Distributed Dynamic Packet Scheduling Framework for Handling Disturbances in Real-Time Wireless Networks

Tianyu Zhang, Tao Gong, Song Han, Member, IEEE, Qingxu Deng, Member, IEEE, and Xiaobo Sharon Hu, Fellow, IEEE

Abstract—Real-time wireless networks (RTWNs) are fundamental to many Internet-of-Things (IoT) applications. RTWNs typically apply time-division multiple access (TDMA)-based media access control mechanisms and often demand deterministic end-to-end packet delivery to meet the given quality of service (QoS) requirements. Packet scheduling in an RTWN thus plays a critical role for achieving the desired performance but is a challenging problem especially when the RTWN is large and must deal with multiple disturbances (i.e., unexpected events causing abrupt workload changes associated with certain sensing tasks) occurring concurrently. This paper introduces a novel distributed dynamic packet scheduling framework, D²-PaS. D²-PaS is capable of processing disturbances and minimizes the number of dropped packets while ensuring that all critical events due to disturbances are handled by their deadlines. As a distributed approach, D²-PaS constructs schedules locally at individual nodes, which significantly reduces the amount of schedule-related information to be broadcast by the gateway. As a dynamic approach, D²-PaS applies a lightweight packet dropping algorithm to determine on-line at the gateway which packets can be dropped in response to disturbances and disseminate this information to the network. D²-PaS has been implemented on a multi-hop RTWN testbed to validate its applicability on hardware and a popular RTWN stack. Both testbed measurements and extensive simulation results demonstrate the effectiveness of D²-PaS.

Index Terms—Real-time wireless networks, disturbances, distributed and dynamic packet scheduling.

1 INTRODUCTION

Real-time wireless networks (RTWNs) are fundamental to many Internet-of-Things (IoT) applications in a broad range of fields such as military, civil infrastructure and industrial automation [1]–[3]. An RTWN typically consists of spatially distributed sensors, actuators, relay nodes and a gateway, where the components communicate over wireless network media to collectively accomplish the execution of certain real-time tasks running on the RTWN. The Quality of Service (QoS) offered by an RTWN is often measured by how well it satisfies the end-to-end (from sensors via gateway to actuators) deadlines of these real-time tasks. Packet scheduling at the data link layer of RTWNs plays a critical role in achieving the desired QoS, and it becomes more challenging as the network size increases. The fact that most RTWNs must deal with dynamic events further aggravate the problem.

Dynamic events can be due to unexpected changes in network resource supply (e.g., because of node and link failure, interference, etc.) or in network resource demand caused by external disturbances in the environment being monitored and controlled (e.g., detection of an intruder, sudden pressure change). When a certain sensor node detects an external disturbance, the workload associated to this sensor node needs to be changed for a certain time duration to accommodate such on-line workload changes in RTWNs becomes a critical problem in recent IoT applications. Many centralized dynamic scheduling approaches have been proposed in the literature, but most of them are designed for handling changes in network resource supply (e.g., [4]–[6]). Studies on addressing external disturbances, the focus of this paper, are relatively few. In the rest of the paper, we simply refer to external disturbance as disturbance.

In this paper, we introduce a novel distributed dynamic packet scheduling framework, referred to as D²-PaS, to handle concurrent disturbances in RTWNs. As a distributed approach, D²-PaS allows individual device node to construct its own schedule so as to significantly reduce the amount of schedule-related information to be broadcast by the gateway when disturbances are detected. As a dynamic approach, D²-PaS determines on-line at the gateway which packets can be dropped in response to disturbances and broadcast minimal amount of information to the rest of the nodes in the RTWN. To ensure that such a dynamic distributed approach is effective and efficient, we have designed a lightweight (in terms of memory usage and computing capability) scheduler to be used at each node. Furthermore, we have developed a low-complexity algorithm to be executed by the gateway to determine a short dynamic schedule interval in which a minimum number of packets will be dropped to handle the disturbances.

We have implemented D²-PaS on a real-time wireless network testbed to validate the applicability of the proposed framework. We also evaluated the performance of D²-PaS through both extensive simulation studies and testbed based experiments. When handling a single disturbance, more frequently monitor the environment. Thus, how to

1. An earlier version of the paper appeared in [7].
compared to the state-of-the-art OLS approach in [8]. D²-PaS on average reduces the packet drop rate from 69% to 4% and in the best case from 99% to 9%. D²-PaS achieves 100% success in meeting the deadlines of the critical packets while OLS only achieves 90% success on average and 62% success in the worst case. In terms of computation overhead, D²-PaS can be 2000X to 10,000X faster than OLS. More significant performance improvement can be observed when handling concurrent disturbances.

The main differences compared to the conference version are summarized as follows. i) We consider a more general disturbance model and extend the proposed D²-PaS framework to support handling concurrent disturbances; ii) more detailed analysis of the D²-PaS framework is provided including the description on how to incrementally compute and store local schedules at each device node and details of the proposed packet dropping heuristic algorithm; iii) to validate the correctness of D²-PaS, new experiments are performed by deploying a representative task set on a real RTWN testbed; iv) new simulation-based performance evaluations are added to demonstrate the effectiveness and efficiency of D²-PaS in handling concurrent disturbances.

The rest of this paper is organized as follows. The related work is discussed in Section 2. Section 3 describes the system model and gives an overview of the proposed D²-PaS framework. We first describe the local schedule generation mechanism of D²-PaS in Section 4. We then introduce the problem formulation and present the algorithm to handle a single disturbance in Section 5. In Section 6, we further generalize D²-PaS to handle concurrent disturbances. Section 7 describes the implementation and experimental studies of D²-PaS on a real-time wireless network testbed. In Section 8, we summarize the performance evaluation of D²-PaS on large-scale RTWNs through extensive simulation studies. Section 9 concludes the paper and discusses the future work.

2 RELATED WORK

Static packet scheduling for RTWNs has been well studied in the literature (e.g. [9]–[11]). Static approaches support deterministic real-time communication, but are unsuitable for handling disturbances. A number of centralized dynamic scheduling approaches for handling network changes have been proposed. The approach in [12] stores a predetermined number of link layer schedules in the system and chooses the appropriate one when disturbances are detected. However, this approach is either incapable of handling arbitrary disturbances or needs to make some approximation. Both [13] and [14] 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 [15] 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.

Extensive research efforts have been devoted on modeling and responding to disturbances in real-time systems. [16] proposes sampling rate adjustments to improve the performance of event-triggered control systems. Rate-adaptive and rhythmic task models which allow tasks to change periods and relative deadlines are introduced in [17] and [18], respectively. Schedulability analysis for these models based on the Rate Monotonic (RM) [19] and Earliest Deadline First (EDF) [20] policies are also presented in [18] and [17]. Although task models such as rhythmic and rate adaptive can be employed in RTWNs to capture unexpected disturbances, their associated scheduling approaches and schedulability analyses are only suitable for single processor platforms but cannot be directly applied in RTWNs where end-to-end timing constraints on the packet delivery need to be considered. This is the limitation of the existing work and motivates us to explore novel scheduling approach(es) for handling disturbances in RTWNs.

In recent years, a number of algorithms have been designed for packet scheduling in Time Slotted Channel Hopping (TSCH) networks, in both centralized (e.g. [21]–[23]) and distributed manner (e.g. [24]–[26]). Most of those approaches, however, assume static network topologies and fixed network traffic which limit their applications in dynamic networks. To overcome this drawback, [27] proposes Orchestra, a distributed scheduling solution that schedules packet transmissions in TSCH networks to support real-time applications. In Orchestra, each node computes its own schedule, without relying on any central scheduling entity, which makes it adaptable to dynamic network topology. Orchestra, however, does not consider real-time constraint, i.e., ignores the hard deadlines associated with tasks running in the network. It only provides best effort but no guarantee on the end-to-end latency of each task.

OLS in [8] is a state-of-the-art dynamic approach to handling disturbances in RTWNs. OLS is a centralized framework built on the rhythmic task model and relies on a dynamic programming based approach at the gateway to generate a dynamic schedule for the network to handle the disturbance. OLS has been shown to be effective for small-scale RTWNs. However, for larger RTWNs, OLS will incur significant computational overhead and may even not be able to find any feasible schedule. Moreover, OLS may drop more periodic packets than necessary due to the limited size of broadcast packets in RTWNs. All these drawbacks lead to degraded system performance when disturbances occur.

3 PRELIMINARIES

In this section, we first describe the system model and then give an overview of the proposed D²-PaS framework.

