAC implementation. Since PriorityMAC adds two subslots (0.4ms each) before each time slot, we also extended the SlotDuration and LongGT of MP-MAC by 0.8ms each. Table II summarizes the slot timing of MP-MAC, and the *Adjusted TxOffset* is computed as follows:

$$\textit{Adjusted TxOffset} = \textit{TxOffset} + (\text{Priority Level}) \times \textit{PriorityTick};$$

[7]For example, when R is small, even if the rhythmic packet only needs one slot from the transmissions of a periodic packet, the whole periodic packet has to be dropped and all its other assigned slots are wasted.

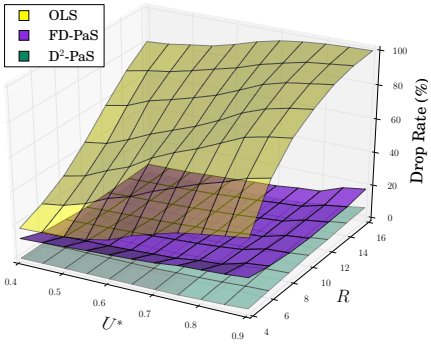

Fig. 7: Comparison of the average DR

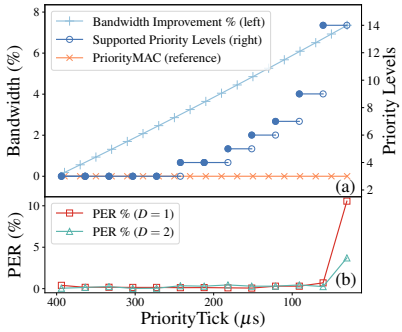

Fig. 8: Priorities and PER vs. PriorityTick

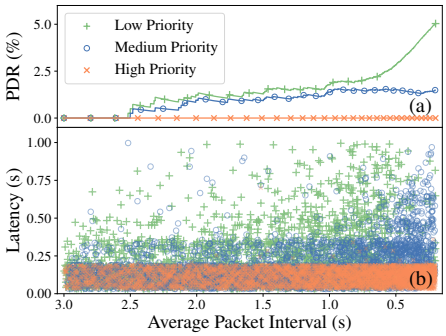

Fig. 9: Measurement of latency and PDR

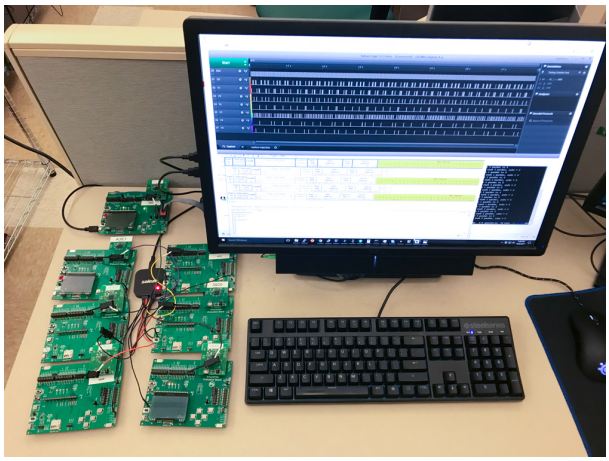

Fig. 10: Overview of the testbed for FD-PaS functional validation

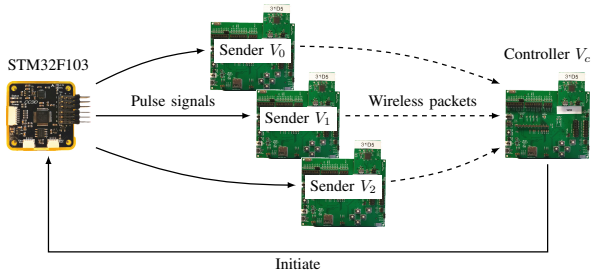

Fig. 11: Experiment setup for the measurement of latency

With a given extended SlotDuration, the number of priority levels that MP-MAC can support, denoted as N, is a function of *PriorityTick*. In our MP-MAC implementation, N is computed by $N = \left\lfloor \frac{0.8ms}{PriorityTick} \right\rfloor + 1$. Fig. 8(a) shows how N changes when the *PriorityTick* varies from 30μs to 400μs with a step size of 30μs (the timer resolution in the OpenWSN stack). Compared to PriorityMAC which can only support 3 effective priority levels, MP-MAC can support up to 14 priority levels in theory by extending the time slot with the same amount (0.8ms). Fig. 8(a) also illustrates the bandwidth improvement, defined as $\frac{10.8ms}{10ms+2\times PriorityTick} \times 100\%$, when MP-MAC only needs to maintain 3 priority levels. It can be seen that the bandwidth is improved by 7% due to the reduction of the *PriorityTick* from 400μs to 30μs with 3 priority levels.

