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

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, D^2-PaS. D^2-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, D^2-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, D^2-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. D^2-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 D^2-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 more frequently monitor the environment. Thus, how 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 D^2-PaS, to handle concurrent disturbances in RTWNs.[1] As a distributed approach, D^2-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, D^2-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 D^2-PaS on a real-time wireless network testbed to validate the applicability of the proposed framework. We also evaluated the performance of D^2-PaS through both extensive simulation studies and testbed based experiments. When handling a single disturbance,

- *T. Zhang and Q. Deng are with the Northeastern University, Shenyang 110819, China. Email: tyzhang@stumail.neu.edu.cn, dengqx@mail.neu.edu.cn. Part of the work by T. Zhang was performed during his visit at University of Notre Dame.*
- *G. Tao and S. Han are with the University of Connecticut, Storrs, CT, 06269. Email: {tao.gong,song.han}@uconn.edu.*
- *X. S. Hu is with the University of Notre Dame, Notre Dame, IN, 46556. Email: shu@nd.edu.*

The first two authors have equal contribution to this work.

1. An earlier version of the paper appeared in [7].

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

compared to the state-of-the-art OLS approach in [8], D^2-PaS on average reduces the packet drop rate from 69% to 4% and in the best case from 99% to 9%. D^2-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, D^2-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 D^2-PaS framework to support handling concurrent disturbances; ii) more detailed analysis of the D^2-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 D^2-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 D^2-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 D^2-PaS framework. We first describe the local schedule generation mechanism of D^2-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 D^2-PaS to handle concurrent disturbances. Section 7 describes the implementation and experimental studies of D^2-PaS on a real-time wireless network testbed. In Section 8, we summarize the performance evaluation of D^2-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 adatapble 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 D^2-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 $V = \{\{V_0, V_1, \dots\}, V_g\}$. 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

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

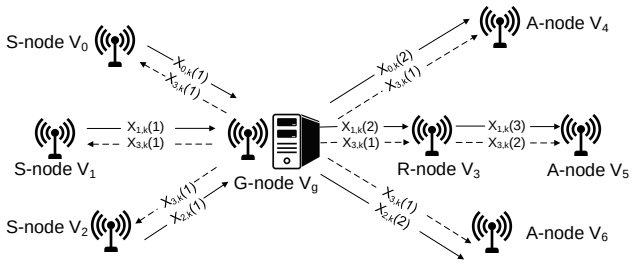


Fig. 1. An example RTWN with 3 unicast tasks and 1 broadcast task.

TABLE 1
Task parameters for the example RTWN

Task	Routing Path	P_i	D_i
τ_0	$V_0 \rightarrow V_g \rightarrow V_4$	10	9
τ_1	$V_2 \rightarrow V_g \rightarrow V_6$	10	8
τ_2	$V_1 \rightarrow V_g \rightarrow V_3 \rightarrow V_5$	10	7
τ_3	$V_g \rightarrow *$ [2]	10	10

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 $\mathcal{T} = \{\tau_0, \tau_1, \ldots, \tau_n, \tau_{n+1}\}$. τ_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 S-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. τ_{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]. Fig. 1 depicts an example RTWN with 3 unicast tasks and 1 broadcast task running on 8 nodes ($V_0, \ldots, V_6$ and V_g) where V_0, V_1 and V_2 are S-nodes, V_4, V_5 and V_6 are A-nodes, V_3 is an R-node and V_g is the G-node. Task parameters are given in Table 1.

In this paper, we define disturbance as an *unexpected* event detected by a certain sensing task and the workload associated with this sensing task needs to be changed for a certain time duration to monitor the environment and perform control actions more frequently. To capture abrupt increases in network resource demands when disturbances are detected, we adopt the rhythmic task model [18] since it is shown to be effective for handling disturbances in resilient event-triggered control systems [8]. (Note that our D^2-PaS framework is not limited to the rhythmic task model and is applicable to any task model that provides the workload changing patterns for handling disturbances.) Specifically, each unicast task τ_i has two states: *nominal state* and *rhythmic state*. In the nominal state, τ_i follows the nominal period P_i and nominal relative deadline $D_i \leq P_i$, which are all constants. When a disturbance corresponding to τ_i occurs, τ_i enters the rhythmic state in which its period and relative deadline are first reduced, in order to properly respond to

2. Task τ_3 is a broadcast task with 2 hops. The first hop is from V_g to $V_0, V_1, V_2, V_3, V_4, V_6$. The second hop is from V_3 to V_5. The broadcast task in RTWNs is schedule-based and will not flood the network.

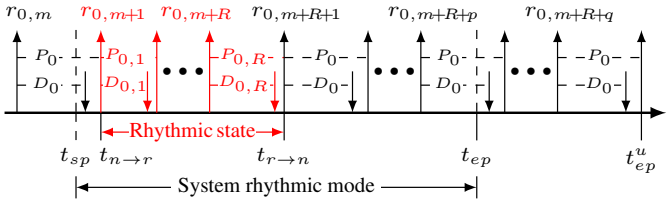


Fig. 2. Timing parameters for τ_0 and for the system in the rhythmic mode.

the disturbance, and then gradually return to their nominal values by following some monotonically non-decreasing function. We use vectors $\overrightarrow{P_i} = [P_{i,x}, x = 1, \ldots, R_i]^T$ and $\overrightarrow{D_i} = [D_{i,x}, x = 1, \ldots, R_i]^T$ to represent the periods and relative deadlines of τ_i when it is in the rhythmic state. As soon as τ_i enters the rhythmic state, its period and relative deadline adopt sequentially the values specified by $\overrightarrow{P_i}$ and $\overrightarrow{D_i}$, respectively. τ_i returns to the nominal state when it starts using the nominal period and nominal deadline again. Fig. 2 depicts the timing parameters of τ_0 as the rhythmic task.

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 τ_0). We will generalize the system model to allow multiple tasks entering their rhythmic states concurrently in Section 6.

Each task τ_i consists of an infinite sequence of instances. The k-th instance of τ_i, referred to as packet $\chi_{i,k}$, is associated with release time $r_{i,k}$, deadline $d_{i,k}$ and finish time $f_{i,k}$[4]. Without loss of generality, we assume that the rhythmic task τ_0 enters the rhythmic state at $r_{0,m+1}$ (denoted as $t_{n\rightarrow r}$) and returns to the nominal state at $r_{0,m+R_0+1}$ (denoted as $t_{r\rightarrow n}$). Thus, τ_0 stays in the rhythmic state during $[t_{n\rightarrow r}, t_{r\rightarrow n})$, where $t_{n\rightarrow r}$ and $t_{r\rightarrow n}$ satisfy $t_{r\rightarrow n} = t_{n\rightarrow r} + \sum_{x=1}^{R_0} P_{0,x}$. Any packet of τ_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 τ_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

3. Since 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.

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 $f_{i,k} \leq d_{i,k}$.

