Reliable Dynamic Packet Scheduling over Lossy Real-Time Wireless Networks

Tao Gong[1]
University of Connecticut, US

Tianyu Zhang
University of Notre Dame, US; Qingdao University, China

Xiaobo Sharon Hu
University of Notre Dame, US

Qingxu Deng
Northeastern University, China

Michael Lemmon
University of Notre Dame, US

Song Han
University of Connecticut, US

Abstract

Along with the rapid development and deployment of real-time wireless network (RTWN) technologies in a wide range of applications, effective packet scheduling algorithms have been playing a critical role in RTWNs for achieving desired Quality of Service (QoS) for real-time sensing and control, especially in the presence of unexpected disturbances. Most existing solutions in the literature focus either on static or dynamic schedule construction to meet the desired QoS requirements, but have a common assumption that all wireless links are reliable. Although this assumption simplifies the algorithm design and analysis, it is not realistic in real-life settings. To address this drawback, this paper introduces a novel reliable dynamic packet scheduling framework, called RD-PaS. RD-PaS can not only construct static schedules to meet both the timing and reliability requirements of end-to-end packet transmissions in RTWNs for a given periodic network traffic pattern, but also construct new schedules rapidly to handle abruptly increased network traffic induced by unexpected disturbances while minimizing the impact on existing network flows. The functional correctness of the RD-PaS framework has been validated through its implementation and deployment on a real-life RTWN testbed. Extensive simulation-based experiments have also been performed to evaluate the effectiveness of RD-PaS, especially in large-scale network settings.

2012 ACM Subject Classification Networks → Network resources allocation; Networks → Network dynamics; Networks → Network reliability

Keywords and phrases Real-time wireless networks, lossy links, dynamic packet scheduling, reliability

1 Introduction

In recent years, real-time wireless networks (RTWNs) have been making their way into a wide range of industrial applications [1, 5, 14, 19]. These applications commonly have stringent timing and reliability requirements to ensure timely data collection and control decision delivery. Thus packet scheduling in RTWNs plays an important role for achieving the desired Quality of Service (QoS) in such applications. QoS here is often measured by how well the network delivers the packets by their deadlines. Although packet scheduling in RTWNs has been studied for a long time, how to handle abruptly increased network traffic in the presence of unexpected disturbances (i.e., events causing more frequent sensing of the

[1] The first two authors have equal contribution to this work.

environment and processing of sensed data) remains a challenge. This challenge is further exacerbated by the lossy wireless links in typical industrial environments [7].

Most RTWNs adopt Time Division Multiple Access (TDMA) based data link layers to achieve deterministic real-time communication. Sensing and control tasks are abstracted as end-to-end (e2e) flows with specified timing and reliability requirements. Most earlier packet scheduling algorithm designs in RTWNs focus on schedulability analysis and employ centralized and static (or infrequently updated) management frameworks (e.g., [17, 18, 16, 8, 22]). Those solutions may fit well for small-scale static RTWNs. They however often lead to significantly degraded QoS when the system becomes large and/or when deployed for monitoring and controlling complex physical processes where disturbances are present.

To model and respond to disturbances in RTWNs, many dynamic scheduling approaches have been proposed. Both [4] and [27] support admission control in response to adding/removing tasks for handling disturbances in the network. They however do not consider scenarios when not all tasks can meet their deadlines. The protocol in [12] proposes to allocate reserved slots for occasionally occurring emergencies (i.e., disturbances), and allow regular tasks to steal slots from the emergency schedule when no emergency exists. However, how to satisfy the deadlines of regular tasks in the presence of emergencies is not considered. [21] proposes a MAC protocol with a centralized reschedule scheme allowing on-line changes of active streams and network topology. However, the scheduler and the data format of the schedule distribution are not specified in [21].

Another thread of research significantly advances the state of the art by providing dynamic packet scheduling functions in RTWNs. Among these approaches, OLS in [9] relies on a centralized gateway to construct and disseminate a dynamic schedule to all the nodes in the network; D^2-PaS in [23] offloads the schedule construction to individual nodes and only disseminates minimum information for the nodes to construct a dynamic schedule locally; and FD-PaS in [26] further eliminates the need of a centralized gateway by notifies and handles the disturbances in a local and distributed manner. They, however, all assume perfect wireless network links, which is not realistic especially in noisy and harsh industrial environments. To our best knowledge, none of the existing dynamic packet scheduling algorithms consider packet losses and thus can lead to poor QoS for real-life deployment.

On the other hand, a rich set of methods have been designed for RTWNs to improve the reliability of wireless packet transmission over lossy links. For instance, most RTWN solutions (e.g., WirelessHART [20], ISA 100.11a [10], and 6TiSCH [6]) employ multiple channels and some frequency hopping mechanisms to minimize potential interference. Further, [8] proposed a set of reliable graph routing algorithms in WirelessHART networks to explore path diversity to improve reliability. These works are complementary to the approach to be introduced in this paper since we focus on single channel with pre-defined routing. [3] proposed an algorithm to allocate a necessary number of retransmision links for individual nodes to guarantee a desired success ratio of packet delivery in a star network topology. [2] extended the network model in [3] to allow multi-hop flows and proposed both Link-Centric and Flow-Centric scheduling policies. However, the policies in [3, 2] tend to assign more retransmission slots than necessary, and thus require higher network bandwidth. Our approach in this work results in an optimal retransmission slot assignment. Furthermore, all aforementioned studies only focus on packet scheduling in static RTWN settings over lossy links, and cannot be easily extended to handle abruptly increased network traffic caused by unexpected disturbances. In a recently submitted work [25], we addressed disturbance handling in lossy RTWNs. However, the schedule used in the static setting is generated by directly applying the retransmission mechanism in [2] which can lead to higher network bandwidth usage than necessary. Further,

[25] is based on the distributed framework FD-PaS to handle disturbances which can cause high QoS degradation on other uncritical tasks according to the results in [26].

In this work, we introduce a reliable dynamic packet scheduling framework, called RD-PaS, for meeting both *timing* and *reliability* requirements in packet scheduling in the presence of disturbances. When no disturbance occurs (i.e., in the static scenario), RD-PaS determines the *minimum* number of retransmission slots needed for each task to guarantee reliable e2e packet delivery, and construct a communication schedule locally in a hybrid manner at individual nodes. The hybrid approach needs a centralized controller and a local schedule generator to keep a good tradeoff between bandwidth usage and QoS. When a disturbance occurs, RD-PaS generates a dynamic schedule to guarantee desired reliability of critical task(s) while judiciously degrade the reliability of packet transmissions for other tasks. We formulate a reliable dynamic scheduling problem to minimize such degradation, prove that this problem is NP-hard, and present an effective heuristic to solve it. The functional correctness of the RD-PaS framework has been validated through its implementation and deployment on a real-life RTWN testbed. Extensive simulation-based experiments have also been performed to evaluate the effectiveness of RD-PaS, especially in large-scale network settings. Our results show that RD-PaS can reduce e2e packet deliver ratio degradation in dynamic schedule by 58% on average compared to the D^2-PaS approach.

The remainder of this paper is organized as follows. Section 2 describes the system model and problem definition, and gives an overview of the RD-PaS framework. Section 3 presents the details of RD-PaS for the Transmission-based Scheduling (TBS) model, including both static schedule construction and dynamic schedule adjustment in the presence of disturbances. These efforts are further extended to the Packet-based Scheduling (PBS) model in Section 4. In Section 5, we present the implementation and functional validation of RD-PaS on a real-life RTWN testbed. Performance evaluation from extensive simulation-based experiments is reported in Section 6. Finally, we conclude the paper and discuss future work in Section 7.

2 Preliminaries

In this section, we first discuss the system model and then give an overview of the proposed RD-PaS framework.

2.1 System Model and Problem Definition

The system architecture of an RTWN studied in this work is modeled after RTWNs often found in industrial process control applications. Such an RTWN consists of multiple sensor and actuator nodes wirelessly connected to a single controller node either directly or through relay nodes. The network is described by a directed graph $G = (V, E)$, where the node set $V = \{V_0, V_1, \ldots, V_c\}$. V_c is the controller node and the rest are referred to as the device nodes. A direct link $e = (V_i, V_j) \in E$ represents a wireless link from node V_i to V_j with a Packet Delivery Ratio (PDR), λ_e^L, which represents the probabilistic transmission success rate on link e[2]. V_c connects to all the nodes via some routes and is responsible for executing relevant control algorithms. V_c also contains a network manager which conducts network configuration and resource allocation. In this work, we focus on RTWNs with only one controller node. Networks with multiple controller nodes are left for future work.

[2] Link PDR λ_e^L is usually measured during the site survey and is stable during normal network operations. In case the value of λ_e^L changes significantly, the new value is assumed to be broadcast to all the nodes in the network.

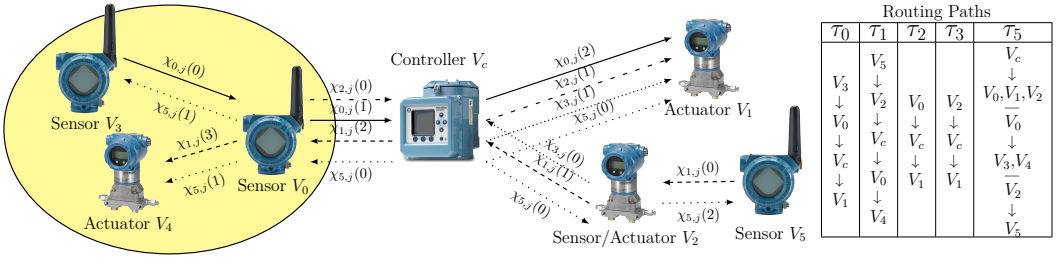


Figure 1 An example RTWN with 5 tasks running on 7 nodes. The sensor and actuator nodes are taken from a crude processing plant.