3.1 System Model

We adopt the system architecture of a typical RTWN, in which multiple sensor nodes (S-nodes) and actuator nodes (A-nodes) are wirelessly connected to a single gateway (G-node) directly or through relay nodes (R-nodes). We assume that both S-nodes and A-nodes have routing capability and they each are equipped with a single omni-directional antenna to operate on a single channel in half-duplex mode. The network is described by a directed graph \( G = (V, E) \), where the node set \( V = \{V_0, V_1, \ldots, V_g\} \). \( V_g \) represents the G-node and the rest are referred to the device nodes (all other types of nodes). A direct link \((V_i, V_j) \in E\) represents
Fig. 1. An example RTWN with 3 unicast tasks and 1 broadcast task.

<table>
<thead>
<tr>
<th>Task</th>
<th>Routing Path</th>
<th>$P_i$</th>
<th>$D_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_0$</td>
<td>$V_0 \rightarrow V_g \rightarrow V_4$</td>
<td>10</td>
<td>9</td>
</tr>
<tr>
<td>$\tau_1$</td>
<td>$V_2 \rightarrow V_0 \rightarrow V_6$</td>
<td>10</td>
<td>8</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>$V_1 \rightarrow V_0 \rightarrow V_3 \rightarrow V_5$</td>
<td>10</td>
<td>7</td>
</tr>
<tr>
<td>$\tau_3$</td>
<td>$V_g \rightarrow *$</td>
<td>10</td>
<td>10</td>
</tr>
</tbody>
</table>

Table 1: Task parameters for the example RTWN

In this paper, a reliable link from node $V_i$ to node $V_j$. (We assume that all links are reliable in this paper. Handling unreliable links can leverage existing methods and is left for future work.) $V_g$ connects all device nodes to control logic. It also contains a network manager which is responsible for network configuration and resource allocation.

For simplicity, we assume that the system runs a fixed set of tasks $\mathcal{T} = \{\tau_0, \tau_1, \ldots, \tau_m\}$. $\tau_i$ ($0 \leq i \leq n$) is a unicast task following a pre-determined single routing path with $H_i$ hops. It periodically takes a sample at node $V_i$, generates a packet to forward the sampled data to the G-node, and delivers the generated control message to the respective A-node. $\tau_{n+1}$ is a broadcast task. It runs on the G-node and is responsible for disseminating the updated schedule information in the network by following a pre-determined broadcast graph [9].

When a disturbance is detected and propagated to the gateway, $V_g$, and the corresponding task enters its rhythmic state, the system switches to the rhythmic mode to handle the disturbance. As long as no other disturbance occurs, the system returns to the nominal mode when the disturbance is completely handled, typically some time after the task has returned to the nominal state (see Fig. 2). To simplify the notation, in the system rhythmic mode, we refer to the task entering its rhythmic state as rhythmic task and any other task as periodic task which remains in its nominal state [3].

Since disturbances may cause catastrophe to the system, the rhythmic tasks have hard deadlines when the system is in the rhythmic mode while periodic tasks can tolerate occasional deadline misses. In the following, we first assume that at any time during the system operation, there is at most one task in the rhythmic state (denote it as $\tau_0$). We will generalize the system model to allow multiple tasks entering their rhythmic states concurrently in Section 6.

Each task $\tau_i$ consists of an infinite sequence of instances. The $k$-th instance of $\tau_i$ referred to as packet $\chi_{i,k}$ is associated with release time $r_{i,k}$, deadline $d_{i,k}$ and finish time $f_{i,k}$ [1]. Without loss of generality, we assume that the rhythmic task $\tau_0$ enters the rhythmic state at $t_{0,m+1}$ (denoted as $t_{n-r}$) and returns to the nominal state at $t_{0,m+R_0+1}$ (denoted as $t_{r-n}$). Thus, $\tau_0$ stays in the rhythmic state during $[t_{n-r}, t_{r-n})$, where $t_{n-r}$ and $t_{r-n}$ satisfy $t_{r-n} = t_{n-r} + \sum_{i=1}^{R_0} P_{0,i}$. Any packet of $\tau_0$ released in the system rhythmic mode is referred to as rhythmic packet, correspondingly denoted as $\chi_{0,k}$, while the packets of any periodic task $\tau_i$ ($1 \leq i \leq n + 1$) are periodic packets. The delivery of packet $\chi_{i,k}$ at the $h$-th hop is referred to as a transmission denoted as $\chi_{i,k}(h)$ ($1 \leq h \leq H_i$). Note that, there is no deadline constraint associated with intermediate transmissions on the routing path but only the last transmission indicating the end-to-end deadline for the packet.

Following industrial practice, we assume a Time Division Multiple Access (TDMA) and the broadcast task has only one state and keeps following a pre-determined period, we refer to periodic tasks as containing both the unicast periodic tasks and the broadcast task if not clarified.

2. Task $\tau_3$ is a broadcast task with 2 hops. The first hop is from $V_g$ to $V_0, V_1, V_2, V_3, V_4, V_5$. The second hop is from $V_5$ to $V_5$. The broadcast task in RTWNs is schedule-based and will not flood the network.

4. For the sake of clarification, we claim that $d_{i,k} = r_{i,k} + D_i$ for each packet $\chi_{i,k}$ and it meets its deadline if and only if $f_{i,k} \leq d_{i,k}$.
Multiple Access (TDMA) data link layer in our model. 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. In this paper, we consider a network-wide interference model without allowing any spatial reuse. That is, at most one transmission can be transmitted within any time slot during the network operation. Table 2 summarizes the notation frequently used in this paper.

Based on the description of the system model, at the highest level, we aim to solve the following problem: Given a RTWN, design a packet scheduling framework that satisfies (i) all packets can be delivered by their respective deadlines when there are no disturbances, (ii) all rhythmic packets are delivered by their rhythmic deadlines and minimum number of periodic packets are dropped when disturbances occur, and (iii) the system can safely return to the nominal mode with all packets meeting their nominal deadlines after all disturbances are properly handled.

It has been shown through a motivational example in [7] that centralized approaches such as OLS have two major drawbacks when solving the above problem. First, they suffer from the limit imposed on the maximum number of allowed updated slots (NUT) in the dynamic schedule for handling the disturbance. This is because centralized approaches require the gateway to construct a temporary schedule for the network in the rhythmic mode and broadcast the differences between the dynamic and static schedules to each device node. These updated slots must be piggybacked to a broadcast packet and propagated to all nodes in the RTWN. Since the payload size of a broadcast packet is always bounded, NUT in centralized approaches is limited. If the NUT exceeds the payload size of a broadcast packet, more periodic packets have to be dropped. Secondly, centralized approaches tend to incur high latency for generating dynamic schedules, and the latency grows quickly as the size of the RTWN increases. If the updated schedule cannot be generated before the next broadcast slot, the current rhythmic event cannot be handled properly. We propose a new approach to address these drawbacks.

### 3.2 Overview of the D²-PaS framework

We observe that in a centralized approach the restriction on NUT is mainly due to the choice that the gateway undertakes all the work to handle disturbances while other device nodes only need to update their own schedules according to the slot update information received from the gateway. Such an approach implicitly assumes that device nodes have no local computing capability, which however is not true for RTWNs nowadays. For example, the CC2538 SoC on which we run the OpenWSN stack and implement the D²-PaS framework, has potential to do more computation than just running the OpenWSN stack. Although these device nodes cannot perform as complex computations as the gateway does, they can afford some basic computations, such as schedule construction. Therefore, we propose to leverage such local computing capability and design a distributed

Algorithm 1 D²-PaS Framework

1: while true do
2: Every node generates and follows its local schedule.
3: if a disturbance is detected by a sensor node then
4: The sensor node sends a disturbance report to $V_g$.
5: $V_g$ checks the schedulability of the system after $t_{\text{dist}}$.
6: $V_g$ calculates the time interval of the system rhythmic mode.
7: if the system is overloaded then
8: $V_g$ determines the periodic packets to be dropped.
9: end if
10: $V_g$ propagates the rhythmic task information and dropped packet set to all nodes.
11: end if
12: end while

Alg. 1 gives an overview of our distributed dynamic packet scheduling framework, D²-PaS. Key steps in D²-PaS will be elaborated in the next three sections. D²-PaS works as follows. After system initialization, when a broadcast packet is received from the gateway, each node generates a schedule for a specific time interval according to a designated scheduling policy and then follows the schedule to operate. We adopt EDF [20] as the scheduling policy on each node. To generate such a schedule, each node uses the task and routing information to determine for each time slot whether it should send/receive a particular packet or stay idle. This can be done by simply following the EDF simulation process. Since every node maintains the same task and routing information, the schedules generated at different nodes are consistent.