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

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. [5] 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 D^2-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 D^2-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 and dynamic scheduling framework which can achieve better performance in terms of fewer dropped packets and more feasible task sets than centralized approaches.

Algorithm 1 D^2-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_{n→r}.
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, D^2-PaS. Key steps in D^2-PaS will be elaborated in the next three sections. D^2-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 τ_0 (following the current schedule) to the gateway. Upon receiving the request at time t' (see Fig. 3)[6], V_g first checks the schedulability of the system assuming τ_0 will be in the rhythmic state. If the system is overloaded, the gateway determines the time duratoin 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 $\overrightarrow{P_i}$ and $\overrightarrow{D_i}$) 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, D^2-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 D^2-PaS works properly, we need to tackle a few challenges. First, ideally, each node could construct and store the entire schedule for the hyperperiod

5. Fragmentation can enable the transmission of a larger dynamic schedule. However, this would introduce much longer response time for handling the disturbance since multiple broadcast packets are required to propagate the entire dynamic schedule to all the nodes.

6. A system does not go to the rhythmic mode immediately after a disturbance is detected. It only enters the rhythmic mode (and τ_0 enters the rhythmic state) after each device receives the broadcast packet at the start point t_{sp}.

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

TABLE 2
Summary of important notations and definitions

Notation	Definition	Notation	Definition
$V_j, j = 0, 1, \ldots$	Device nodes (S-nodes, A-nodes and R-nodes)	$t_{n\to r}$, $t_{r\to n}$	Slot when τ_0 leaves its nominal state and its rhythmic state, respectively
V_g	Gateway (G-node)		
$\tau_i (0 \le i \le n)$ and τ_{n+1}	Unicast task and broadcast task	t_{sp}, t_{ep}, t^c_{ep}, t^u_{ep}	Start point, end point, end point candidate and end point upper bound
τ_0 and τ_i $(1 \le i \le n+1)$	Rhythmic task and periodic task (single rhythmic event)		
		t_{np}	No-carry-over-packet (NCoP) point
H_i	Number of hops of τ_i	$\Gamma(t_{ep})$	Set of end point candidates
P_i (D_i)	Nominal period (deadline) of τ_i	$\Psi(t)$	Set of active packets within $[t_{sp}, t)$
$\overrightarrow{P_i}$ ($\overrightarrow{D_i}$)	Rhythmic period (deadline) vector of τ_i	R_i	Number of elements in $\overrightarrow{P_i}$ ($\overrightarrow{D_i}$)
$\chi_{i,k}$	The k-th released packet of task τ_i	Δ^d	Maximum allowed number of dropped packets
$\chi_{i,k}(h)$	The h-th transmission of packet $\chi_{i,k}$	$\rho[t_{sp}, t)$	Set of dropped periodic packets within $[t_{sp}, t)$
$r_{i,k}$, $d_{i,k}$, $f_{i,k}$	Release time, deadline and finish time of $\chi_{i,k}$	SS_j	Schedule segment of node V_j

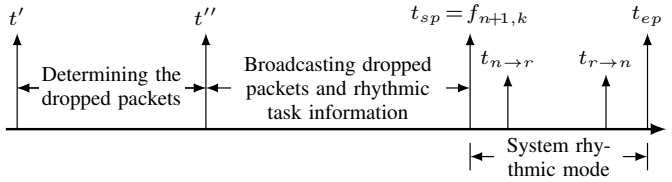


Fig. 3. Network operations after a disturbance is reported to the gateway V_g. t' denotes the time when V_g receives the rhythmic event request. t'' denotes the time when V_g sends the first hop of the broadcast packet.

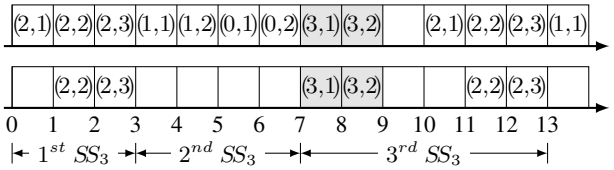


Fig. 4. Schedule segments of node V_3. The top (bottom) is the global (local) schedule of the system (V_3). Blank slots in the top (bottom) schedule are global (local) idle slots, respectively. Gray slots are assigned for transmissions of broadcast packets.

in the nominal mode. This is, however, not practical due to limited computing capability and memory on device nodes. Second, since constructing a schedule takes time, such computation should not occur when the node is supposed to send or receive packets. Third, to allow fast response to disturbances, an efficient method is needed at the gateway to determine which packets to drop. In the following, we discuss in detail how D^2-PaS solves these problems.

4 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 [7] of the whole network and store the portion of the schedule that it involves in as 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, and (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.*

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.*

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. (Note that a broadcast slot at V_j is also a local idle slot for V_j based on Def. 2.) Considering the example RTWN in Section 3, Fig. 4 shows the first three schedule segments of node V_3.

The time needed to generate SS_j depends on the length of SS_j. As shown in Fig. 4, each SS_j consists of one or more consecutive local idle slots followed by a sequence of local busy slots. Because only the number of busy slots deter-

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.

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

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 τ_i passing through V_j, there exists at least one local idle slot at V_j among any three consecutive transmissions*[8] *of τ_i passing V_j.*

Proof: Any three consecutive transmissions from task τ_i passing V_j can only be either (i) from three different packets of τ_i or (ii) from two different packets of τ_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 $\chi_{i,k+1}$, τ_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 τ_i does not pass through G-node or τ_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 τ_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 τ_1 to τ_{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

8. In between consecutive transmissions from a single task, there may be transmissions from other tasks.

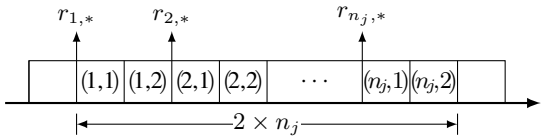


Fig. 5. An example showing the worst-case length of SS_j. $r_{i,*}$ denotes the release time of an arbitrary packet of task τ_i.

TABLE 3
The schedule table stored in node V_3 at $t = 0$.