We assume that the system executes a fixed set of control tasks $\mathcal{T} = \{\tau_0, \tau_1, \dots \tau_n\}$ where τ_i $(0 \leq i < n)$ is a unicast task and τ_n is a broadcast task. Each task τ_i is associated with a period P_i and deadline D_i, and follows a designated single routing path with H_i hops. We use $\overrightarrow{L}_i = [L_i[0], L_i[1], \dots, L_i[H_i - 1]]$ to represent the routing path of task τ_i. For a unicast task, $L_i[h] \in E$ $(0 \leq h < H_i)$. Each unicast task periodically generates a packet originated at a sensor node, passing through the controller node and delivering a control message to the designated actuator node. For the broadcast task τ_n, each hop involves multiple links, thus $L_n[h] = (L_n[h](0), L_n[h](1), \dots)$, where $L_n[h](i) \in E$. The broadcast task runs periodically in V_c and only generates packets when necessary. These packets are broadcast to each node directly or though some intermediate nodes by the designed broadcast path L_n. The j-th released instance of τ_i is referred to as packet $\chi_{i,j}$, with its release time, deadline, and finish time denoted as $r_{i,j}$, $d_{i,j}$ and $f_{i,j}$, respectively. We denote the transmission of packet $\chi_{i,j}$ at the h-th hop as transmission $\chi_{i,j}(h)$, $(0 \leq h < H_i)$.

Fig. 1 shows an example RTWN running 4 unicast tasks (τ_0, τ_1, τ_2 and τ_3) and 1 broadcast task (τ_5) on 7 nodes ($V_0, V_1, \dots, V_5$ and V_c) where V_0, V_3, V_5 are the sensor nodes, V_1, V_4 are the actuator nodes, and V_2 is a combined sensor and actuator node. The routing paths of individual tasks are summarized on the right side of Fig. 1.

In applications such as crude oil refining, a disturbance, e.g., a sudden change in temperature, may occur unexpectedly. When a disturbance occurs, the system usually requires the sensor nodes located within the range of the disturbance to monitor the environment more closely, and thus one or multiple tasks may demand more network bandwidth during the disturbance. To capture such abrupt increase in network resource demand upon the detection of a disturbance, we adopt the rhythmic task model [11] in this work[3]. In the rhythmic model, each task has two states: *nominal state* and *rhythmic state*. In the nominal state, τ_i releases packets following the nominal period P_i and each packet has a relative deadline $D_i \leq P_i$. In the rhythmic state, the period and relative deadline of τ_i adopt a series of new values specified by pre-designed vectors $\overrightarrow{P}_i$ and $\overrightarrow{D}_i$. Once τ_i returns to nominal state, it starts to use P_i and D_i again. When a disturbance occurs and the corresponding tasks (denoted as $\mathcal{T}_{Rhy}$) enter their rhythmic states, we say the system switches to the *rhythmic mode*. The system returns to the *nominal mode* after the disturbance has been completely handled, i.e. all the corresponding tasks return to their nominal states. In Fig. 1, when the disturbance (in the yellow region) occurs, τ_0 and τ_2 (installed on nodes V_3 and V_0, respectively) will enter their rhythmic states and the system switches to the rhythmic mode. In the following, we first assume that at any time during the system operation, at most one disturbance can occur and needs to be detected and handled. We will then generalize the system model to

[3] RD-PaS is not limited to the rhythmic task model and can be applied to any task models capturing unexpected network resource demand changes.

discuss concurrent disturbances at the end of Section 3.2.

Following the industrial practice for RTWNs, we consider a synchronized network adopting a time-slotted schedule. The length of a time slot is typically 10ms. Within each time slot, at most one packet can be transmitted over the air from a sender to a receiver. The acknowledgement (ACK) is then sent back from the receiver to the sender in the same slot to notify the successful reception.

Traditional RTWNs employ Link-based Scheduling (LBS) to allocate time slots. In LBS, each time slot is allocated to a link by specifying the sender and receiver. If packets from different tasks share a common link and are both buffered at the same sender, their transmission order is decided by a node-specified policy (e.g., FIFO). This approach introduces uncertainty in packet scheduling and may violate the e2e timing constraints on packet delivery. To tackle this problem, *Transmission-based Scheduling (TBS)* and *Packet-based Scheduling (PBS)* are proposed in [23] and [2], respectively, to construct deterministic schedules. Each of the two scheduling models has its own advantages and disadvantages and is preferred in different usage scenarios as discussed in [2]. Hence, we consider both models in our RD-PaS framework. Furthermore we focus on single-channel RTWNs in this work since it forms the basis for more advanced studies. Multichannel networks are left for future work.

In the TBS model, each time slot is allocated to the transmission of a specfic packet $\chi_{i,j}$ at a particular hop h or kept idle. Once the network schedule is constructed, packet transmission in each time slot is unique and fixed. In the PBS model, each time slot is allocated to a specific packet $\chi_{i,j}$ or kept idle. Within each time slot assigned to $\chi_{i,j}$, every node along $\chi_{i,j}$'s routing path decides the action to take (e.g., transmit, receive or idle), depending on whether the node has received $\chi_{i,j}$ or not. Table 1 gives a time slot allocation example for task τ_2 in Fig. 1. In TBS model, each time slot is allocated to a dedicated hop. In PBS model, slot 1 can be used to transmit both hops depending on whether the first transmission succeeds in slot 0.

Table 1 An example of time slot allocation in TBS model and PBS model.

	Slot 0	**Slot 1**	**Slot 2**
TBS model	$V_0 \to V_c$	$V_0 \to V_c$	$V_c \to V_1$
PBS model	$V_0 \to V_c$	$V_0 \to V_c$ $V_c \to V_1$	$V_c \to V_1$

Since each link e in the network may suffer packet losses, i.e., $\lambda_e^L < 1$, packet transmissions may fail, which can significantly affect the timely delivery of real-time packets. To handle such cases, a retransmission mechanism is commonly employed in RTWNs [20, 6]. Specifically, if a sender node does not receive the ACK from the receiver node of a packet, it automatically retransmits the packet in the next possible time slot.

To quantify the reliability requirement of the e2e packet delivery for each task, a *required* e2e PDR for τ_i, denoted as λ_i^R, is introduced. For example, a control application can tolerate 0.01% packet loss, so λ_i^R is 99.99%. Based on λ_i^R, the transmission of any packet of τ_i is reliable if and only if the achieved e2e PDR of τ_i is larger than or equal to λ_i^R, i.e., $\lambda_{i,j} \geq \lambda_i^R$. To simplify presentation, we assume that all tasks in the network share a common required e2e PDR value, denoted as λ^R. However, our proposed approach can be easily extended to support different λ^R's for different tasks. Table 2 summarizes the frequently used symbols in this paper.

Based on the above system model, the two key problems that we aim to solve in this work are as follows. **P1:** In the system nominal mode, construct a schedule such that both the e2e timing and reliability requirements of all tasks can be satisfied; **P2:** When disturbances occur and are detected, adjust the schedule in a dynamic and hybrid manner to still guarantee

Table 2 Table of Important Symbols and Notations

$V_0, V_1, \ldots$	Device nodes: sensor, actuator or relay node
V_c	Controller node
$\mathcal{T}$, τ_i	Task set and task i
H_i, P_i, D_i	Number of hops, period and deadline of τ_i
$L_i[h]$	The h-th link on the routing path of τ_i $(0 \leq h < H_i)$
$\chi_{i,j}$	The j-th released packet of τ_i
$r_{i,j}, d_{i,j}, f_{i,j}$	Absolute release time, deadline, finish time of $\chi_{i,j}$
$W_{i,j}$	Total number of slots assigned for $\chi_{i,j}$
λ^R	Required e2e packet delivery ratio (for all tasks)
$\lambda^L_{L_i[h]}$	Measured link packet delivery ratio of link $L_i[h]$
$\lambda_{i,j}$, $\overrightarrow{R}_{i,j}$	E2e PDR value and retry vector of $\chi_{i,j}$
$R_{i,j}[h]$	Number of trials for h-th hop assigned by $\overrightarrow{R}_{i,j}$
$\lambda^*_i(\cdot)$	Optimal PDR of τ_i as a function of number of assigned slots
$\overrightarrow{R}^*_i(\cdot)$	Optimal retry vector of τ_i as a function of numer of assigned slots
w_i^+	The smallest w achieving $\lambda^*_i(w) \geq \lambda^R$
$[t_{sp}, t_{ep})$	Time duration of system rhythmic mode (dynamic schedule)
$\delta_{i,j}$	PDR degradation of $\chi_{i,j}$
Γ	Active packet set containing all packets to be scheduled in the system rhythmic mode
ρ	Updated packet set

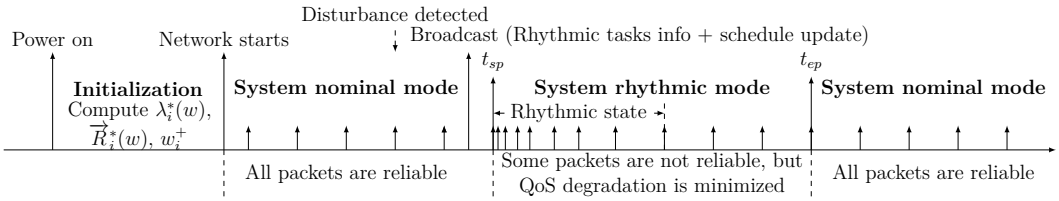


Figure 2 Overview of the execution model of RD-PaS in both nominal and rhythmic modes. Short upward arrows represent the releases of the rhythmic packets.

the reliable and timely transmissions of the rhythmic packets while achieving the minimum reliability degradation on other packets. More formally, we have the following.

P1: Given RTWN $G = (V, E)$ where each link $e \in E$ has an associated PDR, and task set $\mathcal{T}$ in which each task τ_i has a single routing path $\overrightarrow{L}_i$, determine the nominal-mode schedule under which the following constraints are satisfied.

▶ **Constraint 1.** $\forall i, j, \lambda_{i,j} \geq \lambda^R$. (e2e reliability requirements for all tasks)

▶ **Constraint 2.** $\forall i, j, f_{i,j} \leq d_{i,j}$. (e2e timing requirements for all tasks)

P2*: Given the packet set, Γ, in the rhythmic mode under consideration, the PDR function of each task τ_i, and other network related constraints, determine the rhythmic-mode schedule such that $\sum_{\chi_{i,j} \in \Gamma} \max\{0, \lambda^R - \lambda_{i,j}\}$ is minimized, with the following constraints being satisfied.