When disturbance is detected, the S-node sends a rhythmic event request via $\tau_0$ (following the current schedule) to the gateway. Upon receiving the request at time $t'$ (see Fig. 3), $V_g$ first checks the schedulability of the system assuming $\tau_0$ will be in the rhythmic state. If the system is overloaded, the gateway determines the time duration of the system rhythmic mode and the dropped periodic packets to guarantee the deadlines of all rhythmic packets. $V_g$ then piggybacks the start time of the duration ($t_{sp}$ in Fig. 3) and the task information about dropped packets and the rhythmic task (task ID with corresponding $\overrightarrow{\bar{P}}$) to a broadcast packet and disseminates it to all nodes in the network at time $t''$. Otherwise, only $t_{sp}$ and the rhythmic task information needs to be broadcast. Thus, instead of broadcasting the entire updated schedule, D²-PaS only piggybacks the indices of the packets to be dropped to the broadcast packet when the system is overloaded. Upon receiving such broadcast information at or before $t_{sp}$, each node generates its local schedule accordingly using the updated information and the system enters the rhythmic mode at the start point $t_{sp}$.

In order to ensure D²-PaS works properly, we need to tackle a few challenges. First, ideally, each node could construct and store the entire schedule for the hyperperiod
Definition 1 (Local Busy Slot) A time slot is a local busy slot for node \(V_j\) if \(V_j\) either sends or receives a transmission belonging to any unicast task in the time slot.

7. By global schedule, we mean the schedule table of all transmissions of tasks running in the network. Since we consider a single channel network, a single packet transmission in an RTWN is equivalent to a job executing for one time unit on a CPU. Therefore, a global schedule is computed as if all tasks are running on a single processor.

3 LOCAL SCHEDULE GENERATION

This section focuses on addressing the challenges of local schedule generation at device nodes. The basic idea is to incrementally construct the global schedule \(G\) of the whole network and store the portion of the schedule that it involves in its local schedule during run-time. Specifically, we follow two design principles when constructing the schedule at each node: (i) construct one segment of the entire EDF schedule at a time to avoid generating the whole schedule at each node; (ii) perform local schedule generation in the idle slots of each node. The questions that need to be answered include (1) what these segments should be, (2) what is the upper bound on the lengths of these segments, and (3) what need to be maintained from one segment to the next to support the incremental computation. Below we first introduce a few definitions and then present our answers.

**Definition 1 (Local Busy Slot)** A time slot is a local busy slot for node \(V_j\) if \(V_j\) either sends or receives a transmission belonging to any unicast task in the time slot.

We design \(D^2\)-PaS such that each node \(V_j\) generates its local schedule incrementally according to the definition of \(SS_j\). A local schedule is constructed by simply following the EDF policy to determine that each time slot in \(SS_j\) should send/receive which packet or be idle. Generating a schedule segment must be completed within a local idle slot or a broadcast slot to guarantee that \(D^2\)-PaS works properly. Since no acknowledgement process is needed in a broadcast slot in IEEE 802.15.4e, the data link layer activities will not exceed 8 ms in a broadcast slot. Thus, the rest of the time slot is sufficient for local schedule construction as measured in Section 7.2. Using the broadcast slot guarantees that each schedule segment is computed according to the up-to-date system status information since the current segment ends before any broadcast slot in which a disturbance propagation may happen. The next segment starts from the broadcast slot and the computation is performed within this broadcast slot after extracting the up-to-date system information. This approach automatically satisfies the schedule design principle (ii) as it uses local idle slots to derive the schedule.

**Definition 2 (Local Idle Slot)** A time slot is a local idle slot for node \(V_j\) if it is not a local busy slot for \(V_j\).

**Definition 3 (Schedule Segment)** A schedule segment, denoted as \(SS_j\), is a segment of the local schedule constructed by node \(V_j\) at a time. \(SS_j\) starts either at each time slot when \(V_j\) receives a broadcast packet, or at the first local idle slot after \(V_j\) completes a sequence of consecutive local busy slots.

7 By global schedule, we mean the schedule table of all transmissions of tasks running in the network. Since we consider a single channel network, a single packet transmission in an RTWN is equivalent to a job executing for one time unit on a CPU. Therefore, a global schedule is computed as if all tasks are running on a single processor.
mines the computation time for constructing $SS_j$, we only need to find the maximum length of consecutive local busy slots. Since the gateway can handle complex computations, we only consider the length of $SS_j$ on device nodes. Lemma 1 and Theorem 1 below specify the upper bound on any $SS_j$ as a function of the number of tasks passing through $V_j$.

Lemma 1 If the system is schedulable, i.e., each packet completes all its transmissions before the deadline, for any device node $V_j$, and task $\tau_i$ passing through $V_j$, there exists at least one local idle slot at $V_j$ among any three consecutive transmissions of $\tau_i$ passing $V_j$.

Proof: Any three consecutive transmissions from task $\tau_i$ passing $V_j$ can only be either (i) from three different packets of $\tau_i$ or (ii) from two different packets of $\tau_i$. In case (i), suppose $V_j$ finishes transmitting $\chi_{i,k}$. If there were no local idle slot at $V_j$ before $V_j$ works on a transmission from subsequent $\chi_{i,k+1}$, $\tau_i$ would only have one hop from source ($V_j$) to the destination or from the predecessor to actuator ($V_j$). (Otherwise, $V_j$ would have at least one local idle slot for another transmission of $\chi_{i,k}$ not involving $V_j$.) This means that either $\tau_i$ does not pass through G-node or $\tau_i$ has no A-node or S-node. This contradicts our system model. For case (ii), assume that two of the three transmissions are an incoming and outgoing transmission of $\chi_{i,k}$, and the third one is the incoming (or outgoing) transmission of $\chi_{i,k+1}$ (or $\chi_{i,k-1}$). If no local idle slot exists at $V_j$ among these three consecutive transmissions, it indicates that task $\tau_i$ only has two hops on its path from the source (S-node) to the destination (A-node). (Otherwise, there must exist a local idle slot at $V_j$ when packet $\chi_{i,k}$ is transmitted on a third hop that passes $V_j$.) In this case, $V_j$ must be the gateway since according to our system model, each task must pass through G-node. This, however, contradicts the assumption that $V_j$ is a device node.

Theorem 1 If the system is schedulable, the maximum length of a schedule segment $SS_j$ at any device node $V_j$ is $2 \times n_j$, where $n_j$ is the number of tasks passing through $V_j$.

Proof: We first consider the case in which the length of $SS_j$ equals to $2 \times n_j$. We use $\tau_1, \ldots, \tau_{n_j}$ to denote all tasks passing through $V_j$. Suppose $V_j$ is the second node on the paths of all these tasks and the deadlines decrease from $\tau_1$ to $\tau_{n_j}$, i.e., $d_1 > \ldots > d_{n_j}$. Assume packets are released following the pattern shown in Fig. 5. According to EDF, each packet requires two time slots on $V_j$ (for receiving and sending) and is then preempted by a higher priority packet. In this case, the length of $SS_j$ equals to $2 \times n_j$.

Next we prove by contradiction that the length of $SS_j$ cannot be larger than $2 \times n_j$. Assume that $V_j$ is busy for more than $2 \times n_j$ slots consecutively. The number of tasks passing through $V_j$ is $n_j$, indicating that at least one task requires more than two time slots on $V_j$ within $SS_j$. By Lemma 1, there exists at least one local idle slot on $V_j$ among the transmissions. This contradicts our assumption.

To facilitate local schedule computation, each local node needs to maintain certain data. First, the generated schedule must be stored. To save memory, only the information associated with the slots (e.g., slot 1, 2, 11 and 12 in Fig. 4) is stored. Second, to generate the local schedule, $V_j$ must have the full knowledge of task time parameters, including $P_i$, $D_i$, and $H_i$ in the nominal state. Furthermore, since $V_j$ only needs to generate the local schedule involving itself, $V_j$ does not need to store the entire route information of all tasks but only those portions that traverse through itself.

These information at each node is organized into a schedule table to assist local schedule computation. Table 3 shows the schedule table maintained at node $V_j$ at time slot 0 for the example RTWN in Section 3. Here $H_i$, $P_i$, and $D_i$ are as defined in Table 1. Tuple $(I_i, s_i, r_i)$ captures the routing information of task $\tau_i$’s transmissions related to $V_j$. $I_i$ indicates that $V_j$ is the receiver of the $I_i$-th transmission of $\tau_i$ (and thus $V_j$ is the sender of the $(I_i+1)$-th transmission of $\tau_i$). Note that $I_i = 0$ indicates that $V_j$ is the source (S-node) of $\tau_i$ and $I_i = H_i$ indicates that $V_j$ is the destination (A-node) of $\tau_i$. $s_i$ and $r_i$ denote the sender and receiver of transmissions $\chi_{i,I}$ and $\chi_{i,I+1}$, respectively. The last column stores vector $(c_i, p_i)$ representing the execution status of $\tau_i$ at the time slot when the currently generated schedule ends. As expected, the task information needs to be updated dynamically when $V_j$ completes each computation of its schedule. Specifically, $c_i$ denotes the remaining number of hops to its destination (A-node) and $p_i$ indicates that the last packet activated at the end of the current $SS_j$ is the $p_i$-th packet of $\tau_i$. With such a properly maintained schedule table, local schedules can be efficiently computed and stored in our $D^2$-PaS framework. Alg. 2 presents the details on how node $V_j$ generates schedule segment $SS_j$.