Task	H_i	P_i	D_i	$\langle I_i, s_i, r_i \rangle$	$\langle c_i, p_i \rangle$
τ_0	2	10	9	N/A	$\langle 2, 1 \rangle$
τ_1	2	10	8	N/A	$\langle 2, 1 \rangle$
τ_2	3	10	7	$\langle 2, V_g, V_5 \rangle$	$\langle 3, 1 \rangle$
τ_3	2	10	10	N/A	$\langle 2, 1 \rangle$

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.[9] 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_3 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 $\langle I_i, s_i, r_i \rangle$ captures the routing information of task τ_i's transmissions related to V_j. I_i indicates that V_j is the receiver of the I_i-th transmission of τ_i (and thus V_j is the sender of the (I_{i+1})-th transmission of τ_i). Note that, $I_i = 0$ indicates that V_j is the source (S-node) of τ_i and $I_i = H_i$ indicates that V_j is the destination (A-node) of τ_i. s_i and r_i denote the sender and receiver of transmissions χ_{i,I_i} and χ_{i,I_i+1}, respectively. The last column stores vector $\langle c_i, p_i \rangle$ representing the execution status of τ_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 τ_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

9. Upon detection of the disturbance(s), specifications of the rhythmic task(s) are received from the gateway.

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

Algorithm 2 Schedule Segment Computation at Node V_j

Input: Schedule table stored in V_j, local idle slot t
Output: SS_j

```
SS_j ← ∅;
while true do
    Compute the assignment of slot t according to a designated scheduling policy (EDF in this paper);
    if (slot t is to receive a broadcast packet) || (t is a local idle slot and t − 1 is a local busy slot) then
        return SS_j;
    else
        if slot t is assigned to a transmission involving V_j then
            SS_j ← SS_j ⋃{t, i, h}; // i is the task ID and h is the hop index
        else
            SS_j ← SS_j ⋃{t, −1, −1};
        end if
    end if
    t ← t + 1;
end while
```

locally, the device nodes need to know which task is entering the rhythmic state and what packets should be dropped if any. In this section we present the details on how D^2-PaS handles the rhythmic event when a disturbance is detected.

5.1 Problem Formulation

When a disturbance is reported to the gateway via task τ_0, D^2-PaS needs to find an updated schedule for the system to use in the rhythmic mode such that (i) all rhythmic packets meet their deadlines, and (ii) the minimum number of periodic packets are dropped. This updated schedule should be used in the time interval defined by the start point and end point of the rhythmic mode, i.e., $[t_{sp}, t_{ep})$ (see Fig. 3). Suppose the broadcast packet $\chi_{n+1,k}$ reaches all nodes at time slot $f_{n+1,k}(H_{n+1})$. We can simply use $f_{n+1,k}(H_{n+1})$ as t_{sp}. The selection of t_{ep} however must guarantee that all packets released after it meet their nominal deadlines if no task enters the rhythmic state again. Thus the main problem to solve by the gateway is to determine (i) the end point of the rhythmic mode, t_{ep}, and (ii) which periodic packets to drop in the updated schedule. Let $\rho[t_{sp}, t_{ep})$ be the set of dropped periodic packets in $[t_{sp}, t_{ep})$. Our goal is to

$$\min |\rho[t_{sp}, t_{ep})| \tag{1}$$

subject to the following constraints.

Constraint 1 *Each rhythmic packet $\chi_{0,k}$ must meet its deadline $d_{0,k}$, i.e., $f_{0,k} \leq d_{0,k}$.*

Constraint 2 $f_{0,m+R_0} \leq t_{ep} \leq t^u_{ep}$.

Here, t^u_{ep} is a user specified parameter[10] to bound the maximum allowed latency for handling the current rhythmic event. $f_{0,m+R_0} \leq t_{ep}$ ensures that the rhythmic event can be completely handled before the system goes back to the nominal mode. Since the size of the broadcast packet is limited, we set a maximum allowed number of dropped packets in $[t_{sp}, t_{ep})$ and denote it by Δ^d. Thus we have

Constraint 3 $|\rho[t_{sp}, t_{ep})| \leq \Delta^d$.

10. t^u_{ep} is treated as absolute value to avoid introducing too many notation. It can be readily obtained by $t_{sp}+$ "the user specified upper bound on the length of the rhythmic mode duration".

In summary we aim to solve the following problem:
P1: Given task set $\mathcal{T}$, $t_{n\to r}$, t_{sp}, t^u_{ep} and Δ^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 D^2-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 D^2-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 *$t_{ep} \geq f_{0,m+R_0}$, where $f_{0,m+R_0}$ is the finish time of the last packet released in τ_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 τ_0's rhythmic state and nominal state, respectively.*

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

Theoretically, any time slot within $[f_{0,m+R_0}, t^u_{ep}]$ 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 $[f_{0,m+R_0}, t^u_{ep}]$. Let $\Gamma(t_{ep})$ denote the end-point candidate set containing all these feasible release times, denoted as t^c_{ep}. We have

$$\Gamma(t_{ep}) = \left\{ \forall r_{i,k} \in [r_{0,m+R_0} + H_0, t^u_{ep}] \setminus \bigcup (r_{0,k'}, r_{0,k'} + H_0) \right\} \quad \text{where } 0 \leq i \leq n+1, m+R_0+1 \leq k' \leq m+R_0+q, \tag{2}$$

in which $\chi_{0,m+R_0+q}$ denotes the last packet of τ_0 released before t^u_{ep} (see Fig. 2).

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

Algorithm 3 Determining End Point of Rhythmic Mode

Construct EDF schedule for all packets released before t^u_{ep};
$\mathcal{S} \leftarrow \{t_{np} \mid \min(f_{0,m+R_0}, d_{0,m+R_0}) \leq t_{np} \leq t^u_{ep}\}$;
if $\mathcal{S} \neq \emptyset$ **then**
  $t_{np} \leftarrow \min(\mathcal{S})$;
  if no deadline misses before t_{np} **then**
    return {NULL};
  end if
  return $\Gamma(t_{ep}) = \{t_{np}\}$;
end if
Construct $\Gamma(t_{ep})$ according to Equation (2);
return $\Gamma(t_{ep})$;

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*[11].

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 $\Gamma(t_{ep})$.*

Combining Equation (2) for $\Gamma(t_{ep})$ 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, **P2** $(1 \mid pmtn, r_j \mid \sum w_j \times L_j)$, a problem studied in [28].

Definition 6 (P2) *Given a set of jobs τ_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 $t^c_{ep} \in \Gamma(t_{ep})$.

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) $t_{sp} < d_{i,k} \leq t$, holds.*

Now let $\Psi(t^c_{ep})$ be the active packet set containing all active packets to be scheduled within $[t_{sp}, t^c_{ep})$. Based on Definition 7, $\Psi(t^c_{ep})$ contains the following packets: (i) $\chi_{i,k}, s.t., t_{sp} \leq r_{i,k} < d_{i,k} \leq t^c_{ep}$. (ii) Carry-over packet at start point t_{sp}, i.e., $\chi_{i,k}, s.t., r_{i,k} < t_{sp}$ and $d_{i,k} > t_{sp}$. 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^c_{ep}, i.e., $\chi_{i,k}, s.t., r_{i,k} < t^c_{ep}$ and $d_{i,k} > t^c_{ep}$. According to Condition 2 and 3 in Theorem 2, the deadlines of all carry-over packets at end point t^c_{ep} are set to t^c_{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 **P2** as follows. Each packet $\chi_{i,k}$ in $\Psi(t^c_{ep})$ 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 $\Psi(t^c_{ep})$, 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 **P2**. 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 $\Psi(t^c_{ep})$. Using Lawler's method in [28], such a schedule can be found in $O(n^5_\Psi)$ time where n_Ψ is the number of packets in $\Psi(t^c_{ep})$.