▶ **Constraint 3.** $\forall \tau_i \in \mathcal{T}_{Rhy}, \lambda_{i,j} \geq \lambda^R$. (e2e reliability requirements for rhythmic tasks)

▶ **Constraint 4.** $\forall \tau_i \in \mathcal{T}_{Rhy}, f_{i,j} \leq d_{i,j}$. (e2e timing requirements for rhythmic tasks)

Here we use **P2*** instead of **P2** as we have not discussed the network constraints. They will be elaborated in Section 3 and 4 where formal definitions of **P2** will be given.

2.2 Overview of the RD-PaS Framework

We propose a reliable dynamic packet scheduling framework, referred to as RD-PaS, to address the questions raised above. An overview of the execution model of RD-PaS is shown

in Fig. 2. Below we focus on a high-level discussion while leave the detailed explanation of the symbols in Section 3.

In the network initialization phase, each device node stores necessary specification information of all tasks (i.e., H_i, D_i, P_i and λ^R) locally after receiving it from the network manager through broadcast packets. Each device node then calculates the number of time slots to be allocated to each task (for both transmission and retransmission) in order to achieve the required e2e PDR value λ^R.

After the network starts, each device node generates a static schedule locally, following which all tasks can meet their timing and reliability requirements. By locally generating a static schedule, no unnecessary bandwidth is wasted on transmitting the schedule from the gateway. When a disturbance occurs, several sensor nodes within the range may detect it and send a report to the controller node via the corresponding tasks. After the controller node receives the disturbance information from any of the sensor nodes, V_c first determines a time duration, denoted as $[t_{sp}, t_{ep})$, during which the system runs in the rhythmic mode using a temporary dynamic schedule. As RD-PaS and D^2-PaS in [23] both require each node to generate schedule locally, RD-PaS adopts the same end point selection method in D^2-PaS to determine the system rhythmic mode duration $[t_{sp}, t_{ep})$. V_c, then, checks whether all tasks can still be reliably delivered after the rhythmic tasks entering their rhythmic states. If so, V_c only broadcasts the rhythmic tasks information (task IDs and the corresponding $\overrightarrow{P}_i$ and $\overrightarrow{D}_i$) to the network. Otherwise, V_c needs to generate a dynamic schedule in which the number of time slots assigned to certain periodic packets are updated in order to accommodate the increased workload from the rhythmic tasks. V_c then piggybacks the information of the updated packet set as well as the rhythmic tasks information to a broadcast packet and disseminates it to all nodes in the network. After all the nodes receive the updates, the system switches to the rhythmic mode to handle the disturbance.

In the rhythmic mode, individual device nodes generate their own dynamic schedules locally and these local schedules collaboratively guarantee the timing and reliability requirements of the rhythmic packets while minimizing the total reliability degradation suffered by other periodic tasks. After executing the dynamic schedules, all the device nodes return to the nominal mode and re-employ the static schedule.

In the following, we first present the details of the RD-PaS framework for the TBS model in Section 3. We then introduce required modifications to support the RD-PaS framework for the PBS model in Section 4.

3 Reliable Scheduling for TBS

This section focuses on reliable scheduling for the Transmission-based Scheduling (TBS) model. We first describe how RD-PaS constructs a reliable static schedule in the system nominal mode. We then introduce how RD-PaS handles disturbances in the rhythmic mode.

3.1 Reliable Static Scheduling

An RTWN starts at running in the nominal mode in which all tasks need to 1) be reliably scheduled to achieve the required e2e PDRs; and 2) meet the e2e timing constraints for all the packet transmissions. That is, we need to solve **P1** defined in Section 2.1. In the TBS model, each specific time slot is assigned to an individual packet transmission. Considering the lossy nature of wireless links, when a transmission is not successful, retransmissions are needed, which require extra time slots. To reduce the demand on network resources, we aim to minimize the number of extra slots for each task while satisfying the reliability

requirement (i.e. Constraint 1 in **P1**). On the other hand, we observe that Constraint 2 can be handled separately from Constraint 1 since satisfying Constraint 2 can be treated as a standard transmission scheduling problem once the number of extra time slots is determined for each task. Thus, we intend to first tackle the following sub-problem.

P1.1: Given RTWN $G = (V, E)$ where each link $e \in E$ has an associated PDR, and task set $\mathcal{T}$ in which each task τ_i has a single routing path $\overrightarrow{L}_i$, determine the minimum number of extra slots needed by each task τ_i for satisfying Constraint 1.

To solve **P1.1**, we propose to first determine whether a given number of extra time slots for each task can satisfy Constraint 1 and then search for the optimal number of extra time slots for every task. We will prove later that this approach indeed leads to an exact solution for **P1.1**. We discuss our approach in detail below.

Let $\overrightarrow{R}_{i,j} = [R_{i,j}[0], R_{i,j}[1], \ldots, R_{i,j}[H_i - 1]]$ be the *retry vector* of packet $\chi_{i,j}$, where $R_{i,j}[h]$ denotes the number of time slots assigned to hop h of $\chi_{i,j}$. We use $W_{i,j}$ to denote the total number of time slots assigned to $\chi_{i,j}$, i.e., $W_{i,j} = \sum_{h=0}^{H_i-1} R_{i,j}[h]$. Given the PDRs of all the links along the routing path of τ_i and the retry vector of $\chi_{i,j}$, the e2e PDR of $\chi_{i,j}$, $\lambda_{i,j}$, can be derived as:

$$\lambda_{i,j} = \prod_{h=0}^{H_i-1} 1 - (1 - \lambda^L_{L_i[h]})^{R_{i,j}[h]}. \tag{1}$$

According to Constraints 1 and 2 in **P1**, all the packets released by τ_i must meet the same timing and reliability requirements in the system nominal mode. Thus, in the following discussion, we only consider parameter settings (including both the assigned number of slots and the retry vector) for each individual task τ_i instead of each packet $\chi_{i,j}$. For a given number of slots, say w, assigned to τ_i, the number of possible slot allocations, i.e. retry vectors, equals to $\binom{w-1}{H_i-1}$. We further introduce the following definitions.

▶ **Definition 1.** Optimal Retry Vector $\overrightarrow{R}^*_i(w)$: *An optimal retry vector of task τ_i for a given number of slots w is the retry vector that leads to the largest PDR value for the given w, denoted as $\lambda^*_i(w)$, among all the possible allocations.*

▶ **Definition 2.** Optimal Retry Vector Function $\overrightarrow{R}^*_i(\cdot)$: *The optimal retry vector function of task τ_i is the set of pairs $(w, \overrightarrow{R}^*_i(w))$ such that each $\overrightarrow{R}^*_i(w)$ is the optimal retry vector for the given number of slots w.*

▶ **Definition 3.** Optimal PDR Function $\lambda^*_i(\cdot)$: *The optimal PDR function of task τ_i is the set of pairs $(w, \lambda^*_i(w))$ such that each PDR value $\lambda^*_i(w)$ corresponds to the optimal retry vector with the given number of slots w.*

As the first step towards satisfying Constraint 1, we present our solution to calculate the optimal retry vector function $\overrightarrow{R}^*_i(\cdot)$ and the optimal PDR function $\lambda^*_i(\cdot)$ for each task τ_i. As both functions are only related to task τ_i itself, the computation for each task is independent. For the sake of clarity, we create a PDR table for each task τ_i to store both $\overrightarrow{R}^*_i(\cdot)$ and $\lambda^*_i(\cdot)$ for all (needed) values of w in each node, such overhead in our implementation is given in Sec. 5. (An example PDR table can be found in Table 4 in Section 5.) Below, we describe our optimal PDR table generation algorithm, Alg. 1, and prove its optimality.

Alg. 1 iteratively constructs the PDR table. At each iteration, we add one time slot to τ_i at the h-th hop that yields the maximum PDR value λ^*_i and store the resulting retry vector $\overrightarrow{R}^*_i$ into the PDR table (Lines 5-7). The retry vector is initially set to $[1, 1, 1, \ldots]$ and the

Algorithm 1 PDR Table Computation under TBS for Task τ_i

Input: $G = (V, E)$, τ_i, λ^R
Output: PDR table of τ_i and w_i^+
1: $w \leftarrow H_i$;
2: $\overrightarrow{R}_i^*(w) \leftarrow [1, 1, 1, \dots]$;
3: $\lambda_i^*(w) \leftarrow \prod_{h=0}^{H_i-1} \lambda_{L_i[h]}^L$;
4: **while** $\lambda_i^*(w) < \lambda^R$ **do**
5: $w \leftarrow w + 1$;
6: Select the hop index h which yields the maximum PDR value (computed by Eq. (1));
7: Update $\overrightarrow{R}_i^*(w)$ and $\lambda_i^*(w)$ in PDR table;
8: **end while**
9: $w_i^+ \leftarrow w$

corresponding PDR value equals to $\prod_{h=0}^{H_i-1} \lambda_{L_i[h]}^L$ (Lines 1-3). Since the required PDR value is λ^R, the iterative process stops when $\lambda_i^*(w) \geq \lambda^R$. We use w_i^+ to denote the minimum number of slots that guarantees the reliable delivery of τ_i.

Lemma 4 and Theorem 5 below affirm that Alg. 1 indeed results in the optimal retry vector function $\overrightarrow{R}_i^*(\cdot)$ and optimal PDR function $\lambda_i^*(\cdot)$.