Measurement of Packet Error Rate (PER): Reducing the size of *PriorityTick* can support a larger number of priority levels in the RTWN system. Setting the *PriorityTick* too small, however, either causes nodes to lose synchronization, or make low priority senders unable to detect high priority packet transmissions and cause transmission collisions. It is thus important to identify safe *PriorityTick* values to make MP-MAC work appropriately. For this purpose, we set up a testing network with two senders talking directly to one receiver. We intentionally configure the senders to transmit in the same time slot and assign them with different priorities (using D to denote the distance between the priority levels), and measure the number of correctly received packets on the receiver side. We define Packet Error Rate (PER) as the number of the failed transmissions divided by the number of total transmissions. During the test, each sender generates 10,000 packets. Fig. 8(b) shows the PER of the high priority packets by varying the size of *PriorityTick* from 400μs to 30μs. The PER of the low priority packets are always 100%, and are thus omitted in the figure. It can be observed that MP-MAC works properly under most of the *PriorityTick* settings. Its PER only increases when the *PriorityTick* is reduced to 30μs. This indicates that the MP-MAC implementation on our device node (TI CC2538 SoC) can safely support up to 9 priority levels when the *PriorityTick* is set to be no less than 60μs. When the *PriorityTick* is set at 30μs, it also can be observed from Fig. 8(b) that the PER will drop (from around 10% to 5%) when the distance between the two priority levels increases (from $D = 1$ to $D = 2$).

Measurement of Application Layer Performance: To see how MP-MAC behaves in terms of packet transmission latency and packet drop rate (PDR) for different priority levels, we set up a testing network with three senders and a controller node. The three senders are assigned with different priorities (high, medium and low). Their schedules are configured in a way that they transmit in the same time slot every slotframe (with a length of $165ms$). The retransmission mechanism is enabled on all the senders so that if collision happens, the failed transmission retries in the next slotframe until a maximum number of 5 retries is reached, and the packet is then dropped. We define packet drop rate (PDR) as the number of dropped packets divided by the number of total packets. We connect the controller node to a STM32F103 MCU through a UART port to control the packet generation on the senders. This STM32F103 MCU connects to the GPIO of each sender, and uses a pulse signal to trigger the sender to generate a

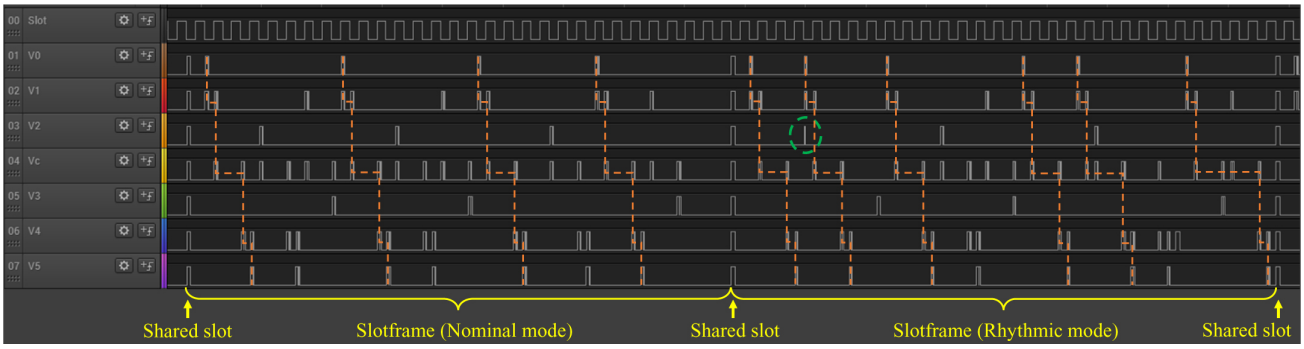

Fig. 12: Slot information and radio activities in the test case captured by Logic Analyzer