Lawler's method with a computational complexity of $O(n^5_\Psi)$ can be rather costly since the size of the active packet set for each t^c_{ep} can be large and the number of t^c_{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

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

Algorithm 4 Packet Dropping using our Heuristic

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

task having larger execution time. Since minimizing the number of dropped packets is equivalent to maximizing the number of scheduled packets, a straightforward heuristic is to schedule packets in the increasing order of their execution times in order to have a higher possibility to schedule more packets. Details of our heuristic is given in Alg. 4. Initially, all rhythmic packets in the active packet set $\Psi(t^c_{ep})$ are stored in the scheduled packet set $\mathcal{SPS}$ (Lines 1-2). To ensure timely dissemination of updated information in the network, we schedule the broadcast packets prior to all other periodic packets (Lines 5-9). Then, we iteratively select packet χ_s that has the shortest execution time and check whether it can be scheduled. If so, χ_s is kept in the scheduled packet set $\mathcal{SPS}$. Otherwise, χ_s is dropped (Lines 11-15). Finally, the dropped packet set $\rho[t_{sp}, t^c_{ep})$ is returned (Line 17). Using the proposed heuristic, a feasible schedule can be found in $O(n^2_\Psi)$ time.

Our packet dropping problem bears similarity to the imprecise-computation scheduling problem [32]. The following theorem gives the approximate ratio of our heuristic and the proof can be found in [32].

Theorem 4 *For any RTWN and given t_{sp}, t^c_{ep}, the number of dropped packets produced by our heuristic is at most twice that of an optimal schedule.*

Alg. 5 summarizes how the gateway determines which packets to drop in order to solve **P1**. For each end-point candidate found by Alg. 3, the corresponding active packet set is constructed and a packet dropping algorithm (Lawler's algorithm or our heuristic) is applied to generate a dropped periodic packet set $\rho[t_{sp}, t^c_{ep})$ (Lines 2-3). If $|\rho[t_{sp}, t^c_{ep})| \leq \Delta^d$, this dropped packet set is accepted and stored in $\mathcal{S}$. Finally, the solution with the minimum number of dropped packets in $\mathcal{S}$ is returned (Line 6). If the solution exceeds the maximum allowed number of dropped packets Δ^d, D^2-PaS uses the earliest end-point candidate in $\Gamma(t_{ep})$ as the end point and all periodic packets in the corresponding active packet set are dropped.

Algorithm 5 Determining Dropped Packets at Gateway

Input: $\Gamma(t_{ep})$
Output: $\rho[t_{sp}, t_{ep})$
1: **for** $(\forall t^c_{ep} \in \Gamma(t_{ep}))$ **do**
2: Construct $\Psi(t^c_{ep})$; // active packet set
3: Generate a dropped periodic packet set $\rho[t_{sp}, t^c_{ep})$ based on $\Psi(t^c_{ep})$ using Lawler's algorithm or our heuristic;
4: $\mathcal{S} \leftarrow \mathcal{S} \bigcup \{\rho[t_{sp}, t^c_{ep})\}$;
5: **end for**
6: **return** $\arg\min_{\rho[t_{sp}, t_{ep}) \in \mathcal{S}} |\rho[t_{sp}, t_{ep})|$;

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 τ_1 and τ_2. Without loss of generality, we assume the rhythmic event request from τ_1 arrives at the gateway first. In Fig. 6, t'_i denotes the time slot when the gateway receives the rhythmic event request from task τ_i; t''_i denotes the time slot when the gateway starts broadcasting the dropped packet set for handling τ_i; t^i_{sp} and t^i_{ep} denote the start point and end point of the system rhythmic mode induced by τ_i, respectively. We refer to $[t^i_{sp}, t^i_{ep})$ as τ_i's rhythmic window in which all packets of τ_i are rhythmic packets that cannot be dropped. It can be readily seen from Fig. 6 that all other relative positions of t'_1 and t'_2, and t^1_{sp} and t^2_{sp} can be mapped to these three cases. In the following, we discuss how we extend D^2-PaS to handle these cases.

Case 1: $t'_1 < t'_2 \leq t'' - 1$ (Fig. 6(a)). Both rhythmic event requests arrive at V_g before the upcoming broadcast slot t''[14]. Since the new schedule for handling τ_1's rhythmic

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., $t'' = \{t''_1$ or $t''_2\}$, $t_{sp} = \{t^1_{sp}$ or $t^2_{sp}\}$, and $t_{ep} = \{t^1_{ep}$ or $t^2_{ep}\}$.

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

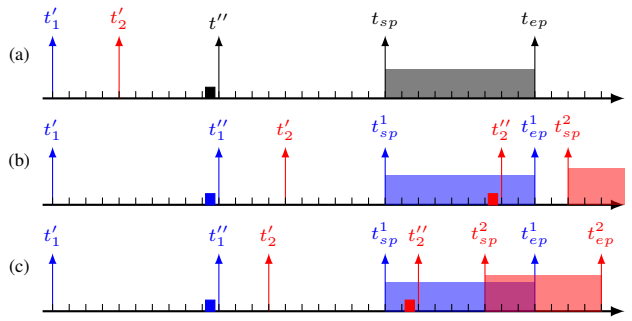


Fig. 6. (a) Case 1, $t'_1 < t'_2 \leq t'' - 1$; (b) Case 2, $t'_2 > t''_1$ and $t^2_{sp} \geq t^1_{ep}$; (c) Case 3, $t'_2 > t''_1$ and $t^1_{sp} < t^2_{sp} < t^1_{ep}$ [13]. Each block before t''_i denotes the time slot when the gateway handles the rhythmic event of τ_i. Each rectangle between t^i_{sp} and t^i_{ep} denotes the rhythmic window of τ_i. $[t''_i, t^i_{sp})$ denotes the time duration for transmitting the broadcast packet piggybacking the information for handling τ_i.

event has not been broadcast yet, V_g can simply recalculate the new schedule based on both τ_1's and τ_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: $t'_2 > t''_1$ and $t^2_{sp} \geq t^1_{ep}$ (Fig. 6(b)). The rhythmic event request from τ_2 arrives at V_g after τ_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 τ_1 and τ_2 at $t''_1 - 1$ and $t''_2 - 1$, respectively.