▶ **Lemma 4.** *Let $\mathcal{G}(R^*(w)[h], \lambda_{L[h]}^L) = \frac{\lambda^*(w+1)}{\lambda^*(w)}$ be a function of $R^*(w)[h]$ and $\lambda_{L[h]}^L$. When $\lambda_{L[h]}^L$ is set to an arbitrary value λ_0, $\mathcal{G}_{\lambda_0} = \mathcal{G}(R^*(w)[h], \lambda_0)$ is a monotonically decreasing function of $R^*(w)[h]$.*

Proof of Lemma 4. If we update $\overrightarrow{R}^*(w)$ by allocating one slot at an arbitrary hop h-th, according to Eq. (1), we only need to update $\lambda^*(w)$ by replacing the term $1-(1-\lambda_{L[h]}^L)^{R^*(w)[h]}$ by $1-(1-\lambda_{L[h]}^L)^{R^*(w)[h]+1}$ to get $\lambda^*(w+1)$. That is,

$$\mathcal{G}(R^*(w)[h], \lambda_{L[h]}^L) = \frac{\lambda^*(w+1)}{\lambda^*(w)} = \frac{1-(1-\lambda_{L[h]}^L)^{R^*(w)[h]+1}}{1-(1-\lambda_{L[h]}^L)^{R^*(w)[h]}}$$

Thus, if $\lambda_{L[h]}^L$ is fixed to λ_0, we have:

$$\mathcal{G}'_{\lambda_0} = \frac{\partial \mathcal{G}(R^*(w)[h], \lambda_0)}{\partial R^*(w)[h]} = \frac{\lambda_0 \cdot (1-\lambda_0)^{R^*(w)[h]} \log(1-\lambda_0)}{\left((1-\lambda_0)^{R^*(w)[h]} - 1\right)^2}$$

Since $0 < \lambda_{L[h]}^L < 1$ and $(1-\lambda_{L[h]}^L)^{R^*(w)[h]} > 0$, we have $\mathcal{G}'_{\lambda_0} < 0$. Further, $\mathcal{G}_{\lambda_0}$ decreases monotonically as $R^*(w)[h]$ increases. ◀

▶ **Theorem 5.** *For any given number of time slots w, no other retry vector can yield a larger PDR value than $\overrightarrow{R}_i^*(w)$ as computed by Alg. 1.*

Proof of Theorem 5. We prove the theorem by mathematical induction, i.e., for any $w = H, H+1, \dots, w^+$, the retry vector $\overrightarrow{R}^*(w)$ determined by Alg. 1 can achieve the largest PDR value $\lambda^*(w)$. (Here we omit the task index i since only one task is considered).
Base case: When $w = H$, the statement holds as only one possible retry vector exists, i.e., $\overrightarrow{R}^*(H) = [1, 1, \dots, 1]$.
Inductive step: Suppose the PDR value of $\overrightarrow{R}^*(w)$ is largest among that of all possible retry vectors when $w = k$, we should prove that the PDR value of $\overrightarrow{R}^*(k+1)$ obtained by Alg. 1, i.e. $\lambda^*(k+1)$ is also the largest. We prove this by contradiction.

Suppose there exists another retry vector (denoted as $\overrightarrow{R}^o(k+1)$) leads a larger PDR value, i.e., $\lambda^*(k+1) < \lambda^o(k+1)$. Since the total number of slots assigned to the task (i.e.,

the sum of all elements in the retry vectors) both equal to $k+1$ and $\overrightarrow{R}^*(k+1) \neq \overrightarrow{R}^o(k+1)$, we can always find one hop at which the number of assigned slots in $\overrightarrow{R}^o(k+1)$ is larger than that in $\overrightarrow{R}^*(k+1)$. We use q to denote this hop index and $R^o(k)[q]$ to denote the number of slots assigned at the q-th hop in $\overrightarrow{R}^o(k)$. Then, $R^o(k+1)[q] > R^*(k+1)[q]$. Suppose $\overrightarrow{R}^*(k+1)$ is achieved by adding one slot at the p-th hop in $\overrightarrow{R}^*(k)$.

Case 1: $p = q$. In this case, $\overrightarrow{R}^*(k+1)$ and $\overrightarrow{R}^o(k+1)$ are both achieved by adding one slot at the p-th hop in $\overrightarrow{R}^*(k)$ and $\overrightarrow{R}^o(k)$, respectively. Then, according to Lemma 4, $\lambda^*(k+1)$ and $\lambda^o(k+1)$ can be rewritten with $\mathcal{G}(R^*(w)[h], \lambda^L_{L[h]})$ function as follows:

$$\lambda^*(k+1) = \lambda^*(k) \cdot \mathcal{G}(R^*(k)[p], \lambda^L_{L[p]}), \quad \lambda^o(k+1) = \lambda^o(k) \cdot \mathcal{G}(R^o(k)[p], \lambda^L_{L[p]}).$$

According to the assumption that the PDR value of $\overrightarrow{R}^*(k)$ is largest, we have $\lambda^*(k) \geq \lambda^o(k)$. Since $R^*(k)[p] < R^o(k)[p]$, according to Lemma 4, we have $\mathcal{G}(R^*(k)[p], \lambda^L_{L[p]}) > \mathcal{G}(R^o(k)[p], \lambda^L_{L[p]})$. Then, $\lambda^*(k+1) > \lambda^o(k+1)$. This contradicts our assumption.

Case 2: $p \neq q$. $\lambda^*(k+1)$ and $\lambda^o(k+1)$ can be rewritten as:

$$\lambda^*(k+1) = \lambda^*(k) \cdot \mathcal{G}(R^*(k)[p], \lambda^L_{L[p]}), \quad \lambda^o(k+1) = \lambda^o(k) \cdot \mathcal{G}(R^o(k)[q], \lambda^L_{L[q]}).$$

As $\lambda^*(k+1) < \lambda^o(k+1)$ and $\lambda^*(k) \geq \lambda^o(k)$, it must holds that

$$\mathcal{G}(R^*(k)[p], \lambda^L_{L[p]}) < \mathcal{G}(R^o(k)[q], \lambda^L_{L[q]}). \tag{2}$$

Since $R^*(k)[q] < R^o(k)[q]$ according to the assumption, the following inequality holds:

$$\mathcal{G}(R^*(k)[q], \lambda^L_{L[q]}) > \mathcal{G}(R^o(k)[q], \lambda^L_{L[q]}). \tag{3}$$

Combining Eq. (2) and Eq. (3), we have $\mathcal{G}(R^*(k)[p], \lambda^L_{L[p]}) < \mathcal{G}(R^*(k)[q], \lambda^L_{L[q]})$. Further,

$$\lambda^*(k) \cdot \mathcal{G}(R^*(k)[p], \lambda^L_{L[p]}) < \lambda^*(k) \cdot \mathcal{G}(R^*(k)[q], \lambda^L_{L[q]}).$$

This means that if we allocate one slot at the q-th hop in $\overrightarrow{R}^*(k)$ instead of at the p-th hop, we can have a larger PDR value. This contradicts with Alg. 1 which allocats one slot at the hop which yields the largest PDR value at each iteration.

Since both cases lead to contradiction, the inductive step is proved. Thus, Theorem.5 holds for all values of w. ◀

Now with the functions $\overrightarrow{R}^*_i(\cdot)$ and $\lambda^*_i(\cdot)$ being determined, we have successfully solved **P1.1**. To satisfy Constraint 2 in **P1**, we need to create a static schedule, i.e., specifying when a packet uses a slot, to ensure real-time constraints are met. We introduce an observation that helps map the reliable static schedule generation problem, i.e., **P1**, to a conventional real-time scheduling problem.

▶ **Observation 1.** *Given task set $\mathcal{T}$ to be reliably scheduled, if we set the number of slots for τ_i to w_i^+ according to $\lambda^*_i(\cdot)$[4], w_i^+ is then equivalent to the execution time of τ_i. Then, each task $\tau_i \in \mathcal{T}$ with P_i, D_i and w_i^+ can be mapped to a task in a conventional real-time task set with the same period, deadline and execution time. Thus, a feasible schedule for the corresponding conventional real-time task set is also a feasible schedule under which all tasks in $\mathcal{T}$ can be reliably delivered.*

[4] All the retry vectors for other w values stored in $\overrightarrow{R}^*_i(\cdot)$ are used in the dynamic schedule generation, which will be discussed in Section 3.2.

Given the schedule specifying the slot assignment for each task, each node can further allocate specific slots to the transmission at each hop according to the retry vector function $\overrightarrow{R}_i^*(\cdot)$. Thus, given a task set to be reliably scheduled in an RTWN, the network can adopt any conventional real-time scheduling algorithm to generate a static schedule that guarantees to meet all the constraints in **P1**. Since we allow at most one transmission within each timeslot, determining the nominal-mode schedule (i.e., **P1**) can be mapped to a uni-processor scheduling problem. Here, we adopt Earliest-Deadline-First (EDF) [13] to generate optimal schedule for tasks and assign time slots to transmissions according to retry vector, consistently at each node.

Note that regarding the broadcast task, two more issues need to be considered. First, the transmission of a broadcast packet at each hop involves one sender node but multiple receiver nodes. Second, no acknowledgement is sent back from the receiver nodes in a broadcast slot. The first issue mainly affects the number of slots assigned at each hop since multiple links with different link PDRs are involved. To tackle this, we directly adopt the lowest link PDR to determine the number of retries assigned at the hop. Due to the second issue, the sender node does not have any knowledge about whether the current transmission succeeds. Thus, we just let the sender node to keep transmitting at all the slots assigned to the current hop to maximize the success probability.

3.2 Reliable Dynamic Scheduling

Our proposed solution for **P1** ensures that both timing and reliability requirements are met in the system nominal mode. However, upon the detection of any disturbance, the corresponding tasks enter their rhythmic states and follow new release patterns and deadlines as shown in Fig. 2. The static schedule may no longer be able to meet both requirements especially for all the critical rhythmic packets. Therefore, a well-designed reliable dynamic packet scheduling mechanism is needed to enable the system to be adaptive to any workload change after the detection of a disturbance.

In our RD-PaS framework, the network generates the static schedule by assigning w_i^+ slots to each task τ_i according to the retry vector function. When a disturbance is detected and reported to the control node, the system follows the execution model outlined in Section 2.2 to switch to the rhythmic mode. The main challenge here is to generate a temporary dynamic schedule when tasks cannot be reliably delivered after the rhythmic tasks (in $\mathcal{T}_{Rhy}$) enter their rhythmic states. That is, problem **P2*** needs to be solved. The dynamic schedule must be able to accommodate the increased rhythmic workload and minimizes the degradation on both timing and reliability of other periodic tasks. Specifically, all the rhythmic packets must meet their timing and reliability requirements. That is, Constraints 3 and 4 are satisfied.

To ensure this, we may have to sacrifice the reliability requirements, i.e. lowering the e2e PDR values of some periodic packets, or even sacrifice their timing requirements, i.e. dropping some periodic packets. That is, the number of slots assigned to each packet may need to be updated. Since the PDR table for each task containing both the retry vector function $\overrightarrow{R}_i^*(\cdot)$ and PDR function $\lambda_i^*(\cdot)$ is pre-calculated and stored at each node, V_c only needs to piggyback on a broadcast packet the information of the updated total number of slots ($W_{i,j}$) assigned to each periodic packet, and then each node can decode the updated retry vector accordingly, once it receives this information.[5]

[5] In the system rhythmic mode, we adjust the assigned number of slots for each packet instead of each task for more flexibility.