5 Scheduling in the Rhythmic Mode

The above local schedule generation approach can work properly when the system is in the nominal mode. However, when the system enters the rhythmic mode, the rhythmic task adopts new periods and deadlines. Some periodic packets may have to be dropped to ensure all rhythmic packets delivered by their deadlines. The information in the schedule table would not be sufficient for each node to generate a consistent local schedule. Though information such as all possible rhythmic task parameters can be stored
In summary we aim to solve the following problem:

**P1:** Given task set \( \mathcal{T}, t_{n-1,r}, t_{ep}, t_{ep}^u \) and \( \Delta_d \), determine \( t_{ep} \) and the set of dropped packets such that objective function (1) is achieved and all the three constraints above are satisfied.

Below, we first discuss how to decide the end point and then describe the packet dropping algorithm in D²-PaS.

### 5.2 Selecting End Point of System Rhythmic Mode

When a disturbance occurs, the duration of the system rhythmic mode should be as short as possible so that the system can promptly return to its nominal mode. However, a shorter system rhythmic mode can lead to worse performance since we may need to drop more periodic packets to properly handle the disturbance. That is, the choice of end point \( t_{ep} \) impacts not only the length of the system rhythmic mode but also the number of dropped periodic packets. Thus, a well-designed end-point selection method is required. The concept of \( [t_{sp}, t_{ep}] \) is also used in OLS [8]. In OLS, \( t_{ep} \) (referred to as switch point) must be aligned with a nominal release time since the centralized approach needs to reuse the static schedule when the system returns to the nominal mode. In D²-PaS, since each node generates its own local schedule, this requirement is no longer needed, which opens up possibilities for dropping fewer periodic packets. Details of the new end-point selection strategy can be found in our previous work [7]. In the following, we only give a high-level description of how we efficiently select the end point \( t_{ep} \) and present some important theorems and definitions. Theorem 2 below summarizes the sufficient conditions for a feasible end point.

**Theorem 2** \( t_{ep} \) is a feasible end point if the following conditions are satisfied.

**Condition 1** \( t_{ep} \geq t_{f,0,m+R_0} \), where \( t_{f,0,m+R_0} \) is the finish time of the last packet released in \( \tau_0 \)'s rhythmic state.

**Condition 2** All rhythmic packets \( \chi_{0,k} \) released earlier than \( t_{ep} \) must be finished by their rhythmic and nominal deadlines in \( \tau_0 \)'s rhythmic state and nominal state, respectively.

**Condition 3** All periodic packets \( \chi_{r,k} \) released earlier than \( t_{ep} \) must be either finished by \( \min(d_{i,k}, t_{ep}) \) or dropped.

Theoretically, any time slot within \( [t_{f,0,m+R_0}, t_{ep}^u] \) that satisfies conditions in Theorem 2 can be an end point. However, selecting the time slot in this set that leads to the minimum number of dropped packets can be very time consuming. To tackle this challenge, we observe that using any time slot between two successive release times as \( t_{ep} \) leads to more dropped packets than just using the later release time as \( t_{ep} \) (Lemma 2 in our previous work [7]). Thus, we can reduce the search space by only examining the release times of all packets in \( [t_{f,0,m+R_0}, t_{ep}^u] \). Let \( \Gamma(t_{ep}) \) denote the end-point candidate set containing all these feasible release times, denoted as \( t_{ep}^c \). We have

\[
\Gamma(t_{ep}) = \{ \forall r_{i,k} \in [t_{f,0,m+R_0} + H_0, t_{ep}^u] \setminus \bigcup \{r_{0,k'}, r_{0,k'}^r + H_0\} \}
\]

where \( 0 \leq i \leq n + 1, m + R_0 + 1 \leq k' \leq m + R_0 + q \),

\[
(2)
\]

in which \( \chi_{0,m+R_0+q} \) denotes the last packet of \( \tau_0 \) released before \( t_{ep}^u \) (see Fig. 2).
Algorithm 3 Determining End Point of Rhythmic Mode
1. Construct EDF schedule for all packets released before \( t_{ep} \);
2. \( S \leftarrow \{ t_{np} \mid \min(f_{0,m+R_{0}, d_{0,m+R_{0}}} \leq t_{np} \leq u_{i,k}) \}; \)
3. if \( S \neq \emptyset \) then
   4. \( t_{np} \leftarrow \min(S) \);
5. if no deadline misses before \( t_{np} \) then
   6. return \{NULL\};
7. end if
8. return \( \Gamma(t_{np}) \);
9. end if
10. Construct \( \Gamma(t_{cp}) \) according to Equation (2);
11. return \( \Gamma(t_{cp}); \)

Performing schedulability test and running the packet dropping algorithm on every end-point candidate can still be time consuming and often unnecessary. Instead, we explore to identify the end point directly by leveraging the concept of carry-over-packet and no-carry-over-packet point defined as follows.

Definition 4 (Carry-Over Packet) A carry-over packet at time \( t \) is a packet released before \( t \) with a deadline after \( t \).

Definition 5 (No-Carry-Over Packet (NCoP) Point) A time slot \( t_{np} \), is an NCoP point if and only if all carry-over packets at \( t_{np} \) are finished before \( t_{np} \) under EDF \(^{31}\).

The definition of NCoP point \( t_{np} \) guarantees that there is no workload carried over into the system after \( t_{np} \). This is consistent with Condition 3 of being a feasible end point. Furthermore, Theorem 3 below states that the earliest NCoP point satisfying Condition 1 & 2 is an end point that yields the fewest number of dropped packets.

Theorem 3 The first NCoP point after \( \min(f_{0,m+R_{0}}, d_{0,m+R_{0}}) \) yields the minimum number of dropped periodic packets among all end-point candidates in \( \Gamma(t_{cp}) \).

Combining Equation (2) for \( \Gamma(t_{cp}) \) and Theorem 3, we have Alg. 3 (executed by the gateway) for determining the end point of the system rhythmic mode.

5.3 Determining Dropped Packets

The problem of minimizing the number of late jobs is well studied \([28]-[30]\) in real-time scheduling literature. In this subsection, we show how our problem of determining the minimum number of dropped packets can be mapped to the problem of minimizing the number of late jobs, which can be solved in polynomial time using Lawler’s algorithm \([28]\).

The late-job minimization scheduling problem is categorized into different classes according to the processor environment, the constraints associated with the jobs and the scheduling criteria. These three parameters are described using notation \( C_1|C_2|C_3 \) \([31]\). Following this notation, our packet dropping problem can be reduced to the problem, \( P_2 \) \((1 \mid pmtn, r_j \mid \sum w_j \times L_j) \), a problem studied in \([28]\).

Definition 6 (P2) Given a set of jobs \( \tau_j \), with each job having release time \( r_j \), execution time \( e_j \), deadline \( d_j \) and weight \( w_j \), find a preemptive schedule on a single processor such that the sum of the late job’s weight is minimized. A job is on time \((L_j = 0)\) if it completes before its deadline, otherwise it is a late one \((L_j = 1)\).

11. If a packet misses its deadline, its unfinished part is dropped but the finished part is kept.

To construct the mapping, we first introduce the concept of active packet for each end-point candidate \( t_{ep} \in \Gamma(t_{cp}) \).

Definition 7 (Active Packet) \( \chi_{i,k} \) is an active packet for time slot \( t \) if and only if at least one of the two conditions: (1) \( t_{sp} \leq r_{i,k} < t \), and (2) \( t < t_{sp} \leq d_{i,k} \leq t_{cp} \).