Case 3: $t'_2 > t''_1$ and $t^1_{sp} < t^2_{sp} < t^1_{ep}$ (Fig. 6(c)). The rhythmic windows of τ_1 and τ_2 overlap. Since the system has started using τ_1's dynamic schedule when τ_2's rhythmic event request arrives at V_g, τ_2's dynamic schedule computation needs to take τ_1's dynamic schedule into consideration because τ_1's dropped packet information, i.e., $\rho_1[t^1_{sp}, t^1_{ep})$, can affect both the selection of end point t^2_{ep} and the construction of active packet set $\Psi_2(t^2_{ep})$. 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 τ_2's rhythmic event, i.e., t^2_{ep}. As shown in Fig. 6(c), the system follows τ_1's dynamic schedule within $[t^1_{sp}, t^2_{sp})$ and then uses the updated schedule after t^2_{sp}. Thus, when running Alg. 3 to determine t^2_{sp}, two modifications should be made to construct the EDF schedule of the system during $[t^2_{sp}, t^2_{ep})$: (a) Starting from t^1_{sp}, τ_1 adopts its rhythmic periods and deadlines; and (b) for any dropped packet $\chi_{i,k}$ in $\rho_1[t^1_{sp}, t^1_{ep})$, $\chi_{i,k}$ is dropped if $r_{i,k} < t^2_{sp}$. Otherwise, $\chi_{i,k}$ is kept since the schedule after t^2_{sp} will be updated.

(ii) If the system is overloaded, V_g determines the periodic packets to be dropped within $[t^2_{sp}, t^2_{ep})$ (Line 5-6 in Alg. 1). According to Line 2 in Alg. 5, V_g constructs the active packet set $\Psi_2(t^2_{ep})$. Similarly, considering τ_1's dropped packet information, any carry-over packet at t^2_{sp} in $\rho_1[t^1_{sp}, t^1_{ep})$ should be removed from $\Psi_2(t^2_{ep})$ and all packets released after t^2_{sp} must be included in $\Psi_2(t^2_{ep})$ even if they are dropped to handle the rhythmic event of τ_1.

(iii) Based on $\Psi_2(t^2_{ep})$, V_g runs Alg. 4 to determine the dropped packet set within $[t^2_{sp}, t^2_{ep})$ to guarantee the deadlines of all rhythmic packets of both τ_1 and τ_2. Here, we refer to rhythmic packet of task τ_i as any packet released in the rhythmic window of τ_i, i.e., $[t^i_{sp}, t^i_{ep})$.

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 τ_2. Next we consider how to guarantee the system to be always fail safe during the time when τ_1 and τ_2 are both in the rhythmic state, i.e., $[t_{sp}, t_{ep})$ in Case 1 and $[t^2_{sp}, t^1_{ep})$ 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., $H_i \leq \min(D_i, D_{i,x} | x = 1, \ldots, R_i)$. 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 allow the gateway 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 $t_{n \to r}$ for each rhythmic task sequentially. If V_g finds that allowing any task, suppose τ_0, to switch into its rhythmic state from the first release time after t_{sp} (i.e., $t_{n \to r} = r_{0,m+1}$ as shown in Fig. 2) would lead to rhythmic deadline violation even if all periodic packets are dropped, V_g delays $t_{n \to r}$ to τ_0's next nominal release time $r_{0,m+2}$ and checks the feasibility of the system. This process repeats until a release time of τ_0 can be set as $t_{n \to r}$. The gateway then determines $t_{n \to r}$ 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.

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

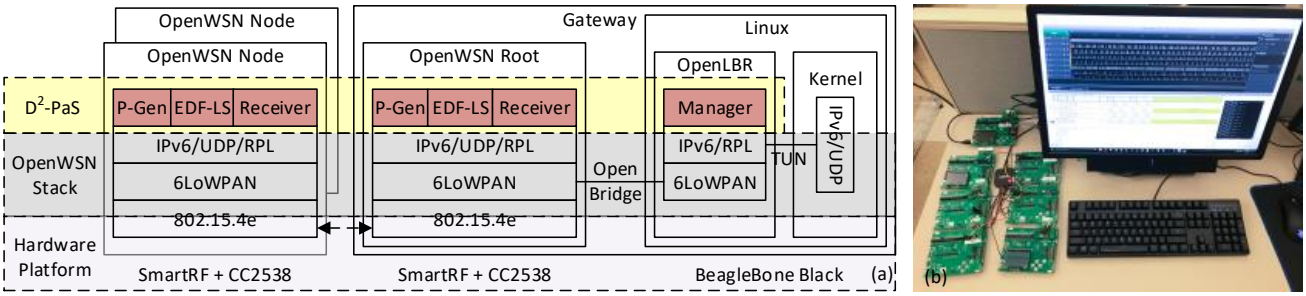


Fig. 7. An overview of the real-time wireless network testbed.

Algorithm 6 Handling Two Concurrent Disturbances

1: At $t_2''-1$, a rhythmic event request from τ_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}^2 according to the upcoming broadcast packet;
7: **if** $t_{sp}^2 \geq t_{ep}^1$ (Case 2) **then**
8: Handle τ_2 independently;
9: **end if**
10: **if** $t_{sp}^1 < t_{sp}^2 < t_{ep}^1$ (Case 3) **then**
11: **while** the system cannot admit τ_2 to enter its rhythmic state at $t_{n\to r}^2$ **do**
12: $t_{n\to r}^2 += P_2$;
13: **end while**
14: Handle τ_2 with τ_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 validate the system operation.

7.1 Software Stack Enhancement to Support D^2-PaS

To implement D^2-PaS on the testbed, as highlighted in Fig. 7(a), we added the following four software modules.

EDF Local Scheduler (EDF-LS) implements the distributed local schedule generation algorithm on the nodes. In the first idle slot of each schedule segment, it cleans up the existing link schedule, constructs the local schedule in the new schedule segment and installs them to the slotframe.

Packet Generator (P-Gen) constructs periodic packets according to its task information. It is invoked before the end of a time slot when MAC layer finishes all activities. If a packet needs to be transmitted in the next slot, P-Gen samples 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.

7.2 Testbed Experiments

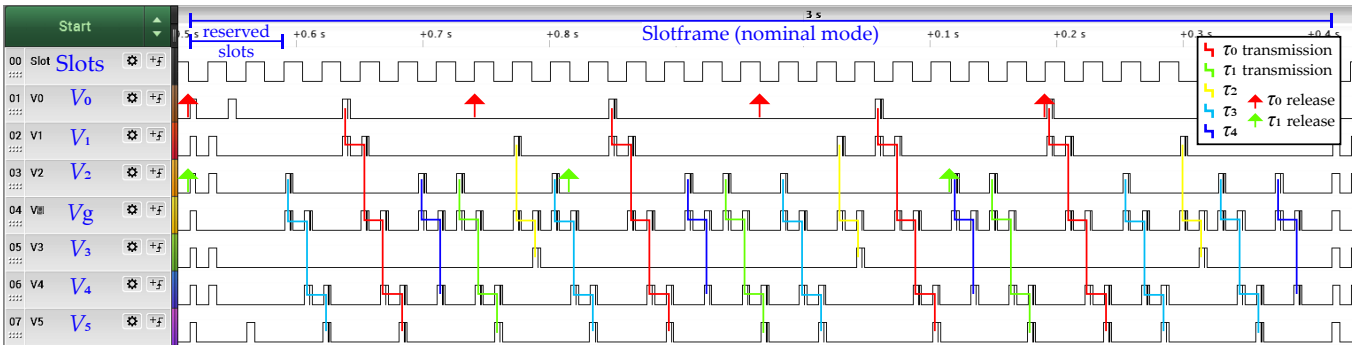
In our testbed experiments, we first validate the functional correctness of the D^2-PaS implementation at the device level, and then at the system level on our multi-hop RTWN.