To formally define the dynamic schedule generation problem, we introduce some concepts/notation. Let Γ denote the *active packet set* containing all the packets to be scheduled within the rhythmic mode duration $[t_{sp}, t_{ep})$. Since the payload size of a broadcast packet is bounded, we set an upper bound on the number of periodic packets whose $W_{i,j}$ can be changed, and denote it as α. To capture the reliability degradation for periodic packet $\chi_{i,j}$, let $\delta_{i,j}$ represent the difference between the required PDR λ^R and the updated PDR value $\lambda_{i,j} = \lambda_i^*(W_{i,j})$ in the dynamic schedule, i.e., $\delta_{i,j} = \max\{0, \lambda^R - \lambda_{i,j}\}$. Note that the timing degradation of each packet can also be captured by $\delta_{i,j}$ where $\delta_{i,j} = \lambda^R$ if $\chi_{i,j}$ is dropped. Now the dynamic schedule generation problem, which is defined formally below, becomes finding $W_{i,j}$ for each periodic packet in Γ to satisfy Constraint 3 and 4.

P2: Given the active packet set Γ, the PDR function $\lambda_i^*(\cdot)$ of each task τ_i, the maximum allowed number of updated packets α, determine the updated packet set $\rho = \{W_{i,j} | \chi_{i,j} \in \Gamma\}$ such that i) the size of ρ is not larger than α, i.e., $|\rho| \leq \alpha$, and ii) the total reliability degradation is minimized, i.e., $\forall \chi_{i,j} \in \rho, \min \sum \delta_{i,j}$.

The theorem below states that determining the updated packet set, i.e. solving **P2**, is non-trivial.

▶ **Theorem 6.** *The updated packet set generation problem **P2**, i.e., the dynamic schedule generation problem, is NP-hard.*

Proof of Theorem 6. We prove the theorem by reducing the 0-1 knapsack problem [15] to a special case of the updated packet set generation problem.

The 0-1 knapsack problem is defined as follows: Given a set of n items numbered from 1 up to n, each with a weight w_i and a value v_i, along with a maximum weight capacity W. Each item can either be included in the knapsack, denoted as $x_i = 1$, or not which is denoted by $x_i = 0$. The 0-1 knapsack problem is to maximize the sum of the values of the items in the knapsack, i.e. $\max \sum_{i=1}^{n} v_i x_i$, so that the sum of the weights is less than or equal to the knapsack's capacity W, i.e. $\sum_{i=1}^{n} w_i x_i \leq W$ and $x_i \in \{0, 1\}$.

Given a knapsack problem, we construct a special case of the updated packet set generation problem in polynomial time: Suppose the active packet set $\Gamma = \{\chi_1, \chi_2, ..., \chi_n\}$ such that $\forall \chi_i \in \Gamma, r_i = 0, D_i = W, H_i = w_i$. Each packet χ_i can either be scheduled, i.e. $\lambda_i = v_i$ or dropped, i.e. $\lambda_i = 0$. Let the required PDR value λ^R for all packets equals to $\max\{v_i\}$. Then, the PDR degradation $\delta_i = \lambda^R - v_i$ if χ_i is scheduled. Otherwise, $\delta_i = \lambda^R$.

As minimizing the total PDR degradation for all packets equals to maximizing the total PDR value, the updated packet set with the minimum total PDR degradation can be determined if and only if a knapsack with the maximum value can be identified. ◀

Next we propose a heuristic to solve **P2** and the high-level idea is as follows. Since dropping any packet $\chi_{i,j}$ leads to a significant decrease in the PDR value of $\chi_{i,j}$, i.e., $\delta_{i,j} = \lambda^R$, we always prefer to allocate at least the basic number of slots (i.e., H_i) to each packet. If the network bandwidth is sufficient, we assign extra slots to periodic packets in a greedy manner according to their PDR degradation. Alg. 2 summarizes the updated packet set generation algorithm which uses the greedy extra slots assignment heuristic described in Alg. 3. Specifically, at each iteration, Alg. 3 adds one slot to the packet resulting in the minimum PDR degradation after an extra slot has been assigned. Using Alg. 2 and Alg. 3, the updated packet set can be determined in $\mathcal{O}(\alpha \cdot W_{max})$ time where W_{max} is the maximum w_i^+ among all the tasks.

Note that the proposed RD-PaS framework can be readily extended to handle concurrent disturbances in RTWNs, following the similar way as elaborated in [24]. Specifically, we

Algorithm 2 Updated Packet Set Generation

Input: $\Gamma, \alpha, \lambda_i^*(w)$
Output: ρ

Schedule the rhythmic packets in Γ using w_0^+;
//Suppose n is the number of periodic packets in Γ
if all periodic packets in Γ can be reliably scheduled **then**
  No packet needs to be updated;
else
  Find the first $n-\alpha$ schedulable periodic packets with the minimum w_i^+ using the packet-dropping heuristic in [23];
  if Such $n-\alpha$ periodic packets can be found **then**
    if the α packets can be scheduled using H_i **then**
      Assign extra slots to the α packets by Alg. 3;
    else
      Determine the dropped packet set (suppose m packets) using the dropping heuristic in [23];
      Assign extra slots to the $\alpha - m$ packets by Alg. 3;
    end if
  else
    Drop all the periodic packets;
  end if
end if

Algorithm 3 Extra Slots Assignment

$\mathcal{S}_{extra} \leftarrow$ {Packets to be assigned extra slots};
while $\mathcal{S}_{extra} \neq \emptyset$ **do**
  Add one slot to the packet χ_s if doing so leads to the minimum PDR degradation;
  if the system is schedulable **then**
    if χ_s is already reliable **then**
      Remove χ_s from $\mathcal{S}_{extra}$;
    end if
  else
    Reduce one slot from χ_s;
    Remove χ_s from $\mathcal{S}_{extra}$;
  end if
end while

need to handle two cases depending on the relative positions of any two consecutive disturbances [24]. The first case is when both disturbances occur before an upcoming broadcast slot. Then, V_c simply generates a dynamic schedule considering all rhythmic tasks triggered by the two disturbances to handle them together. The second case is when a subsequent disturbance arrives at V_c after the dynamic schedule information for handling the first disturbance has been broadcast. In this case, V_c must update the dynamic schedule starting from the next broadcast slot. The readers are referred to [24] for the details.

4 Reliable Scheduling for PBS

In this section, we discuss how to support the RD-PaS framework for the packet-based scheduling (PBS) model. At the highest level, reliable scheduling for PBS has three main differences from that for TBS. First, since each time slot is assigned to a specific packet instead of a dedicated hop, retry vector $\overrightarrow{R}_{i,j}$ and its function $\overrightarrow{R}_i^*(\cdot)$ are no longer needed. Second, the computation for PDR function $\lambda_i^*(\cdot)$ is different because the time slot allocation

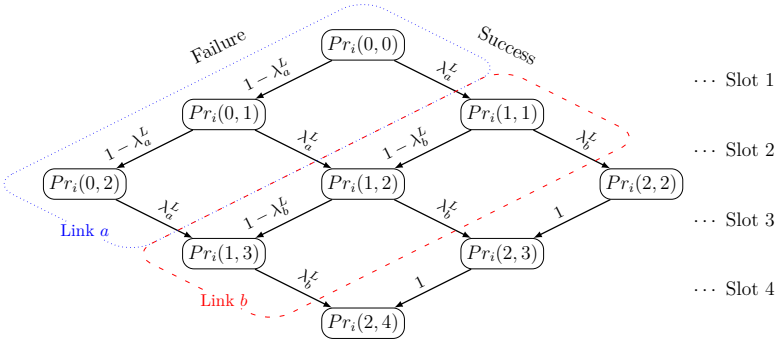


Figure 3 PDR computation for a task with two hops under the PBS model.

mechanism has changed. Third, the retransmission mechanism of the broadcast task for TBS, i.e., keep transmitting using all assigned slots at each hop, does not work for PBS since each slot allocation is not dedicated to a hop but a packet.

Since PDR function is a key parameter in checking reliability, we first describe how to compute the PDR value for a task with a given number of slots in PBS. Let $Pr_i(0,w)$ denote the probability of a packet of τ_i staying in the source node within w slots; $Pr_i(h,w)$ denote the probability of a packet of τ_i being transmitted to the receiver of the h-th hop along the routing path ($1 \le h \le H_i$), and have not been successfully forwarded, within w slots. $Pr_i(h,w)$ can be computed by:

$$Pr_i(h,w) = \begin{cases} 1 & h=0, w=0 \\ \lambda^L_{L_i[h-1]} Pr_i(h-1,w-1) & h \neq 0, w=h \\ (1-\lambda^L_{L_i[h]}) Pr_i(h,w-1) & h=0, w \neq 0 \\ Pr_i(h,w-1) + \lambda^L_{L_i[h-1]} Pr_i(h-1,w-1) & h=H_i, w \neq h \\ (1-\lambda^L_{L_i[h]}) Pr_i(h,w-1) + \lambda^L_{L_i[h-1]} Pr_i(h-1,w-1) & \text{otherwise.} \end{cases} \quad (4)$$

In Fig. 3, we use an example task with 2 hops (links a and b with PDR λ^L_a and λ^L_b, respectively) and 4 slots to describe the computation of $Pr_i(h,w)$. As shown in the figure, $Pr_i(h,w)$ can be either reached by $Pr_i(h-1,w-1)$, followed by a successful transmission ($\lambda^L_{L_i[h-1]}$), or $Pr_i(h,w-1)$, followed by a failed transmission ($1-\lambda^L_{L_i[h]}$), except for boundary conditions. These boundary conditions include the following:

Case 1: When $h=0, w=0$, the source node generates a packet ($Pr_i(0,0)=1$).
Case 2: When $h \neq 0, w=h$, it is not possible for $Pr_i(h,w)$ to be reached by $Pr_i(h,w-1)$ ($Pr_i(1,1)$, $Pr_i(2,2)$ in the figure). Thus only $Pr_i(h-1,w-1)$ is considered.
Case 3: When $h=0, w \neq 0$, it is not possible for $Pr_i(h,w)$ to be reached by $Pr_i(h-1,w-1)$ ($Pr_i(0,1)$, $Pr_i(0,2)$ in the figure). Thus only $Pr_i(h,w-1)$ is considered.
Case 4: When $h=H_i, w \neq h$, $Pr_i(h,w-1)$ always reaches $Pr_i(h,w)$ ($Pr_i(2,3)$, $Pr_i(2,4)$ in the figure).