packet. In the experiments, the controller node initiates and timestamps the packet generation. By comparing it to the timestamp of the packet reception, the application layer latency is obtained. After a successful packet reception, the controller node waits for a randomly selected time interval, and then triggers the next packet generation. To test latency and packet drop rate, we gradually reduce this time interval to increase the traffic volume. This will cause more transmission collisions in the network, which leads to more packet retransmissions and packet drop.

Fig. 9(a) and (b) show the PDR and application layer latency respectively for the three senders during the test. From the results, we observe that the packets from the high priority sender can always be transmitted in its first attempt while the medium and low priority senders have to yield upon collision by retransmission in future slotframes and suffer longer application layer latency. Similarly, when collision happens with the packets from the medium priority sender, the low priority sender has to yield again thus it is observed to have the longest latency. In Fig. 9(a), we note that the high priority packets can always guarantee the delivery and thus its PDR is consistently 0. On the other hand, both the low priority sender and medium priority sender experience increasing packet losses when the volume of the network traffic grows, and the impact on the low priority sender is more severe.

2) Functional validation in a multi-task multi-hop RTWN: We validate the correctness and effectiveness of FD-PaS by deploying it on a 7-node multi-hop network as shown in Fig. 2. The system running in the network consists of three tasks, $\tau_0 = \{\{V_0, V_1, V_c, V_3, V_4\}, 15\}$, $\tau_1 = \{\{V_2, V_c, V_3, V_4\}, 20\}$ and $\tau_2 = \{\{V_1, V_c, V_5\}, 20\}$, in each of which the first element denotes the routing path and the second is the period (relative deadline). We further assume that τ_0 is the rhythmic task and $\overrightarrow{P_0}(\overrightarrow{D_0}) = [8, 9, 10, 11, 12]$. We use a Logic Analyzer to capture the radio activities from a pin of each device. A sample of collected waveforms is shown in Fig. 12.

In Fig. 12, a falling edge or a rising edge in the top waveform marks the start of a new slot, and the seven waveforms below show the radio activities (either transmit or receive) for the respective seven nodes, as indicated in the left panel. For simplicity, we use only one shared slot at the beginning of each slotframe for management traffic. The slotframe boundary can be easily identified since all devices are radio active at a shared slot. Fig. 12 illustrates 2 slotframes, where the network runs at the nominal mode in the 1st slotframe and switches to the rhythmic mode in the 2nd slotframe. In the experiments, we emulated disturbance by triggering a button during network operation. Upon detecting the event, at the next release time of τ_0, the system switches to the rhythmic mode. In the rhythmic mode (2nd slotframe), τ_0 handled 6 packets while handled only 4 packets in the 1st slotframe (see the 2nd waveform from top). The transmissions of τ_0's packets are marked by orange dashed lines in Fig. 12. To accommodate the increased workload in the system rhythmic mode, the first released packet of τ_1 in the 2nd slotframe is dropped. However, since the sensor node V_2 of τ_1 does not receive the disturbance information and still follows the static schedule, V_2 still sends a packet at the same slot when V_0 sends its second rhythmic packet (marked by the green circle in Fig. 12) according to the generated dynamic schedule. With MP-MAC, the rhythmic transmission from τ_0 is successfully transmitted while the periodic transmission from τ_1 is not. This exactly matches the result that the simulated FD-PaS produces. Hence, the example demonstrates how FD-PaS implemented on the testbed achieves the desired effect in handling disturbance.

Related Work

Network resource management in RTWNs in the presence of unexpected disturbances has drawn a lot of attention in recent years. Traditional static packet scheduling approaches (*e.g.*, [19]–[21]), where decisions are made offline or only get updated infrequently can support deterministic real-time communication, but either cannot properly handle unexpected disturbances or must make rather pessimistic assumptions. Many centralized dynamic scheduling approaches for handling internal disturbances have been proposed (*e.g.*, [22]–[24]). Studies on addressing external disturbances are relatively few and mostly rely on centralized decision making. The approach in [6] only stores a predetermined number of link layer schedules in the system and thus it is incapable of handling arbitrary disturbances. Both [7] and [9] do not consider scenarios when not all tasks can meet their deadlines. The approach in [5] does not consider how to satisfy the deadlines of regular tasks in the presence of disturbances.