7.2.1 Software stack validation

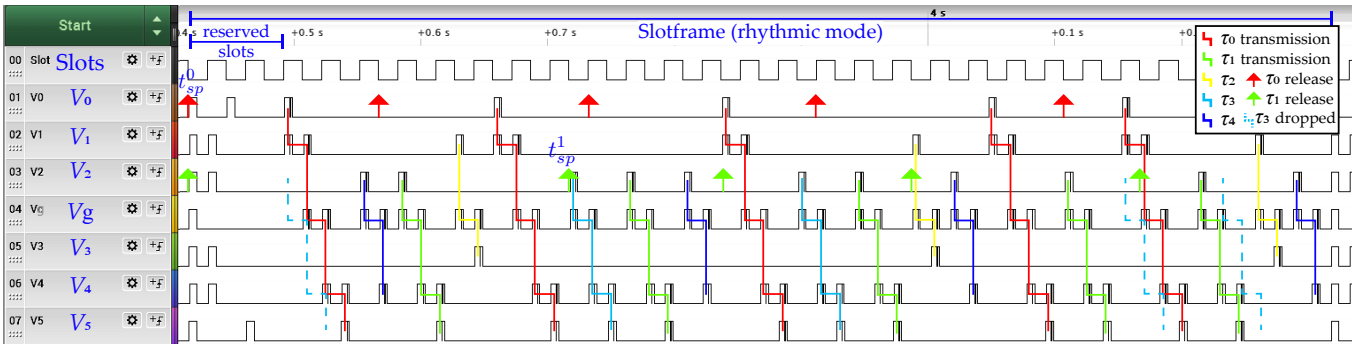
The functional correctness of the D^2-PaS implementation on the OpenWSN stack is validated through logging and analyzing the debug information generated from the device nodes during the experiments. We used two tools (Segger RTT Viewer and SystemView) to retrieve two types of debug information, procedure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

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



(a) Screen capture of the 1st slotframe (nominal mode).



(b) Screen capture of the 2nd slotframe (rhythmic mode). Dashed light blue lines indicate the dropped τ_3 packets

Fig. 8. Slot information and radio activities in the test case captured by the logic analyzer. The upper arrows show the releases of packet (only those of τ_0 and τ_1 are shown), and the connected lines show the hop-by-hop transmissions of the packet.

1) the P-Gen module releases periodic packets following the specified task information; 2) the EDF-LS module generates correct schedule segments according to our local schedule generation algorithm; and 3) the MAC layer operates correctly following the generated schedule segments.

Time measurement: Since the radio operations, schedule segment calculation, and packet generation all happen within a time slot, the timing of each operation needs to be carefully measured to ensure that all these operations can be completed within one time slot. We kept the task set (see Table 4) running in the testbed for 30 minutes and used Segger SystemView to generate a system view of the time spent on each operation in each time slot. From the measurements, we observed that in an active slot, after all radio activities, the P-Gen module can generate a packet and insert it to the MAC layer queue within 0.4ms. For the EDF-LS module, we measured the time it took to calculate and install each schedule segment by CPU-cycle stamping. The results in Fig. 9 show that the time spent on the schedule segment calculation is proportional to the size of active slots in that schedule segment. The installation time is not constant but is always less than 1ms. This gives a total time of less than 1.2ms for each execution of the EDF-LS module.

7.2.2 Test case validation

To demonstrate the correctness and effectiveness of D^2-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 D^2-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 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 D^2-PaS.

In the experiment, we set t^0_{sp} to be at the beginning of the 2nd slotframe and set t^1_{sp} to be at τ_1's 2nd nominal release time in the same slotframe (see Fig. 8). As shown by the release times of τ_0 and τ_1 labeled in Fig. 8, both τ_0 and τ_1 enter their rhythmic states and adopt their rhythmic periods (deadlines) at t^0_{sp} and t^1_{sp}, respectively. To accommodate the increased rhythmic workload, a dynamic schedule is generated by D^2-PaS in which 3 periodic packets from τ_3 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 D^2-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

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

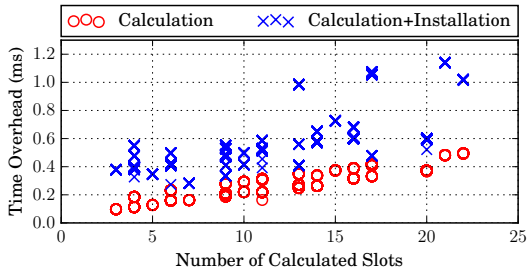


Fig. 9. Time measures of schedule segment calculation and installation.

TABLE 4
Parameters of the task set deployed in the 7-node testbed.

Task	Routing Path	$P_i(=D_i)$	$\overrightarrow{P_i}(=\overrightarrow{D_i})$
τ_0	$V_0 \rightarrow V_1 \rightarrow V_g \rightarrow V_4 \rightarrow V_5$	15	$[10, 11, 12, 13]^T$
τ_1	$V_2 \rightarrow V_g \rightarrow V_4 \rightarrow V_5$	20	$[8, 10, 12]^T$
τ_2	$V_1 \rightarrow V_g \rightarrow V_3$	20	-
τ_3	$V_2 \rightarrow V_g \rightarrow V_4 \rightarrow V_5$	12	-
τ_4	$V_2 \rightarrow V_g \rightarrow V_4$	15	-

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 $\mathcal{T}$.

Each periodic task τ_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 schedulality 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 τ_0. The period vector, $\overrightarrow{P}_0$ ($\overrightarrow{P}_0 = \overrightarrow{D}_0$), is generated with the following parameters: (i) the initial rhythmic period ratio, $\gamma = P_{0,1}/P_0$, is fixed to 0.2, and (ii) the number of elements in $\overrightarrow{P}_0$, R_0, can be any integer in the set of $\{4, 6, \ldots, 16\}$. We fixed γ 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 γ, and then gradually increases back to P_0 following the pattern of $P_{0,k}\ (1 \leq k \leq R_0) = \lfloor P_0 \times (\gamma + (k-1) \times \frac{1-\gamma}{R_0}) \rfloor$.