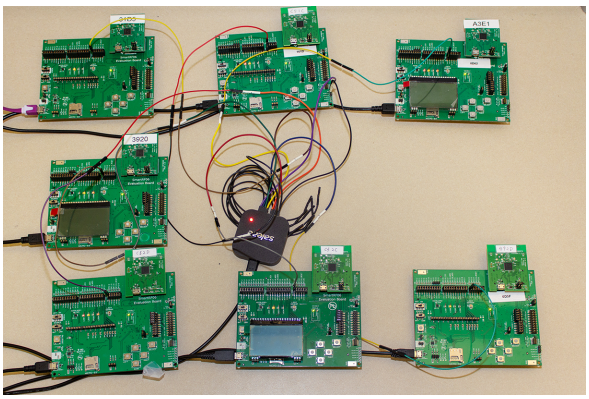
Different from TBS, which finds the optimal PDR values by using retry vectors for a given w, the PDR values in PBS is solely determined by w, i.e., $\lambda_i^*(w) = Pr_i(H_i,w)$. Based on Eq.(4), we propose a dynamic programming algorithm (Alg. 4) to compute $Pr_i(h,w)$ and finally $\lambda_i^*(w)$. In Alg. 4, the iteration starts from $w=1$, and stops when λ^R is reached. In each iteration, it computes all $Pr_i(h,w)$ for $0 \le h \le H_i$, and stores them to $\lambda_i^*(\cdot)$ if $w \ge H_i$.

After the PDR function is computed, we can apply the same method proposed in Section 2.2 and 3 to generate reliable static and dynamic schedule, respectively. More specifically, we

Algorithm 4 PDR Table Computation under PBS for Task τ_i

Input: $G = (V, E)$, τ_i, λ^R
Output: The PDR function of τ_i and w_i^+

```
w ← 0;
while λ_i(w) < λ^R or w < H_i do
    w ← w + 1;
    for h = 0 to H_i do
        Compute Pr_i(h, w) following Eq.(4);
    end for
    if w >= H_i then
        λ*_i(w) ← Pr_i(H_i, w);
    end if
end while
w_i^+ ← w
```



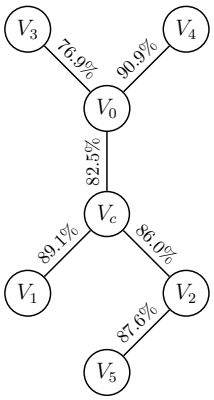


Figure 4 Left: the RTWN testbed with 7 CC2538 evaluation boards; Right: the testing topology with emulated link PDR values.

use Observation 1 with computed PDR function to generate a reliable static schedule, and use Alg. 2 and Alg. 3 to determine the updated W in the rhythmic mode.

Now let us consider the broadcast task. Because the link layer multicast does not have ACK and in PBS each slot is allocated to a packet instead of a hop, it is not possible for the broadcast task to track its progress. Thus the broadcast task still needs to follow the TBS model. That is, for the broadcast task, we adopt the lowest link PDR for each hop among all the receivers, and use Alg. 1 to compute $\overrightarrow{R}_i^*(\cdot)$ and $\lambda_i^*(\cdot)$.

5 Testbed Implementation and Validation

To validate the functionality of the proposed RD-PaS framework in real-life RTWNs, we implemented RD-PaS on a 7-node RTWN testbed (see Fig. 4) running the 6TiSCH protocol. The testbed consists of seven CC2538 evaluation boards. One of these boards is configured as the controller node, while the others are configured as device nodes. A 16-channel 802.15.4 sniffer and an 8-channel logic analyzer are used to capture and analyze the activities of each device node. Our modified 6TiSCH stack utilizes 5KB more ROM and 2KB more RAM space for implementing RD-PaS (in TBS and PBS). These are relatively small compared to the original 6TiSCH stack which needs 69KB ROM and 6KB RAM. Due to the page limit, the implementation details of the RD-PaS framework is omitted. Below, we focus on discussing the functional validation of RD-PaS on the testbed.

The testing topology is shown on the right side of Fig. 4. To attain the link PDRs as

specified in the topology, we implemented a random packet dropper at the MAC layer of each device node. Six tasks are installed in the testbed and the task specifications are summarized in Table 3. The desired e2e PDR for all the tasks, λ^R, is set to 99%. τ_0, τ_1, τ_2 and τ_3 are unicast tasks, τ_5 is a broadcast task, and τ_4 is a task that handles all network management packets. Since we always allocate two *shared* slots at the beginning of τ_4's period, we set $D_4 = 2$. For simplicity, only τ_0 enters the rhythmic state when a rhythmic event occurs.

5.1 Validation of reliable static scheduling

To validate the static schedule construction in RD-PaS, we run the specified task set on the testing topology in the nominal mode under both TBS and PBS models. The PDR tables computed by the testbed are exactly the same as those obtained from simulation. The PDR table for task τ_1 is given in Table 4 (while others are not shown due to the page limit). The highlighted rows indicate the corresponding w_i^+'s for TBS ($w_i^+ = 13$) and PBS ($w_i^+ = 7$) when λ^R is reached.

We further test 5000 packets for each unicast and broadcast task under both models, and compare the actual e2e PDR values collected from the testbed with the simulated values from Alg. 1 and Alg. 4. These results are summarized in Table 5. τ_4 is omitted in the table since it is a task dedicated for network management packets. It can be concluded from the table that the reliable static scheduling function in RD-PaS executes correctly as the actual e2e PDRs are improved to the desired values ($\geq 99\%$) in both models in the presence of specified packet loss. The slight differences between the measured and predicted e2e PDR values are expected due to the limited sample size.

5.2 Validation of reliable dynamic scheduling

To validate the functional correctness of reliable dynamic scheduling in RD-PaS on our testbed, we let the network trigger rhythmic events, and use the logic analyzer to capture the radio activities through a physical pin on each device node and plot the waveforms. We configure the network to enter the rhythmic mode at slot 720. The hyperperiod of the task set is 360 according to Table 3. (Rhythmic events can happen at any time. We chose this integer multiple of the hyperperiod to simplify the waveform demo.) Fig. 5 illustrates a sample waveform for 240 consecutive slots (slot 600-840) in the TBS model. (Both TBS and PBS models are validated. We present the results in the TBS model here for ease of explanation.) The network runs in the nominal mode for the first 120 time slots (Fig. 5b) and then switches to the rhythmic mode in the next 120 slots (Fig. 5c). Seven waveforms represent the radio activities, either transmitting, receiving, or listening, for all the 7 nodes, as labeled on the left side of the figures. Each rising and falling edge in the *Slot* row (lower part of the figures) mark the start of a new time slot. In the *schedule* row (lower part of the figures), slot assignments are indicated using different colors.

Table 3 Parameters of the task set deployed on the testbed.

Task	Routing Path	$P_i(D_i)$	$\overrightarrow{P}_i = \overrightarrow{D}_i$
τ_0	$V_3 \to V_0 \to V_c \to V_1$	30 (30)	$[20, 20, 20, 20, 20, 20]$
τ_1	$V_5 \to V_2 \to V_c \to V_0 \to V_4$	45 (45)	-
τ_2	$V_0 \to V_c \to V_1$	40 (40)	-
τ_3	$V_2 \to V_c \to V_1$	60 (60)	-
τ_4	-	60 (2)	-
τ_5	$V_c \to (V_0, V_1, V_2), V_0 \to (V_3, V_4), V_2 \to (V_5)$	120 (120)	-

Table 4 PDR table for task τ_1 in TBS and PBS models.

w	PDR Table in TBS Model		PDR Table in PBS Model
	$\lambda_i^*(w)$	$\overrightarrow{R}_i^*(w)$	$\lambda_i^*(w)$
4	0.564963	1,1,1,1	0.564963
5	0.663832	1,1,2,1	0.864394
6	0.756769	1,2,2,1	0.964613
7	0.850608	2,2,2,1	0.991720 ($\lambda_i^*(w_i^+)$)
8	0.928013	2,2,2,2	
9	0.952201	2,2,3,2	
10	0.968572	2,3,3,2	
11	0.981822	3,3,3,2	
12	0.989274	3,3,3,3	
13	0.993672 ($\lambda_i^*(w_i^+)$)	3,3,4,3	

Table 5 Reliable static schedule validation in TBS and PBS models on the testbed.

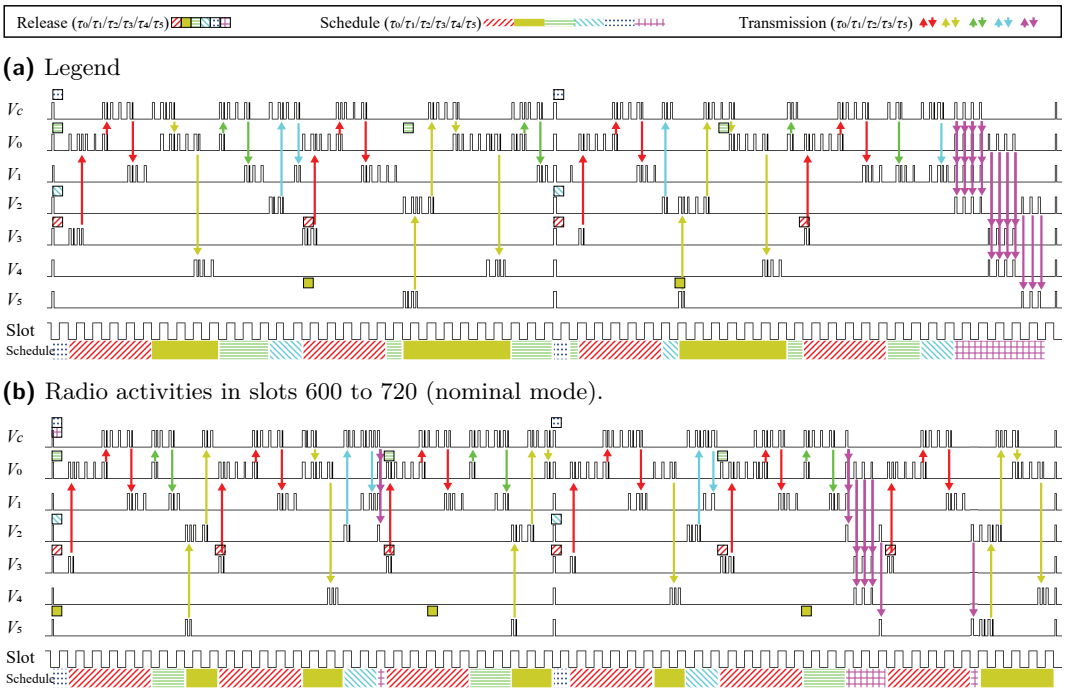
Task	TBS Model			PBS Model		
	$\overrightarrow{R}_i^*$	$\lambda_i^*(w_i^+)$	Measured PDR	w_i^+ or $\overrightarrow{R}_i^*$	$\lambda_i^*(w_i^+)$	Measured PDR
τ_0	[4,3,3]	99.01%	99.21%	7	99.68%	99.22%
τ_1	[3,3,4,3]	99.37%	99.61%	7	99.17%	99.65%
τ_2	[3,3]	99.34%	99.41%	5	99.80%	99.34%
τ_3	[3,3]	99.60%	99.71%	4	99.29%	99.65%
τ_5	[4,4,3]	99.38%	100%	[4,4,3]	99.38%	100%