Now let \( \Psi(t_{cp}) \) be the active packet set containing all active packets to be scheduled within \( t_{sp}, t_{cp} \). Based on Definition 7 \( \Psi(t_{cp}) \) contains the following packets: (i) \( \chi_{i,k} \), \( s \leq t_{sp} \leq r_{i,k} < t_{cp} \); (ii) Carry-over packet at start point \( s \), i.e., \( \chi_{i,k} \), \( s \leq t_{sp} \leq r_{i,k} < t_{cp} \), and \( d_{i,k} > t_{cp} \). The release times of these packets are set to \( t_{sp} \) and their execution times are set to their remaining number of hops by \( t_{sp} \). (iii) Carry-over packet at end point \( t_{ep} \), i.e., \( \chi_{i,k} \), \( s \leq t_{sp} \leq r_{i,k} < t_{cp} \) and \( d_{i,k} > t_{cp} \). According to Condition 2 and in Theorem 2, the deadlines of all carry-over packets at end point \( t_{ep} \) are set to \( t_{ep} \) and their execution times are set to \( H_i \). Note that, since the broadcast task is responsible for disseminating updated information to all nodes in the network, ideally any broadcast packet should not be dropped. However, if scheduling some broadcast packet would induce a deadline miss of the rhythmic task, the broadcast packet must be dropped to guarantee the disturbance can be properly handled. Thus, to ensure that any broadcast packet would be dropped only if it is absolutely necessary, we set the priority of the broadcast task to be higher than any other periodic task when determining the dropped packet set.

The packet dropping problem is mapped to \( P_2 \) as follows. Each packet \( \chi_{i,k} \) in \( \Psi(t_{cp}) \) with release time \( r_{i,k} \), deadline \( d_{i,k} \) and number of hops \( H_i \) is mapped to a job with the same release time, deadline and execution time. (Note that a job executing for one time unit on a CPU is equivalent to a single packet transmission in an RTWN.) Let \( n_p \) and \( n_b \) denote the number of periodic packets and the number of broadcast packets in \( \Psi(t_{cp}) \), respectively. The weight of different types of packets are set as follows. (i) The weight of each periodic packet is set to 1. (ii) The weight of each broadcast packet is set to \( n_p + 1 \). (iii) The weight of each rhythmic packet is set to \( n_p + n_b \cdot (n_p + 1) + 1 \). Setting the weight of any broadcast packet to be larger than the sum of the weights of all periodic packets ensures that dropping any broadcast packet is more expensive than dropping all periodic packets. Furthermore, the weight of each rhythmic packet is determined in a way that no rhythmic packet will be dropped before the system drops all periodic packets. Since the execution time of any rhythmic packet is less than or equal to its deadline, no rhythmic packet would be dropped after all the periodic packets are dropped, which satisfies the definition of \( P_2 \). It follows that a schedule for the job set that minimizes the weighted sum of the late jobs is also a feasible schedule that minimizes the number of dropped periodic packets for \( \Psi(t_{cp}) \). Using Lawler’s method in \([28]\), such a schedule can be found in \( O(n_p^5) \) time where \( n_p \) is the number of packets in \( \Psi(t_{cp}) \).

Lawler’s method with a computational complexity of \( O(n_p^5) \) can be rather costly since the size of the active packet set for each \( t_{cp} \) can be large and the number of \( t_{ep} \) may also be not small. It is thus desirable to find a more efficient packet dropping algorithm. A key observation is that, with the same utilization, a task having shorter execution time releases more packets within a fixed time interval than a
Algorithm 4 Packet Dropping using our Heuristic

Input: $\Psi(t_{ep})$
Output: $\rho(t_{sp}, t_{ep})$

1: $SPS \leftarrow \{\text{all rhythmic packets in } \Psi(t_{ep})\}$; // scheduled packet set
2: $\Psi(t_{ep}) \leftarrow \Psi(t_{ep}) \setminus SPS$;
3: $\rho(t_{sp}, t_{ep}) \leftarrow \emptyset$;
4: while $\Psi(t_{ep}) \neq \emptyset$ do
5: if Broadcast packet exists in $\Psi(t_{ep})$ then
6: Add the broadcast packet $\chi_s$ with the shortest execution time in $\Psi(t_{ep})$ to $SPS$;
7: else
8: Add the unicast packet $\chi_s$ with the shortest execution time in $\Psi(t_{ep})$ to $SPS$;
9: end if
10: $\Psi(t_{ep}) \leftarrow \Psi(t_{ep}) \setminus \{\chi_s\}$;
11: if $SPS$ is schedulable under EDF then
12: Continue;
13: end if
14: $SPS \leftarrow SPS \setminus \{\chi_s\}$;
15: $\rho(t_{sp}, t_{ep}) \leftarrow \rho(t_{sp}, t_{ep}) \cup \{\chi_s\}$;
16: end while
17: return $\rho(t_{sp}, t_{ep})$;

Algorithm 5 Determining Dropped Packets at Gateway

Input: $\Gamma(t_{ep})$
Output: $\rho(t_{sp}, t_{ep})$

1: for $(\forall \chi_{ep} \in \Gamma(t_{ep}))$ do
2: Construct $\Psi(t_{ep})$; // active packet set
3: Generate a dropped periodic packet set $\rho(t_{sp}, t_{ep})$ based on $\Psi(t_{ep})$ using Lawler’s algorithm or our heuristic;
4: $S \leftarrow S \cup \{\rho(t_{sp}, t_{ep})\}$;
5: end for
6: return arg min $\rho(t_{sp}, t_{ep}) \in S | \rho(t_{sp}, t_{ep})$;

6 Supporting Concurrent Disturbances

In the previous sections, we have assumed that at any given time, the system has at most one task in the rhythmic state. This assumption simplifies the D$^2$-PaS framework for handling disturbances. However, in real-world RTWNs, multiple unexpected disturbances may happen simultaneously, or the intervals of the rhythmic mode may overlap.

In this section, we generalize the system model to allow multiple tasks to be in their rhythmic states simultaneously and extend D$^2$-PaS to handle concurrent rhythmic events. Since concurrent rhythmic events all have high priority, they compete for network resources. Thus, we cannot simply use the packet dropping scheme for periodic packets as elaborated in Section 5.3. Note that, the high-level D$^2$-PaS framework can still be applied but we need to tackle two specific challenges: (i) when and how to compute and update the dynamic schedule in the presence of multiple unexpected disturbances, and (ii) how to guarantee the system to be fail safe when multiple tasks exist in their rhythmic states at the same time.

Below, we use two concurrent rhythmic events to illustrate how we address the two challenges above. The presented technique can be easily extended to handle an arbitrary number of concurrent disturbances. To simplify the discussion, we group the relative positions of two concurrent rhythmic events into three cases, according to their arrival times at $V_g$ and the start point of the system rhythmic mode induced by each rhythmic event. Fig. 6 illustrates the three cases for task $t_1$ and $t_2$.

13. In Case 1, $t'_1$ cannot be equal to $t'_2$ since the gateway can receive at most one rhythmic event request within one slot. Similarly, in Case 2 & 3, $t'_2$ cannot be equal to $t'_1$ since the gateway cannot receive a rhythmic event request and send a broadcast packet in a same slot.

14. In Case 1, we do not distinguish notation of each disturbance, i.e., $t'' = \{t'_1 \text{ or } t'_2\}$, $t_{sp} = \{t_{sp}^1 \text{ or } t_{sp}^2\}$, and $t_{ep} = \{t_{ep}^1 \text{ or } t_{ep}^2\}$.
event has not been broadcast yet, $V_g$ can simply recalculate the new schedule based on both $\tau_1$’s and $\tau_2$’s rhythmic events. Specifically, $V_g$ determines a common $[t_{sp}, t_{ep}]$ window, computes the dropped packet set to guarantee the deadlines of both rhythmic tasks within $[t_{sp}, t_{ep}]$ and broadcasts the information to the network at $t''$. Note that, to avoid redundant computation, instead of determining dropped packets upon receiving any rhythmic event request, $V_g$ waits till the last time slot before the upcoming broadcast slot (i.e., $t'' - 1$) to calculate the dropped packet set so that it handles all detected disturbances at the same time. This principle is also adopted in the other two cases (Given the powerful computing capability of the gateway, even when a rhythmic event request arrives at $V_g$ right at $t'' - 1$, the remaining idle time of the slot should still be sufficient for $V_g$ to complete the computation.).

Case 2: $t'' > t_1$ and $t'' > t_2$ (Fig. 6(b)). The rhythmic event request from $\tau_2$ arrives at $V_g$ after $\tau_1$’s dropped packet information has been broadcast, and there is no overlap between the rhythmic windows of the two rhythmic events. In this case, $V_g$ simply handles the two disturbances separately, i.e., generating the dropped packet sets for $\tau_1$ and $\tau_2$ at $t'' - 1$ and $t'' - 1$, respectively.