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 α is set to 2 (same value as used in [8] for comparing with other approaches). α determines the switch point upper bound in OLS where $t^u_{sw} = t_{r \rightarrow n} + (\alpha - 1) \times P_0$ [8]. To ensure fair comparison, we also set the upper bound on the end point in D^2-PaS as $t^u_{ep} = t^u_{sw}$. (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 Δ^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 β. For fair comparison, we set $\beta = 40$ which is one of the best values of β for the performance of OLS [8][15].

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 $\overrightarrow{P}_0$) 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.

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

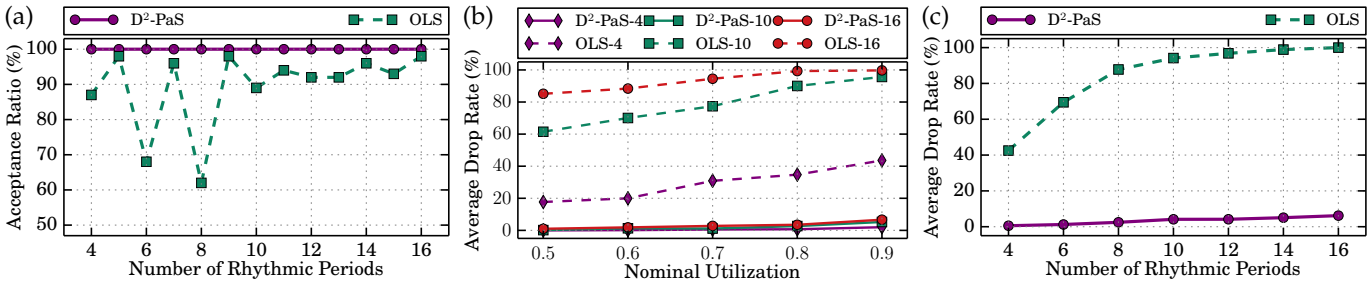


Fig. 10. Simulation results. (a) Comparison of the acceptance ratio with a nominal utilization $U^* = 0.5$; (b) Comparison of the avg. drop rate under different settings of R_0 (denoted as D^2-PaS-R_0 or OLS-R_0 in the legend); (c) Comparison of the avg. drop rate with a nominal utilization $U^* = 0.9$.

the performance of OLS while solid curves denote that of D^2-PaS. It can be clearly observed that D^2-PaS significantly outperforms OLS under all settings. As shown in Fig. 10(b), D^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^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^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^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 U^* increases from 50% to 90%, while D^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 can be rather expensive when the number of active packets and the number of switch point candidates are large. A serious consequence of high computation overhead is that it may prevent OLS from successfully constructing a dynamic schedule before the next broadcast slot and thus the rhythmic event request may not be handled in time.

8.3 Handling Concurrent Rhythmic Events

In this section, we evaluate the performance of D^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^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 programming based approach which is very time consuming, we only vary $\mathcal{N}_r$ within $\{2, 3\}$. (ii) Start point t^i_{sp} of the rhythmic window of τ_i. 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 different rhythmic event windows have partial overlaps (Case 3). We let the rhythmic window of each rhythmic event overlap with that of its previous one, i.e., $t^i_{sp} < t^{i-1}_{ep}$.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^i_{sp} is randomly picked from $\{t^{i-1}_{sp} + 1, \ldots, t^{i-1}_{ep} - 1\}$.

TABLE 5
DR of D^2-PaS for different settings of $\mathcal{N}_r$.

$\mathcal{N}_r$	1	2	3	4	5
DR	3.7%	11.3%	15.4%	17.9%	18.6%

Fig. 11(a) shows AR as a function of the number of rhythmic periods, R_i[17], under different settings of $\mathcal{N}_r$ for the task sets generated with a nominal utilization U^* of 0.9. Since both OLS and D^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^2-PaS can achieve 100% AR since a feasible end point always exists in D^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., $R_i = 4$. As observed in Fig. 11(b), increasing the No. of concurrent rhythmic events leads to degraded performance for both OLS and D^2-PaS. However, D^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^2-PaS when the number of

17. To better understand the impact from the number of rhythmic periods, we let all concurrent rhythmic tasks have a same value of R_i.

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

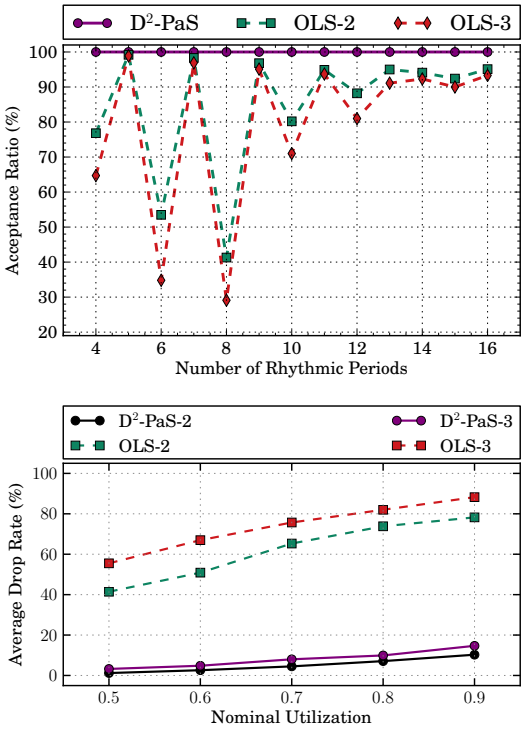


Fig. 11. Simulation results for handling concurrent disturbances under different settings of $\mathcal{N}_r$ (denoted as D^2-PaS-$\mathcal{N}_r$ and OLS-$\mathcal{N}_r$). (a) Comparison of the acceptance ratio with a nominal utilization $U^* = 0.5$; (b) Comparison of the average drop rate with $R_i = 4$.

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 $\mathcal{N}_r$ 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 D^2-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, D^2-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 D^2-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 D^2-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 D^2-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