From Fig. 5b, we observe that each task τ_i releases its packets according to P_i, and w_i^+ number of slots are allocated to each packet before its deadline (shown in the *schedule* row). In each scheduled slot, the sender attempts to transmit the packet and may succeed (marked by the arrows). Although some attempts fail, all the packets are still delivered to the destination node because of the right amount of retransmission slots as determined by the reliable static scheduling function. In Fig. 5c, τ_0 enters the rhythmic state, and its period is reduced according to $\overrightarrow{P}_0$ given in Table 3. Also as shown in the *schedule* row, the $W_{i,j}$ values for τ_0 do not change, while those for τ_1, τ_2, τ_3, τ_5 are reduced to $[9, 9, 9]$, $[4, 5, 5]$, $[4, 4]$, $[7]$, respectively. The $\overrightarrow{R}_{i,j}$ vectors are also selected correctly by the updated $W_{i,j}$ values in the rhythmic mode, and all the packets from the rhythmic task (τ_0) are successfully delivered to the destination. The captured results match the results from the simulation, and this validates the correctness of the reliable dynamic scheduling function in RD-PaS.

6 Simulation-based Performance Evaluation

In this section, we evaluate the performance of RD-PaS through extensive simulations and compare RD-PaS with a state-of-the-art dynamic approach, D^2-PaS.[6] The first three sets of simulations compare packet delivery ratio, network bandwidth usage and number of extra slots produced by RD-PaS with those by D^2-PaS. The last set of simulations studies the behavior of the rhythmic mode. We evaluate the reliability degradation by comparing RD-PaS with D^2-PaS on handling disturbances in RTWNs.

[6] [26] shows that D^2-PaS has a clear advantage in packet dropping performance compared to the fully distributed scheduling framework FD-PaS, so we omit the comparison between RD-PaS and FD-PaS. Also, since we have proved the optimality of our retransmission slots assignment in Sec. 3.1, we omit to compare with the retransmission mechanism in [2] in the static setting.



(a) Legend

(b) Radio activities in slots 600 to 720 (nominal mode).

(c) Radio activities in slots 720 to 840 (rhythmic mode). Task τ_0 is in the rhythmic state and releases packets following $\overrightarrow{P}_0$ given in Table 3.

Figure 5 Slot information and radio activities in the reliable dynamic scheduling test case captured by the logic analyzer.

6.1 Comparison of Packet Delivery Ratio

As RD-PaS utilizes retransmission slots to guarantee the required e2e PDR value for each task, there is no doubt that the system reliability will be improved compared with a traditional scheduling framework not considering reliablity. To quantify such improvements, we calculate the e2e PDR resulted from applying D^2-PaS in lossy links with randomly generated link PDRs. Since the e2e PDR for each task is independent, we use different settings to randomly generate tasks and compute the PDR value for each task. The number of hops for a task, H, is drawn from the uniform distribution over $\{1, 2, ..., 10\}$ and the PDR value of each link on the routing path is randomly generated by controlling the average value of link PDR, λ^L, following a uniform distribution in $\{0.5, 0.55, ..., 1\}$. As periods and deadlines do not affect the packet delivery ratio, we only study PDR's dependcy on H and λ^L. Fig. 6 shows the e2e PDR of a task as a function of λ^L and H. Because RD-PaS can always guarantee the required PDR value, its results are always at the ceiling (above 99%) of the figure and are thus omitted. From Fig. 6, we can observe the large gap between RD-PaS and D^2-PaS (60.6% on average) in guaranteeing the e2e PDR of the task.

6.2 Comparison of Network Bandwidth Usage

Allocating extra retransmission slots can significantly improve the reliability of packet delivery. However, higher network bandwidth is required which may affect system schedulability. In this set of experiments, we study the efficiency of using time slots to deliver packets, in different scheduling frameworks, according to the performance metric *throughput*. Throughput is defined as the number of packets delivered per slot (PPS) and is the ratio between the e2e

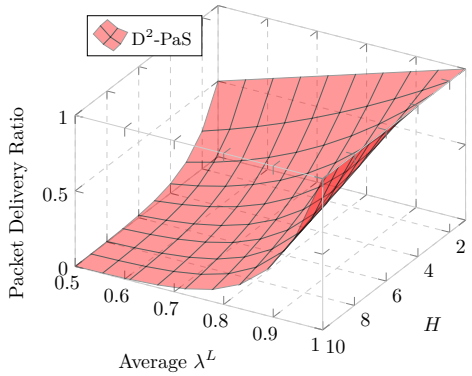


Figure 6 PDR in D^2-PaS framework.

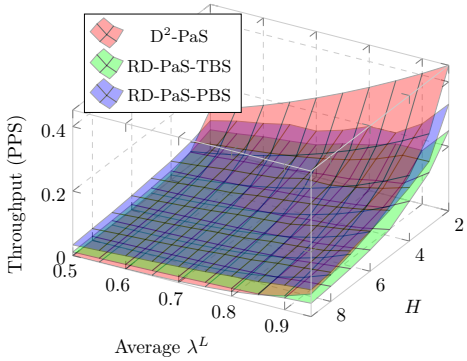


Figure 7 Throughput comparison among different scheduling frameworks.

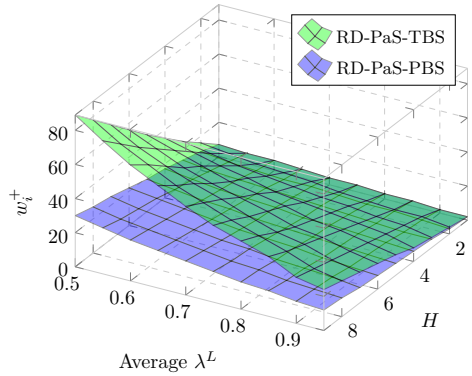


Figure 8 Comparison of w_i^+ in TBS and PBS.

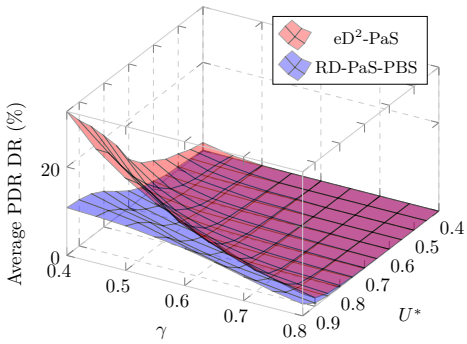


Figure 9 Comparison of the PDR degradation rate.

PDR value and the number of allocated slots assigned to the task, i.e. $\frac{\lambda_i^*(w)}{w}$. The parameter settings of this set of experiments are the same as that in Section 6.1.

Fig. 7 summarizes throughputs for different scheduling frameworks with varied average link PDR λ^L and the number of hops, H, for the generated task. From the results, we can observe that D^2-PaS has a higher throughput when H is small and when λ^L is close to 1. However when the link PDR drops and H increases, RD-PaS (in both TBS and PBS models) gains better throughput. This is mainly due to the fact that using a time slot for retransmission can gain more throughput than transmitting a new packet in these cases. The simulation results also show that RD-PaS in the PBS model can always achieve a better throughput than in the TBS model. The reason is that the PBS model can always achieve same PDR with less number of slots, compared to the TBS model due to the PBS's ability in sharing slots among transmissions of a packet.

6.3 Comparison of Required Numbers of Slots

In this set of experiments, we make further evaluation on RD-PaS in TBS and PBS models. As discussed in Section 4, the PBS model provides more flexibility on the retransmission slot assignment, and a less number of slots, w_i^+, is required to achieve the same λ^R as compared to the TBS model. Fig. 8 gives the comparison on the required number of slots under different settings of average λ^L and H, and the required end-to-end PDR value λ^R is set to 99%. As can be observed, tasks in PBS model require less number of slots than in TBS model, when $H > 1$. The required number of slots in the PBS model is 55.0% less on average compared

to that in TBS model. This is consistent with the observation that one packet requires less number of slots to achieve the same λ^R in the PBS model.

6.4 Effectiveness in Handling Rhythmic Events

To evaluate the performance of RD-PaS in handling rhythmic events, we compare the *degradation rate (DR)* between RD-PaS and D^2-PaS. DR is defined as the ratio between the sum of reliability degradation (i.e., $\delta_{i,j}$) from all periodic packets and the total number of generated periodic packets in the rhythmic mode. As D^2-PaS does not consider unreliable wireless links, we first extend D^2-PaS to support reliable transmission, denoted as eD^2-PaS. Specifically, all packets in eD^2-PaS are reliably transmitted using w_i^+ slots in the static schedule. In the dynamic schedule, transmission and retransmission slots assigned for each packet are not differentiated, i.e., each packet can either be reliably scheduled or dropped.

To better control the system workload, we vary the nominal utilization of the task set. Specifically, we use a random periodic task set generated according to a target nominal utilization U^*. The generation of each random task τ_i is controlled by the following parameter settings: i) the number of hops H_i is drawn from the uniform distribution over $\{2, 3, ..., 16\}$, ii) the nominal period P_i is equal to deadline D_i and follows a uniform distribution in $\{50, 51, ...100\}$. As the simulation results in the last sub-section have shown, the PBS model requires less total number of slots to achieve the same transmission reliability. Thus, here we use the PBS model to generate the PDR function $\lambda_i^*(\cdot)$ for each task τ_i.