In [8], a centralized dynamic approach, named OLS, to handle disturbances in RTWNs is proposed. OLS is built on a dynamic programming based approach which can be rather time consuming even for relatively small RTWNs. Moreover, OLS may drop more periodic packets than necessary due to the limited payload size of the packet in RTWNs and thus further degrade the system performance. To overcome the drawbacks of OLS, D^2-PaS in [10] proposes to offload the computation of

the dynamic schedules to individual nodes locally by leveraging their local computing capabilities, that is, letting each node construct its own schedule so as to achieve better performance than OLS in terms of fewer dropped packets and lower time overhead. However, as observed from the motivating example presented in Section III-A, centralized approaches, including D^2-PaS, suffer from long DRT especially in large RTWNs.

Most MAC layer designs for supporting packet prioritization are based on star topology. For example, the wireless arbitration (WirArb) method [25] is designed to use different frequencies to indicate different priorities. It only supports star topology where the gateway keeps sensing the arbitration signals and determines which user has a higher priority to access the channel. [26] studies a similar problem in the context of vehicular Ad Hoc networks. The proposed multi-priority MAC protocol has seven channels, among which one is the public control channel (CCH) for safety action messages and the others are service channels for non-safety applications. The protocol transmits packets of different priorities with optimal transmission probabilities in a dynamic manner. The PriorityMAC [15] proposes to add two very short sub-slots before each time slot to indicate the priority. Four priority levels are defined but only three levels of over-the-air preemption can be achieved. The last priority level is only used for buffer reordering. In PriorityMAC, a higher priority packet indicates the priority in the sub-slots to deter the transmissions of lower priority packets. PriorityMAC is also based on star topology so each device must be directly connected to the coordinator.

VIII. Conclusion and Future Work

In this paper, we propose FD-PaS, a fully distributed packet scheduling framework, to handle unexpected disturbances in RTWNs. Unlike centralized approaches where dynamic schedules are generated in the controller node and disseminated to the entire network, FD-PaS makes on-line decisions to handle disturbances locally without any centralized control. Such a fully distributed framework not only significantly improves the scalability but also provides guaranteed fast response to external disturbances. Our FD-PaS framework including both the multi-priority data link layer design and the dynamic schedule construction method is implemented on our RTWN testbed. Extensive experiments have been conducted to validate its correctness and effectiveness. As future work, we will extend FD-PaS to support multi-channel settings and will explore how to handle concurrent disturbances in a fully distributed manner.

IX. Acknowledgement

This work is supported in part by the NSF of China under Grant No. 61528202 and 61472072, the U.S. NSF under Grant No. CNS-1319904 and ECCS-1446157 and the collaborative innovation center of major machine manufacturing in Liaoning. The authors would like to thank the reviewers for their valuable comments.

References

[1] X. Hei, X. Du, S. Lin, and I. Lee, "PIPAC: Patient infusion pattern based access control scheme for wireless insulin pump system," in *INFOCOM*, 2013.

[2] K. Gatsis, A. Ribeiro, and G. Pappas, "Optimal power management in wireless control systems," in *ACC*, 2013.

[3] V. M. Karbhari and F. Ansari, "Structural health monitoring of civil infrastructure systems," CRC Press, 2009.

[4] O. Chipara, C. Lu, and G.-C. Roman, "Real-time query scheduling for wireless sensor networks," in *RTSS*, 2007.

[5] B. Li, L. Nie, C. Wu, H. Gonzalez, and C. Lu, "Incorporating emergency alarms in reliable wireless process control," in *ICCPS*, 2015, pp. 218–227.

[6] M. Sha, R. Dor, G. Hackmann, C. Lu, T.-S. Kim, and T. Park, "Self-adapting mac layer for wireless sensor networks," in *RTSS*, 2013.