Case 3: $t'' > t_1$ and $t'' > t_2$ and $t_{sp} < t_{sp}' < t_{ep}$ (Fig. 6(c)). The rhythmic windows of $\tau_1$ and $\tau_2$ overlap. Since the system has started using $\tau_1$’s dynamic schedule when $\tau_2$’s rhythmic event request arrives at $V_g$, $\tau_2$’s dynamic schedule computation needs to take $\tau_1$’s dynamic schedule into consideration because $\tau_1$’s dropped packet information, i.e., $\rho_1[t_{sp}, t_{ep}]$, can affect both the selection of end point $t_{ep}$ and the construction of active packet set $\Psi_2(t_{sp})$. To deal with this complex case, we modify the $D^2$-PaS framework following the high-level steps in Alg. 1.

(i) At Line 4 in Alg. 1, $V_g$ checks if the system is overloaded and determines the end point of the system rhythmic mode induced by $\tau_2$’s rhythmic event, i.e., $t_{ep}$. As shown in Fig. 6(c), the system follows $\tau_1$’s dynamic schedule within $[t_{sp}, t_{ep}]$ and then uses the updated schedule after $t_{sp}$. Thus, when running Alg. 3 to determine $t_{sp}'$, the system should be made to construct the EDF schedule of the system during $[t_{sp}, t_{ep}]$: (a) Starting from $t_{sp}$, $\tau_1$ adopts its rhythmic periods and deadlines; and (b) for any dropped packet $\chi_{i,k}$ in $\rho_1[t_{sp}, t_{ep}]$, $\chi_{i,k}$ is dropped if $r_{i,k} < t_{sp}$. Otherwise, $\chi_{i,k}$ is kept since the schedule after $t_{sp}'$ will be updated.

(ii) If the system is overloaded, $V_g$ determines the periodic packets to be dropped within $[t_{sp}', t_{ep}]$ (Line 5-6 in Alg. 1). According to Line 2 in Alg. 5, $V_g$ constructs the active packet set $\Psi_2(t_{sp})$. Similarly, considering $\tau_1$’s dropped packet information, any carry-over packet at $t_{sp}'$ in $\rho_1[t_{sp}, t_{ep}]$ should be removed from $\Psi_2(t_{sp})$ and all packets released after $t_{sp}'$ must be included in $\Psi_2(t_{sp})$ even if they are dropped to handle the rhythmic event of $\tau_1$.

(iii) Based on $\Psi_2(t_{sp}')$, $V_g$ runs Alg. 4 to determine the dropped packet set within $[t_{sp}', t_{ep}]$ to guarantee the deadlines of all rhythmic packets of both $\tau_1$ and $\tau_2$. Here, we refer to rhythmic packet of task $\tau_i$, as any packet released in the rhythmic window of $\tau_i$, i.e., $[t_{sp}, t_{ep}]$.

With the above extension to $D^2$-PaS, we addressed the question of when and how to update the dynamic schedule to handle newly arriving rhythmic event from $\tau_2$. Next we consider how to guarantee the system to be always fail safe during the time when $\tau_1$ and $\tau_2$ are both in the rhythmic state, i.e., $[t_{sp}, t_{ep}]$ in Case 1 and $[t_{sp}', t_{ep}]$ in Case 3. By fail safe, we mean that the rhythmic packets of a task are guaranteed to meet their deadlines regardless of the states of other packets.

Under the assumption that at most one rhythmic task is present, the deadlines of rhythmic packets can always be guaranteed if all periodic packets are dropped since the deadline of each task is larger than or equal to its execution time in both the nominal and rhythmic states, i.e., $H_i \leq \min(D_i, D_{i,r}) |x = 1, \ldots, R_i|$. Hence, the system is fail safe under $D^2$-PaS. However, if multiple tasks enter their rhythmic states concurrently, the system can be overloaded even if all periodic packets are dropped. To tackle this problem, we consider how to delay the time of a task entering its rhythmic state and the delay decision is made by $V_g$ in a First-Come-First-Serve (FCFS) manner for all disturbances to be handled. (This is similar to an admission control policy. Other alternative priority schemes may be adopted as well.) Specifically, suppose multiple rhythmic event requests arrive at $V_g$ at $t'' - 1$ (see Fig. 6(a)). $V_g$ sorts all the disturbances by their occurrences and determines $t_{n-\tau}$ for each rhythmic task sequentially. If $V_g$ finds that allowing any task, suppose $\tau_0$, to switch into its rhythmic state from the first release time after $t_{ep}$ (i.e., $t_{n-\tau} = \tau_0_m+1$ as shown in Fig. 2) would lead to rhythmic deadline violation even if all periodic packets are dropped, $V_g$ delays $t_{n-\tau}$ to $\tau_0$’s next nominal release time $\tau_{0,m+2}$ and checks the feasibility of the system. This process repeats until a release time of $\tau_0$ can be set as $t_{n-\tau}$. The gateway then determines $t_{n-\tau}$ for the next rhythmic task in the queue.

Note that, delaying a task from entering its rhythmic state will increase the latency for handling the delayed rhythmic task. However, such delay can avoid catastrophe to the system since guaranteeing the deadlines of all rhythmic packets is the first priority. Summarizing the discussions above, we have Alg. 6 (executed at $V_g$) for handling two concurrent disturbances and it can be readily extended to handle an arbitrary number of concurrent disturbances.
Algorithm 6 Handling Two Concurrent Disturbances

1: \(At t_2' - 1\), a rhythmic event request from \(\tau_2\) is to be handled;

2: \(if t_1' < t_2' \leq t'' - 1\) (Case 1) then

3: Handle both disturbances together;

4: end if

5: \(if t_2' > t_1'\) then

6: Determine \(t_{sp}\) according to the upcoming broadcast packet;

7: \(if t_{sp} \geq t_1'(Case 2)\) then

8: Handle \(\tau_2\) independently;

9: end if

10: \(if t_{sp} < t_2' < t_1'(Case 3)\) then

11: while the system cannot admit \(\tau_2\) to enter its rhythmic state at \(t_{2'}\) do

12: \(t_{2'} = P_2\);

13: end while

14: Handle \(\tau_2\) with \(\tau_1\)'s rhythmic information considered;

15: end if

16: end if

7 Testbed Implementation and Evaluation

We have implemented \(D^2\)-PaS on a RTWN testbed to validate its applicability in real-world RTWNs. Our testbed is based on OpenWSN [33] with required enhancements to support \(D^2\)-PaS. As shown in Fig. 7(a), an OpenWSN network consists of multiple OpenWSN devices, an OpenWSN Root, and an OpenLBR (Open Low-Power Border Router). The Root and OpenLBR communicate through a wired connection (e.g., UART), using OpenBridge protocol. They together form the gateway. As shown in Fig. 7(b), our testbed consists of 7 wireless devices (TI CC2538 SoC + SmartRF06 evaluation board). Among these 7 devices, one is configured as the Root and connected to OpenLBR which runs on a Linux machine. The others are configured as device nodes and form a multi-hop RTWN. An 8-channel logic analyzer is used to verify the system operation.

7.2 Testbed Experiments

In our testbed experiments, we first validate the functional correctness of the \(D^2\)-PaS implementation at the device level, and then at the system level on our multi-hop RTWN.

7.2.1 Software stack validation

The functional correctness of the \(D^2\)-PaS implementation on the OpenWSN stack is validated through logging and analyzing the debug information generated from the device nodes during the experiments. We used two tools (Segger RTT Viewer and SystemView) to retrieve two types of debug information, procedure check point and time measurement.

Procedure check point: To verify that the device node runs correctly following the specified task information, we programmed each node to generate a log when it encounters an event. These events include: 1) active slot event with the associated absolute slot number (ASN) and slot type, 2) TX/RX events from both MAC and OpenBridge, 3) EDF-LS execution event with the computed active slots, and 4) packet release event from P-Gen. We used RTT Viewer to collect these events from individual nodes and record them into a log file. We also developed a system-wide log analyzer to process these logged events. This tool utilizes the ASN information associated with the events to align them into a system-wide log, and display the events generated at the same ASN from different nodes as the output. By analyzing the log, we confirmed the following observations:

If a packet needs to be transmitted in the next slot, P-Gen prepares the sensor and prepares the packet. A broadcast packet, however, is initiated by the gateway in a broadcast slot instead of by P-Gen, and forwarded by the nodes according to a calculated broadcast graph.

Receiver binds to a UDP port to receive packets from the Manager. It handles three types of messages: 1) the link info message to install broadcast slots, 2) the task info message for the P-Gen and EDF-LS modules to construct periodic packets and schedule segments, respectively, in the nominal mode, and 3) the rhythmic event response message for the P-Gen and EDF-LS modules to construct rhythmic packets and schedule segments, respectively, in the rhythmic mode. Manager is responsible for installing the broadcast graph in the network and initializing the P-Gen and EDF-LS modules in the device nodes. It also runs the end-point selection and packet dropping algorithms to handle rhythmic events. The results along with the rhythmic task information are broadcast to the device nodes for constructing local schedules.