After a task set is generated, we randomly select two tasks to be the rhythmic tasks. To better control the workload of the rhythmic event, we assume that all the rhythmic periods (deadlines) are the same in $\overrightarrow{P}_i(\overrightarrow{D}_i)$ and the number of elements in $\overrightarrow{P}_i$ equals to 10. The value of each element $P_{i,R}$ is thus controlled by the rhythmic period ratio, $\gamma = \frac{P_{i,R}}{P_i}$.

Fig. 9 shows the results of DR as a function of both the nominal task set utilization U^* and the rhythmic period ratio γ. Each data point is the average value of $1,000$ trials. From Fig. 9, we can observe that RD-PaS has a lower PDR degradation rate (58.4% on average) over eD^2-PaS. The main reason is that eD^2-PaS either schedules or drops any packet $\chi_{i,j}$, i.e. $W_{i,j} \in \{0, w_i^+\}$. However, RD-PaS has more flexibility on tuning the number of slots assigned to $\chi_{i,j}$, i.e. $W_{i,j} \in \{0, H_i, \ldots, w_i^+\}$.

7 Conclusion and Future Work

In this paper, we present RD-PaS, a reliable dynamic packet scheduling framework for RTWNs. RD-PaS provides guaranteed reliability of packet delivery in RTWNs for both transmission-based scheduling model and packet-based scheduling model in a hybrid manner. In the presence of unexpected disturbances, RD-PaS makes dynamic schedule adjustment judiciously to guarantee timely and reliable delivery of the critical rhythmic packets while minimizes reliability degradation for noncritical packets. A provably optimal algorithm (for the static case) as well as a heuristic (for the dynamic case) are introduced for realizing RD-PaS. Extensive testbed and simulation based experiments are conducted to validate the correctness and effectiveness of RD-PaS. Our experimental results show that RD-PaS can significantly improve the QoS (in terms of reliability) compared with the state-of-the-art approaches. As future work, we will extend RD-PaS to further support RTWNs with multi-channel scheduling and multi-path routing capabilities, and evaluate its performance in large-scale RTWN testbeds.

References

1. Johan Åkerberg, Mikael Gidlund, and Mats Björkman. Future research challenges in wireless sensor and actuator networks targeting industrial automation. In *2011 9th IEEE International Conference on Industrial Informatics*, pages 410–415, July 2011. doi:10.1109/INDIN.2011.6034912.
2. Ryan Brummet, Dolvara Gunatilaka, Dhruv Vyas, Octav Chipara, and Chenyang Lu. A flexible retransmission policy for industrial wireless sensor actuator networks. In *2018 IEEE International Conference on Industrial Internet (ICII)*, pages 79–88, Oct 2018. doi:10.1109/ICII.2018.00017.
3. Yu Chen, Hongwei Zhang, Nathan Fisher, Le Yi Wang, and George Yin. Probabilistic per-packet real-time guarantees for wireless networked sensing and control. *IEEE Transactions on Industrial Informatics*, 14(5):2133–2145, May 2018. doi:10.1109/TII.2018.2795567.
4. Octav Chipara, Chengjie Wu, Chenyang Lu, and William Griswold. Interference-aware real-time flow scheduling for wireless sensor networks. In *2011 23rd Euromicro Conference on Real-Time Systems*, pages 67–77, July 2011. doi:10.1109/ECRTS.2011.15.
5. Li Da Xu, Wu He, and Shancang Li. Internet of things in industries: A survey. *IEEE Transactions on Industrial Informatics*, 10(4):2233–2243, Nov 2014. doi:10.1109/TII.2014.2300753.
6. Diego Dujovne, Thomas Watteyne, Xavier Vilajosana, and Pascal Thubert. 6TiSCH: deterministic ip-enabled industrial internet (of things). *IEEE Communications Magazine*, 52(12):36–41, December 2014. doi:10.1109/MCOM.2014.6979984.
7. Vehbi C Gungor, Gerhard P Hancke, et al. Industrial wireless sensor networks: Challenges, design principles, and technical approaches. *IEEE Transactions on Industrial Electronics*, 56(10):4258–4265, Oct 2009. doi:10.1109/TIE.2009.2015754.
8. Song Han, Xiuming Zhu, Aloysius K Mok, Deji Chen, and Mark Nixon. Reliable and real-time communication in industrial wireless mesh networks. In *2011 17th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 3–12, April 2011. doi:10.1109/RTAS.2011.9.
9. Shengyan Hong, Xiaobo Sharon Hu, Tao Gong, and Song Han. On-line data link layer scheduling in wireless networked control systems. In *2015 27th Euromicro Conference on Real-Time Systems*, pages 57–66, July 2015. doi:10.1109/ECRTS.2015.13.
10. ISA Standard. Wireless systems for industrial automation: process control and related applications. *ISA-100.11 a-2009*, 2009.
11. Junsung Kim, Karthik Lakshmanan, and Ragunathan Raj Rajkumar. Rhythmic tasks: A new task model with continually varying periods for cyber-physical systems. In *2012 IEEE/ACM Third International Conference on Cyber-Physical Systems*, pages 55–64, April 2012. doi:10.1109/ICCPS.2012.14.
12. Bo Li, Lanshun Nie, Chengjie Wu, Humberto Gonzalez, and Chenyang Lu. Incorporating emergency alarms in reliable wireless process control. In *Proceedings of the ACM/IEEE Sixth International Conference on Cyber-Physical Systems*, ICCPS '15, pages 218–227, New York, NY, USA, 2015. ACM. URL: http://doi.acm.org/10.1145/2735960.2735983, doi:10.1145/2735960.2735983.
13. Chung Laung Liu and James W Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 1973.
14. Chenyang Lu, Abusayeed Saifullah, Bo Li, Mo Sha, Humberto Gonzalez, Dolvara Gunatilaka, Chengjie Wu, Lanshun Nie, and Yixin Chen. Real-time wireless sensor-actuator networks for industrial cyber-physical systems. *Proceedings of the IEEE*, 104(5):1013–1024, May 2016. doi:10.1109/JPROC.2015.2497161.
15. Silvano Martello, David Pisinger, and Paolo Toth. New trends in exact algorithms for the 0–1 knapsack problem. *European Journal of Operational Research*, 123(2):325 – 332, 2000. URL: http://www.sciencedirect.com/science/article/pii/S037722179900260X, doi:https://doi.org/10.1016/S0377-2217(99)00260-X.

16 Abusayeed Saifullah, Dolvara Gunatilaka, Paras Tiwari, Mo Sha, Chenyang Lu, Bo Li, Chengjie Wu, and Yixin Chen. Schedulability analysis under graph routing in WirelessHART networks. In *2015 IEEE Real-Time Systems Symposium*, pages 165–174, Dec 2015. doi:10.1109/RTSS.2015.23.

17 Abusayeed Saifullah, You Xu, Chenyang Lu, and Yixin Chen. Real-time scheduling for WirelessHART networks. In *2010 31st IEEE Real-Time Systems Symposium*, pages 150–159, Nov 2010. doi:10.1109/RTSS.2010.41.

18 Abusayeed Saifullah, You Xu, Chenyang Lu, and Yixin Chen. End-to-end communication delay analysis in industrial wireless networks. *IEEE Transactions on Computers*, 64(5):1361–1374, May 2015. doi:10.1109/TC.2014.2322609.

19 Emiliano Sisinni, Abusayeed Saifullah, Song Han, Ulf Jennehag, and Mikael Gidlund. Industrial internet of things: Challenges, opportunities, and directions. *IEEE Transactions on Industrial Informatics*, 14(11):4724–4734, Nov 2018. doi:10.1109/TII.2018.2852491.

20 Jianping Song, Song Han, Al Mok, Deji Chen, Mike Lucas, Mark Nixon, and Wally Pratt. WirelessHART: Applying wireless technology in real-time industrial process control. In *2008 IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 377–386, April 2008. doi:10.1109/RTAS.2008.15.

21 Federico Terraneo, Paolo Polidori, Alberto Leva, and William Fornaciari. TDMH-MAC: Real-time and multi-hop in the same wireless mac. In *2018 IEEE Real-Time Systems Symposium (RTSS)*, pages 277–287, Dec 2018. doi:10.1109/RTSS.2018.00044.

22 Haibo Zhang, Pablo Soldati, and Mikael Johansson. Performance bounds and latency-optimal scheduling for convergecast in WirelessHART networks. *IEEE Transactions on Wireless Communications*, 12(6):2688–2696, June 2013. doi:10.1109/TWC.2013.050313.120543.

23 Tianyu Zhang, Tao Gong, Chuancai Gu, Huayi Ji, Song Han, Qingxu Deng, and Xiaobo Sharon Hu. Distributed dynamic packet scheduling for handling disturbances in real-time wireless networks. In *2017 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 261–272, April 2017. doi:10.1109/RTAS.2017.11.

24 Tianyu Zhang, Tao Gong, Song Han, Qingxu Deng, and X Sharon Hu. Distributed dynamic packet scheduling framework for handling disturbances in real-time wireless networks. *IEEE Transactions on Mobile Computing*, pages 1–1, 2018. doi:10.1109/TMC.2018.2877681.

25 Tianyu Zhang, Tao Gong, Song Han, Qingxu Deng, and Xiaobo Sharon Hu. Fully distributed packet scheduling framework for handling disturbances in lossy real-time wireless networks, 2019. arXiv:1902.02023.

26 Tianyu Zhang, Tao Gong, Zelin Yun, Song Han, Qingxu Deng, and Xiaobo Sharon Hu. FD-PaS: A fully distributed packet scheduling framework for handling disturbances in real-time wireless networks. In *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 1–12, April 2018. doi:10.1109/RTAS.2018.00007.

27 Marco Zimmerling, Luca Mottola, Pratyush Kumar, Federico Ferrari, and Lothar Thiele. Adaptive real-time communication for wireless cyber-physical systems. *ACM Transactions on Cyber-Physical Systems*, 1(2):8:1–8:29, February 2017. URL: http://doi.acm.org/10.1145/3012005, doi:10.1145/3012005.