[7] O. Chipara, C. Wu, C. Lu, and W. G. Griswold, "Interference-aware real-time flow scheduling for wireless sensor networks," in *ECRTS*, 2011.

[8] S. Hong, X. S. Hu, T. Gong, and S. Han, "On-line data link layer scheduling in wireless networked control systems," in *ECRTS*, 2015.

[9] M. Zimmerling, L. Mottola, P. Kumar, F. Ferrari, and L. Thiele, "Adaptive real-time communication for wireless cyber-physical systems," *ACM Transactions on Cyber-Physical Systems*, vol. 1, no. 2, p. 8, 2017.

[10] T. Zhang, T. Gong, C. Gu, H. Ji, S. Han, Q. Deng, and X. S. Hu, "Distributed dynamic packet scheduling for handling disturbances in real-time wireless networks," in *RTAS*, 2017.

[11] J. Kim, K. Lakshmanan, and R. Rajkumar, "Rhythmic tasks: A new task model with continually varying periods for cyber-physical systems," in *ICCPS*, 2012.

[12] J. Song, S. Han, A. Mok, D. Chen, M. Lucas, M. Nixon, and W. Pratt, "Wirelesshart: Applying wireless technology in real-time industrial process control," in *RTAS*, 2008.

[13] D. Dujovne, T. Watteyne, X. Vilajosana, and P. Thubert, "6tisch: deterministic ip-enabled industrial internet (of things)," *IEEE Communications Magazine*, vol. 52, no. 12, pp. 36–41, 2014.

[14] D. De Guglielmo, G. Anastasi, and A. Seghetti, "From ieee 802.15. 4 to ieee 802.15. 4e: A step towards the internet of things," in *Advances onto the Internet of Things*. Springer, 2014, pp. 135–152.

[15] W. Shen, T. Zhang, F. Barac, and M. Gidlund, "Prioritymac: A priority-enhanced mac protocol for critical traffic in industrial wireless sensor and actuator networks," *IEEE Trans. on Industrial Informatics*, 2014.

[16] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of computer computations*. Springer, 1972, pp. 85–103.

[17] T. Watteyne, X. Vilajosana, B. Kerkez, F. Chraim, K. Weekly, Q. Wang, S. Glaser, and K. Pister, "OpenWSN: a standards-based low-power wireless development environment," *Transactions on Emerging Telecommunications Technologies*, vol. 23, no. 5, pp. 480–493, 2012.

[18] T. Watteyne, M. Palattella, and L. Grieco, "Using IEEE 802.15.4e time-slotted channel hopping (TSCH) in the internet of things (IoT): Problem statement," RFC 7554, May 2015.

[19] S. Han, X. Zhu, D. Chen, A. K. Mok, and M. Nixon, "Reliable and real-time communication in industrial wireless mesh networks," in *RTAS*, 2011.

[20] Q. Leng, Y.-H. Wei, S. Han, A. K. Mok, W. Zhang, and M. Tomizuka, "Improving control performance by minimizing jitter in RT-WiFi networks," in *RTSS*, 2014.

[21] A. Saifulah, C. Lu, Y. Xu, and Y. Chen, "Real-time scheduling for WirelessHART networks," in *RTSS*, 2010.

[22] T. L. Crenshaw, S. Hoke, A. Tirumala, and M. Caccamo, "Robust implicit edf: A wireless mac protocol for collaborative real-time systems," *ACM Trans. Embed. Comput. Syst.*, 2007.

[23] W. Shen, T. Zhang, M. Gidlund, and F. Dobslaw, "SAS-TDMA: a source aware scheduling algorithm for real-time communication in industrial wireless sensor networks," *Wireless Networks*, 2013.

[24] F. Ferrari, M. Zimmerling, L. Mottola, and L. Thiele, "Low-power wireless bus," in *SenSys*, 2012.

[25] T. Zheng, M. Gidlund, and J. Åkerberg, "Wirarb: A new mac protocol for time critical industrial wireless sensor network applications," *IEEE Sensors Journal*, vol. 16, no. 7, pp. 2127–2139, 2016.

[26] C. Shao, S. Leng, Y. Zhang, and H. Fu, "A multi-priority supported medium access control in vehicular ad hoc networks," *Computer Communications*, vol. 39, pp. 11–21, 2014.