Fig. 7. An overview of the real-time wireless network testbed.

<table>
<thead>
<tr>
<th>Algorithm 6 Handling Two Concurrent Disturbances</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: (At t_2' - 1), a rhythmic event request from (\tau_2) is to be handled;</td>
</tr>
<tr>
<td>2: (if t_1' &lt; t_2' \leq t'' - 1) (Case 1) then</td>
</tr>
<tr>
<td>3: Handle both disturbances together;</td>
</tr>
<tr>
<td>4: end if</td>
</tr>
<tr>
<td>5: (if t_2' &gt; t_1') then</td>
</tr>
<tr>
<td>6: Determine (t_{sp}) according to the upcoming broadcast packet;</td>
</tr>
<tr>
<td>7: (if t_{sp} \geq t_1'(Case 2)) then</td>
</tr>
<tr>
<td>8: Handle (\tau_2) independently;</td>
</tr>
<tr>
<td>9: end if</td>
</tr>
<tr>
<td>10: (if t_{sp} &lt; t_2' &lt; t_1'(Case 3)) then</td>
</tr>
<tr>
<td>11: while the system cannot admit (\tau_2) to enter its rhythmic state at (t_{2'}) do</td>
</tr>
<tr>
<td>12: (t_{2'} = P_2);</td>
</tr>
<tr>
<td>13: end while</td>
</tr>
<tr>
<td>14: Handle (\tau_2) with (\tau_1)'s rhythmic information considered;</td>
</tr>
<tr>
<td>15: end if</td>
</tr>
<tr>
<td>16: end if</td>
</tr>
</tbody>
</table>

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMC.2018.2877681, IEEE Transactions on Mobile Computing
7.2.2 Test case validation

To demonstrate the correctness and effectiveness of D²-PaS in handling concurrent rhythmic events, we deployed a representative task set on the 7-node multi-hop RTWN in Fig. 7(b). In this subsection, we describe the operation of D²-PaS in handling two concurrent rhythmic events (and omit the simpler case of handling single rhythmic events). Task parameters are given in Table 4. The radio activities of each device are indicated by toggles of a physical pin monitored by the 8-channel Logic Analyzer.

A sample of recorded waveforms is shown in Fig. 8. The waveforms capture the activities of the 7 devices for two consecutive slotframes. The network runs in the nominal mode during the 1st slotframe (Fig. 8(a)) and switches to the rhythmic mode during the 2nd slotframe (Fig. 8(b)). In Fig. 8(a), a falling or a rising edge in the Slots channel marks the start of a new slot, and the 7 waveforms below show the radio activities (either transmit or receive) for the respective 7 nodes, as all labeled at the left panel. We set the first slot in each slotframe to be a shared slot for network management traffic, and the following 4 slots to be used for network-wide broadcast, referred to as reserved slots. Starting from the 6th slot, tasks are transmitted according to the static schedule or a dynamic schedule generated by D²-PaS.

In the experiment, we set $t_{sp0}$ to be at the beginning of the 2nd slotframe and set $t_{sp1}$ to be at $\tau_1$'s 2nd nominal release time in the same slotframe (see Fig. 8). As shown by the release times of $\tau_0$ and $\tau_1$ labeled in Fig. 8, both $\tau_0$ and $\tau_1$ enter their rhythmic states and adopt their rhythmic periods (deadlines) at $t_{sp0}$ and $t_{sp1}$, respectively. To accommodate the increased rhythmic workload, a dynamic schedule is generated by D²-PaS in which 3 periodic packets from $\tau_2$ are dropped, as evidenced by the 3 missing (dashed) light blue lines in Fig. 8(b). This exactly matches the result from our simulation and demonstrates that D²-PaS operates correctly on the testbed and handles disturbances properly.

8 Simulation-based Evaluation

In this section, we focus on evaluating the performance of the end-point selection method and the packet dropping
algorithm in the rhythmic mode of D^2-PaS. Since extensive studies need to be performed on the scheduling methods under various parameter settings, we conduct the performance evaluation through simulation.

In our simulation studies, we compare D^2-PaS with OLS [8], the state of the art which has been compared with a baseline algorithm and showed clear advantages. As OLS does not natively support handling concurrent rhythmic events, we first consider the task model that only one rhythmic task is present at any time during the network operation. We then extend OLS to support concurrent rhythmic events, and compare its performance with D^2-PaS.

8.1 Simulation Setup

To better control workload, we vary the nominal utilization of the task set deployed on the simulated RTWNs. Specifically, we use random periodic task sets generated according to a target nominal utilization $U^*$. Each task set is generated by incrementally adding random periodic tasks to an initially empty set $T^*$. Each periodic task $t_i$ is generated based on the following parameter settings: (i) the number of hops $H_i$ follows a uniform distribution in $[2, 3, \ldots, 10]$, and (ii) period $P_i$ is equal to deadline $D_i$ and follows a uniform distribution in $\{15, 16, \ldots, 50\}$ (Note that the unit of $P_i$ and $D_i$ values is intentionally left not specified since only their relative values are important to schedulability study. The ranges of the parameters are chosen to reflect realistic RTWN applications.). After a task set is generated, we randomly select one task to be the rhythmic task $t_0$. The period vector, $P_0 = \{P_0\} = \{D_0\}$, is generated with the following parameters: (i) the initial rhythmic period ratio, $\gamma_0 = P_0/D_0$, is fixed to 0.2, and (ii) the number of elements in $P_0$, $R_0$, can be any integer in the set of $\{4, 6, \ldots, 16\}$. We fixed $\gamma$ and used $R_0$ to tune the workload of the rhythmic task during the rhythmic mode, because $R_0$ provides better control on changing the workload of the rhythmic task. We assume that upon entering the system rhythmic mode, rhythmic task period $P_{0,k}$ first drops from $P_0$ to $P_{0,1}$ according to $\gamma$, and then gradually increases back to $P_0$ following the pattern of $P_{0,k} (1 \leq k \leq R_0) = \{P_0 \times (\gamma + (k-1) \times \frac{R_0-1}{R_0})\}$. Additional parameters of both D^2-PaS and OLS are set as follows. (i) Start point $t_{sp}$ is drawn from the uniform distribution over $\{50, \ldots, 200\}$. (ii) Switch point scaling factor $\alpha$ is set to 2 (same value as used in [8] for comparing with other approaches). $\alpha$ determines the switch point upper bound in OLS where $t_{sw} = t_{sp} \times (\alpha - 1) \times P_0[8]$. To ensure fair comparison, we also set the upper bound on the end point in $D^2$-PaS as $t_{ep} = t_{sw}$. (iii) The payload of a broadcast packet is set to 90 bytes. To represent one updated slot, OLS needs 3 bytes where 13, 7 and 4 bits are used for slot ID, task ID and hop index, respectively. This indicates that the maximum allowed number of updated slots $\Delta^u$ in OLS is 30. Instead, in D^2-PaS, we only need 2 bytes to represent one dropped packet, where 7 and 9 bits are used for task ID and packet ID, respectively. Thus the maximum allowed number of dropped packets in D^2-PaS is $\Delta^d = 45$. (iv) Another parameter of OLS is the maximum allowed number of maintained child schedules in dynamic programming, denoted as $\beta$. For fair comparison, we set $\beta = 40$ which is one of the best values of $\beta$ for the performance of OLS [8].

8.2 Handling a Single Rhythmic Event

We compare the performance of D^2-PaS and OLS on handling a single rhythmic event using three metrics: (i) acceptance ratio (AR), defined as the fraction of feasible task sets over all the task set, (ii) average drop rate (DR), defined as the ratio between the number of dropped periodic packets and the total number of active packets within $[t_{sp}, t_{ep}]$, and (iii) computation overhead (CO), defined as the time taken for handling one rhythmic event request. A task set is feasible if the deadlines of all rhythmic packets are satisfied in the system rhythmic mode. Since the deadline of a single rhythmic task can always be guaranteed if dropping all periodic packets, the only case may make a rhythmic packet miss its deadline is the selection of an unfeasible end point.

Fig. 10(a) shows AR as a function of $R_0$ (the number of elements in $P_0$) for the task sets generated with a nominal utilization $U^*$ of 0.5. Other utilization levels have the similar behavior and are omitted. Each data point is the average results of 1,000 trials. We can observe from Fig. 10(a) that OLS cannot always find a feasible switch point to guarantee the timing requirements of all rhythmic packets. Its AR is around 90% but can be as low as 60%[16]. On the other hand, D^2-PaS can achieve 100% AR for all data points since a feasible end point always exists in D^2-PaS.