[1] X. Hei, X. Du, S. Lin, and I. Lee, "PIPAC: Patient infusion pattern based access control scheme for wireless insulin pump system," in *INFOCOM*, 2013.
[2] K. Gatsis, A. Ribeiro, and G. Pappas, "Optimal power management in wireless control systems," in *ACC*, 2013.
[3] V. M. Karbhari and F. Ansari, "Structural health monitoring of civil infrastructure systems," CRC Press, 2009.
[4] T. L. Crenshaw, S. Hoke, A. Tirumala, and M. Caccamo, "Robust implicit edf: A wireless mac protocol for collaborative real-time systems," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 6, no. 4, p. 28, 2007.
[5] W. Shen, T. Zhang, M. Gidlund, and F. Dobslaw, "Sas-tdma: a source aware scheduling algorithm for real-time communication in industrial wireless sensor networks," *Wireless Networks*, vol. 19, no. 6, pp. 1155–1170, 2013.
[6] F. Ferrari, M. Zimmerling, L. Mottola, and L. Thiele, "Low-power wireless bus," in *SenSys*, 2012.
[7] T. Zhang, T. Gong, C. Gu, H. Ji, S. Han, Q. Deng, and X. S. Hu, "Distributed dynamic packet scheduling for handling disturbances in real-time wireless networks," in *RTAS*, 2017.
[8] S. Hong, X. S. Hu, T. Gong, and S. Han, "On-line data link layer scheduling in wireless networked control systems," in *ECRTS*, 2015.
[9] S. Han, X. Zhu, D. Chen, A. K. Mok, and M. Nixon, "Reliable and real-time communication in industrial wireless mesh networks," in *RTAS*, 2011.
[10] Q. Leng, Y.-H. Wei, S. Han, A. K. Mok, W. Zhang, and M. Tomizuka, "Improving control performance by minimizing jitter in RT-WiFi networks," in *RTSS*, 2014.
[11] A. Saifulah, C. Lu, Y. Xu, and Y. Chen, "Real-time scheduling for WirelessHART networks," in *RTSS*, 2010.
[12] M. Sha, R. Dor, G. Hackmann, C. Lu, T.-S. Kim, and T. Park, "Self-adapting mac layer for wireless sensor networks," in *RTSS*, 2013.

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

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

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

[16] L. Li, B. Hu, and M. Lemmon, "Resilient event triggered systems with limited communication," in *CDC*, 2012.

[17] G. Buttazzo, E. Bini, and D. Buttle, "Rate-adaptive tasks: Model, analysis, and design issues," in *DATE*, 2014.

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

[19] J. Lehoczky, L. Sha, and Y. Ding, "The rate monotonic scheduling algorithm: Exact characterization and average case behavior," in *RTSS*, 1989.

[20] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the ACM (JACM)*, vol. 20, no. 1, pp. 46–61, 1973.

[21] M. R. Palattella, N. Accettura, L. A. Grieco, G. Boggia, M. Dohler, and T. Engel, "On optimal scheduling in duty-cycled industrial iot applications using ieee802. 15.4 e tsch," *IEEE Sensors Journal*, vol. 13, no. 10, pp. 3655–3666, 2013.

[22] R. Soua, P. Minet, and E. Livolant, "Modesa: an optimized multichannel slot assignment for raw data convergecast in wireless sensor networks," in *IPCCC*, 2012.

[23] R. Soua, E. Livolant, and P. Minet, "Musika: A multichannel multi-sink data gathering algorithm in wireless sensor networks," in *IWCMC*, 2013.

[24] A. Tinka, T. Watteyne, and K. Pister, "A decentralized scheduling algorithm for time synchronized channel hopping," in *ADHOCNETS*, 2010.

[25] A. Morell, X. Vilajosana, J. L. Vicario, and T. Watteyne, "Label switching over ieee802. 15.4 e networks," *Trans. on Emerging Telecommunications Technologies*, vol. 24, no. 5, pp. 458–475, 2013.

[26] R. Soua, P. Minet, and E. Livolant, "Wave: a distributed scheduling algorithm for convergecast in ieee 802.15. 4e tsch networks," *Trans. on Emerging Telecommunications Technologies*, vol. 27, no. 4, pp. 557–575, 2016.

[27] S. Duquennoy, B. Al Nahas, O. Landsiedel, and T. Watteyne, "Orchestra: Robust mesh networks through autonomously scheduled tsch," in *SenSys*, 2015.

[28] E. Lawler, "New and improved algorithms for scheduling a single machine to minimize the weighted number of late jobs," *Preprint, Computer Science Division, University of California*, 1989.

[29] J. M. Moore, "An n job, one machine sequencing algorithm for minimizing the number of late jobs," *Management science*, vol. 15, no. 1, pp. 102–109, 1968.

[30] P. Baptiste, "An $\mathcal{O}(n^4)$ algorithm for preemptive scheduling of a single machine to minimize the number of late jobs," *Operations Research Letters*, vol. 24, no. 4, pp. 175–180, 1999.

[31] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. R. Kan, "Optimization and approximation in deterministic sequencing and scheduling: a survey," *Annals of discrete mathematics*, vol. 5, pp. 287–326, 1979.

[32] K. I. Ho, J. Y. Leung, and W. Wei, "Scheduling imprecise computation tasks with 0/1-constraint," *Discrete applied mathematics*, vol. 78, no. 1, pp. 117–132, 1997.

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

[34] M. Jonsson and K. Kunert, "Towards reliable wireless industrial communication with real-time guarantees," *IEEE Transactions on Industrial Informatics*, vol. 5, no. 4, pp. 429–442, 2009.

[35] M. J. Neely, "Opportunistic scheduling with worst case delay guarantees in single and multi-hop networks," in *INFOCOM*, 2011.

[36] R. Li and A. Eryilmaz, "Scheduling for end-to-end deadline-constrained traffic with reliability requirements in multihop networks," *IEEE/ACM Transactions on Networking*, vol. 20, no. 5, pp. 1649–1662, 2012.

[37] P. Soldati, H. Zhang, Z. Zou, and M. Johansson, "Optimal routing and scheduling of deadline-constrained traffic over lossy networks," in *GLOBECOM*, 2010.

[38] D. Chen, M. Nixon, and A. Mok, "Wirelesshart," in *WirelessHART*. Springer, 2010.

Tianyu Zhang received the BS degree from Qingdao University in 2010, the MS degree from Northeastern Univerity, China, in 2013. He is currently pursuing the PhD degree with the School of Computer Science and Engineering, Northeastern University, China. His research interests include real-time systems, cyber-physical systems and wireless sensor networks.

Tao Gong received the BS degree from Beihang University, China, in 2013. He is currently pursuing the Ph.D. degree in the Department of Computer Science and Engineering at the University of Connecticut. His research interests include real-time and embedded systems, industrial wireless networks, and distributed real-time data analytics.

Song Han received the BS degree from Nanjing University in 2003, the M.Phil. degree from the City University of Hong Kong in 2006, and the Ph.D. degree from the University of Texas at Austin in 2012, all in Computer Science. He is currently an assistant professor in the Department of Computer Science and Engineering at the University of Connecticut. His research interests include cyber-physical systems, real-time and embedded systems, and wireless networks.

Qingxu Deng received the Ph.D. degree in computer science from Northeastern University, Shenyang, China, in 1997. He is currently a Full Professor with the School of Computer Science and Engineering, Northeastern University, China. His research interests include reconfigurable computing systems, multiprocessor real-time scheduling, worst-case execution time (WCET) analysis and formal methods in real-time system analysis.

Xiaobo Sharon Hu received the BS degree from Tianjin University, China, the MS degree from the Polytechnic Institute of New York, and the PhD from Purdue University. She is a professor in the Dept. of Computer Science and Engineering at the University of Notre Dame. Her research interests include real-time embedded systems, low-power system design, and computing with emerging technologies. She has authored more than 250 papers in related areas.