Fig. 10(b) shows DR as a function of the nominal utilization $U^*$ under different settings of $R_0$. Dash curves denote 15. Note that, since the performance of the packet dropping algorithms in both D^2-PaS and OLS depends on the task set but not network topology, we only control the generation of the task set and assume the same network topology when comparing their performance.

16. The AR curve of OLS is not monotonic, which agrees with the observation in [8]. This is due to the way that the switch point is selected in OLS. In OLS, the switch point must be aligned to a release time of the rhythmic task in its nominal state so as to reuse the static schedule. This requirement sometimes makes the rhythmic task miss its deadline even if all the periodic packets are dropped. However, no such requirement is needed in D^2-PaS.
the performance of OLS while solid curves denote that of D\textsuperscript{2}-PaS. It can be clearly observed that D\textsuperscript{2}-PaS significantly outperforms OLS under all settings. As shown in Fig. 10(b), D\textsuperscript{2}-PaS exhibits a very low drop rate (1% on average) when the nominal utilization is less than 90%. This is mainly because the increased workload introduced by one rhythmic task does not overload the system in most cases if the nominal utilization is not high. Therefore, the disturbance can be properly handled with no periodic packet being dropped. (In Section 8.3, we will show that DR of D\textsuperscript{2}-PaS increases when the number of concurrent rhythmic events increases.) However, on the other hand, the performance of OLS drops significantly with the increasing of \( R_0 \). To provide a finer granularity comparison, we also set the nominal utilization \( U^* = 0.9 \) and increase \( R_0 \) from 4 to 16 with a step size of 2. Fig. 10(c) shows that D\textsuperscript{2}-PaS performs much better than OLS, and the improvement on DR increases from 40\% (\( R_0 = 4 \)) to 95\% (\( R_0 = 16 \)).

We also compared CO of D\textsuperscript{2}-PaS and OLS for handling one rhythmic event. The execution times are from an Intel i3-2120 CPU (3.30GHz) used as the gateway. The average execution time of OLS varies from 2.2s to 10.4s when the nominal utilization is less than 90\% and increase \( R_0 \) from 4 to 16. D\textsuperscript{2}-PaS needs no more than 1ms to handle the same rhythmic event. This is mainly because OLS relies on a dynamic programming based approach which is very time consuming, we only vary \( \mathcal{N}_r \) within \{2, 3\}. (ii) Start point \( t_{SP}^i \) of the rhythmic window of \( \tau \). As discussed in Sec. 6, there are three cases for the relative positions of two concurrent disturbances (see Fig. 6). Since disturbances rarely come in a burst of time in reality (Case 1) and sequential rhythmic events are relatively easy to solve (Case 2), here we are more interested in the case when a rhythmic event overlap with partial overlaps (Case 3). Let the rhythmic window of each rhythmic event overlap with that of its previous one, i.e., \( t_{SP}^i \leq t_{SP}^{i+1} \). Specifically, the start point for handling the first occurred rhythmic event is drawn from the uniform distribution over \{50, \ldots, 200\}. The start point of the rhythmic window of each following rhythmic event \( t_{SP}^i \) is randomly picked from \{\( t_{SP}^{i+1} + 1, \ldots, t_{SP}^{i+1} - 1 \)\}.

Fig. 11(a) shows AR as a function of the number of rhythmic periods, \( R_{\tau}^{\mathcal{N}_r} \) under different settings of \( \mathcal{N}_r \), for the task sets generated with a nominal utilization \( U^* = 0.9 \). Since both OLS and D\textsuperscript{2}-PaS can delay the entrance of rhythmic tasks to avoid the system being overloaded, the only case that may make a rhythmic packet miss its deadline is still the selection of an unfeasible end point. As shown in Fig. 11(a), D\textsuperscript{2}-PaS can achieve 100\% AR since a feasible end point always exists in D\textsuperscript{2}-PaS. However, OLS cannot always find a feasible switch point to guarantee the task set to be feasible. Furthermore, the AR of OLS decreases when the number of concurrent rhythmic events becomes larger since OLS is getting more difficult to find a feasible switch point.

Fig. 11(b) depicts DR as a function of \( U^* \) under different settings of \( \mathcal{N}_r \). Since we are more interested to explore how DR changes when the No. of concurrent rhythmic events increases, we fix the No. of rhythmic periods of each rhythmic task, i.e., \( R_t = 4 \). As observed in Fig. 11(b), increasing the No. of concurrent rhythmic events leads to degraded performance for both OLS and D\textsuperscript{2}-PaS. However, D\textsuperscript{2}-PaS degrades slower (from 7\% to 8\% on average) than OLS (from 61\% to 74\% on average) and still significantly outperforms OLS under all settings of \( \mathcal{N}_r \).

Due to the high computation overhead of OLS, we only evaluate the trend of DR for D\textsuperscript{2}-PaS when the number of

### Table 5

<table>
<thead>
<tr>
<th>( \mathcal{N}_r )</th>
<th>DR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.7%</td>
</tr>
<tr>
<td>2</td>
<td>11.3%</td>
</tr>
<tr>
<td>3</td>
<td>15.4%</td>
</tr>
<tr>
<td>4</td>
<td>17.9%</td>
</tr>
<tr>
<td>5</td>
<td>18.6%</td>
</tr>
</tbody>
</table>

8.3 Handling Concurrent Rhythmic Events

In this section, we evaluate the performance of D\textsuperscript{2}-PaS for handling concurrent disturbances. For fair comparison, we extend OLS to support concurrent rhythmic events. Specifically, since OLS is a centralized approach relying on the gateway to undertake all the work to handle disturbances, any time when a new rhythmic event arrives, the gateway only needs to regenerate an updated schedule and disseminate it to the network. Same as D\textsuperscript{2}-PaS, all disturbances are handled in FCFS manner as discussed in Section 6.

Besides the parameters used in the previous simulation studies, several additional parameters are used in random task set generation: (i) Number of concurrent rhythmic events, denoted as \( \mathcal{N}_r \). Since OLS relies on a dynamic
concurrent disturbances keeps increasing. As discussed in Section 6, there are three cases for the relative positions of two concurrent disturbances (see Fig. 6). Since disturbances rarely come in a burst manner (Case 1), i.e., many disturbances are detected by sensors from anywhere simultaneously, and sequential rhythmic events are relatively easy to solve (Case 2), in the simulation we focus on the case when different rhythmic event windows have partial overlaps (Case 3). We set the rhythmic window of each disturbance to be partially overlapped with the previous and succeeding ones (except the first and last ones) with $R_i = 4$ and $U^* = 0.9$. As shown in Table 5, DR keeps increasing when $N_r$ increases from 1 to 5 but the increasing speed slows down. The reason is that though partially overlapped rhythmic events lead to more dropped packets, the increase in the number of dropped packets is smaller than the increase in the total number of packets. Thus the resulting packet drop rate increases at a slower speed as the number of concurrent disturbances increases.

9 Conclusion and Future Work

In this paper, we introduce D²-PaS, a distributed dynamic packet scheduling framework, to handle external disturbances in RTWNs. Different from a centralized packet scheduling approach where dynamic schedules are generated in the gateway and disseminated to all nodes, D²-PaS leverages the computing capability on device nodes to generate schedules locally. The gateway only executes a lightweight packet dropping algorithm and disseminates the information of dropped packets if the system is overloaded. Our extensive simulation studies and testbed experiments demonstrate the applicability and effectiveness of D²-PaS in real-world RTWNs.

In this work, we assume all links are reliable in the network. How to ensure reliable end-to-end packet transmissions in RTWN (with given reliability requirements) has been extensively studied in the literature [34]–[37]. Acknowledged broadcast mechanisms [38] have also been developed to ensure the reliability of the broadcast messages. These techniques can be readily employed in D²-PaS for the static schedule construction. For the dynamic schedule construction, if we do not differentiate transmission and retransmission slots allocated for a given link, the developed packet dropping algorithm can still be applied. Once a packet is decided to be dropped in the dynamic schedule, all its associated transmissions and retransmissions along its routing path will be dropped as well. This will lead to a feasible but not optimal schedule to compensate the disturbances. As the future work, we will consider specific static schedule construction methods, and design optimal dynamic schedule construction methods.

Another important future work is to extend D²-PaS in multi-channel RTWNs allowing spatial reuse and explore fully distributed framework which does not rely on any centralized control point in the network.

10 Acknowledgement

This work is supported in part by the NSF of China under Grant No. 61472072 and 61528202, US NSF under Grant No. CNS-1319904 and CNS-1647209 and the collaborative innovation center of major machine manufacturing in Liaoning.

References


